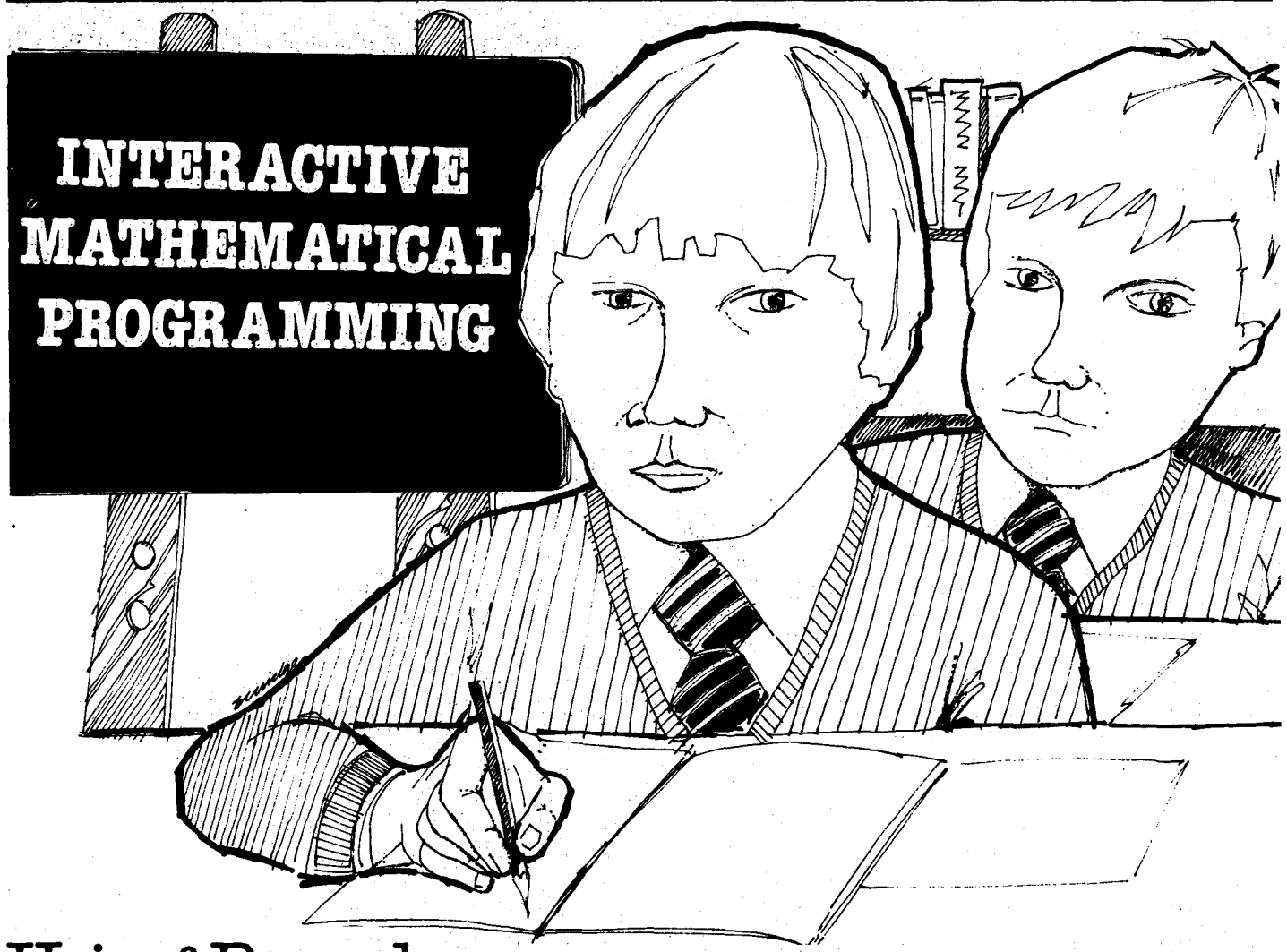


Using Pascal

The author discusses developments at his University and points out the dangers of the proliferation of educational software that has no hint of standardisation.

In this article I would like to recount some of the experiences we have had in the Pure Mathematics department at the Queen's University of Belfast in developing the classroom teaching potential of a microcomputer. In our case it was an ITT 2020, but hopefully many of the lessons we have learnt will be of interest to educational users of other machines, at other levels and, to some extent, in other subject areas.

When we purchased an ITT 2020 late in 1979, we were interested in it as a tool to be used in front of a class, as a supplement to the more traditional teaching aids of blackboard, overhead projector and duplicated handout. So far we have not investigated its potential for Computer Aided Learning (CAL) in the sense of a one-to-one relationship between a student and a microcomputer. There are possibilities for future development in this area, especially for remedial teaching, but this is more likely to take place through terminals linked to one of the University's mainframe computers. As a front-of-the-classroom teaching aid the portability of a microcomputer is essential, but that would not be needed for CAL use.

The possible uses that we had in mind were the ability to produce accurate pictures and to do calculations associated with them in the classroom. One advantage that was not so clearly foreseen, but which has turned out to be one of the most important, is the ability to allow the class to interact with the demonstration.

'Great programming care is needed with input from and output to the user.'

After seeing some examples chosen by the teacher the class can be given the opportunity to see that examples *they* choose also work, and that the teacher has not just chosen simple examples that work (students can be suspicious, especially the better ones). Once emotionally convinced of the probable truth of some assertion it is far easier to force oneself to put the effort required into mastering a formal proof. For this kind of teaching to succeed, a microcomputer seems essential; at least if graphics or complicated calculations are involved.

We are not in the business of teaching statistics or dynamics, where the potential of a microcomputer is obvious, but in that of teaching basic mathematical concepts like differentiation, convergence of sequences, etc. The microcomputer cannot teach any of the proofs involved here, at least, not as we are using it; but it can get across the basic definitions by illustrating them graphically. Thus we need to produce visually convincing graphics rather than do precise numerical calculations. The limited precision of APPLE Pascal's real arithmetic is no bother to us; the fact that on the ITT 2020 it does not plot lines of length 2 correctly is. We need to produce graphs that look right — lines should be solid and not dotted, and jumps should occur only when they are needed (and *should* occur then). The problem is much more that of putting on a show than of typical scientific computing.

Our decision to purchase an ITT 2020 was governed by the desire to have a machine with good-resolution graphics and which also had local dealer support. Our initial purchase included one disk drive and a full complement of 48K of random access memory. We ordered an APPLE language card in order to run Pascal, although it was not immediately available. After seeing this demonstrated it became obvious that a second disk drive was essential. Not only is a single drive system inconvenient to operate but

it also limits severely the size of program that can be compiled (APPLE Pascal programs can consist of up to 7 segments, including one permanently resident and so can be quite large).

At present we use a 9" monitor for the lecturer and a 12" monitor for the class. This has proved adequate for demonstrating *graphics* to a small class (about a dozen students) but more and larger monitors will be needed for bigger classes even with graphics. It does not seem worth trying to put text across, apart from writing large characters on the graphics page. Both monitors are black and white. The colour output to a domestic TV is disappointing, and even with a dedicated colour monitor it is only satisfactory for large areas of colour. A sine curve, for example, is multi-coloured (if drawn in white, dotted if drawn in red/green) unless all points are duplicated by the point to one side, which effectively halves the horizontal resolution.

After initial experiments proved promising and it had become obvious that program development time had to be cut, we purchased a printer to list programs. It has also proved useful for monitoring the output of programs and for tracing the operation of programs running in graphics mode.

If our expectations of a microcomputer are to be justified it is essential to make it part of the teaching repertoire of *all* our teaching staff, not just of a few who can program it. It is therefore essential that programs should be as easy to use as possible. One should certainly not expect the user to understand how to alter programs or change operating systems, and all output and input should be in plain language. Few appropriate programs appear to be readily available either commercially or from existing programs on University mainframe installations. What is available that might be adapted is marred by the demands it makes on the user. So we decided we had to develop our own software.

In common with most educational users, all our programming is done by members of the teaching staff in addition to their normal duties of teaching, administration and research. Program development time must therefore be cut as much as possible. When our system was purchased Pascal was not immediately available; and even when we did get it, proper TURTLEGRAPHICS (the high-resolution graphics output) was not available for the ITT 2020 for several months (it employs 9 bit bytes to store graphics, which gives better resolution than the APPLE, but the 9'th bit requires special access). Our initial programming attempts were, therefore, in PALSOFT BASIC. We wrote and tested in the classroom a general purpose graph-drawing program, one to illustrate the definition of the Riemann integral, and a third to demonstrate the uniform approximation of continuous functions by polynomials. Although they were successful, both in that they worked and were well-received in the classroom, it was clear that program development needed to be speeded.

Queen's University

It seemed likely to us that the structured nature of Pascal was likely to improve the progress of program development. The combination of using Pascal and a printer certainly has cut development time dramatically, especially the time actually spent on the machine – vital when several people use one microcomputer. The nature of the language is such that much more time must be spent designing the program before getting near a machine, but getting from that stage to a running program is much quicker. Typing errors can be eliminated in a single pass by using the quiet compilation operation (* $ Q + *), and sending the output listing to a printer. This option continues compilation after errors, so all can be corrected at the end of the run. In theory, at least, the listing could be set to a disk file and then studied later; but, in practice, it is not possible to accommodate the text, code and listed versions of a large program on a two-disk system. Between them these two factors cut down time spent on the machine and mean that much work can be done away from the machine in odd moments.

'The combination of using Pascal and a printer certainly has cut development time dramatically.'

One advantage of BASIC in writing programs for mathematical education is the ease of entering the definition of a function at run time. The easy (and lazy) way to do this is to enter the definition in a program line. There is no delay, as the BASIC is interpreted. The situation is radically different with Pascal. To alter the definition of a function in a Pascal program you must first edit the program, update the disk copy of the text version of the program, then recompile it. Apart from this expecting a lot of the user it is time consuming (up to 5 minutes for a medium-sized program). Even when this has been done you are limited to studying functions that are defined at each point that is likely to be used, as there is no facility for turning off the checking of the validity of arithmetic expressions in APPLE Pascal as there is with PALSOFT BASIC. So whether or not you can graph 360 points across the screen for (say) y = 1/x will depend on whether or not one of the points corresponds to x = 0. Clearly unsatisfactory!

There is no obvious way of checking the continuity of a function, essential for drawing good graphs. Poles can be dealt with by not plotting if values are off the screen; all other possibilities can be dealt with in BASIC by searching the string defining the function for occurrences of SGN and INT and then producing function definitions containing these (see below) — but in Pascal?

The solution (of course!?) is to interpret the function definition in the final program. The system we have adopted is to first convert the definition into reversed polish notation once and for all, using a program segment that need only be resident in memory for a short time, and then use another segment procedure to evaluate this each time it is required. The evaluation procedure, written in Pascal, is of course slower than the BASIC interpreter which is written in assembler language. However, the higher speed of routine repetitive operations using Pascal means that the same program will run slightly faster when written in Pascal than in BASIC, unless it involves extremely complicated function definitions.

In one mode the evaluation procedure adopted sets three Boolean variables to indicate whether the function is, respectively, defined, continuous and differentiable at a single value of the variable, and returns its value if it is defined. The alternative mode calculates the function's value at up to 360 evenly spaced values of the argument. This time three packed arrays of Boolean are set and an array of values calculated.

The function definitions actually to be used are BASIC ones. Where they differ from the Pascal ones they are slightly more useful to us (e.g. INT is a standard mathematical function, TRUNC is not). Also the BASIC function symbols are all three characters long, which simplifies programming slightly. The development of these routines proved surprisingly easy and was an encouraging start to this kind of programming in Pascal. (I had earlier written some upper/lower case text editing programs to gain familiarity with the structure of APPLE Pascal, its file handling facilities and graphics).

Once the problem of defining functions was dealt with, routines to draw axes for graphs, draw the graphs themselves and to gather the information needed for these (the definition of the function and ranges of x and y values) were quickly written, then combined into a general-purpose graph-drawing program. Afterwards they were built into graphics based programs to illustrate the definition of continuity, the convergence of gradients of chords to that of the tangent and to demonstrate the relationship between a function and its derivative.

The graph drawing program also allows the option of storing graphs on disk-files and adding a caption, which is useful for showing the behaviour of a sequence of functions, as the graphs may take some considerable time to draw. At present this is done by storing the data needed to draw the graph in a text file, which is rather slow. I hope later to produce a machine code picture load/same routine that will operate in a Pascal program. To produce long-term reinforce-

ment of the lesson some hard copy output is desirable for distribution to the students. This is possible as we can produce a disk file (using SCREENBIT to access the screen, 69,120 times, so rather slow) that codes the screen image for printing to a Paper Tiger printer at another site. We have also linked the ITT 2020 to an ICL 1904A and thence to a 1906S which gives us access to a graph plotter; but we are now digressing from the topic of classroom uses of a microcomputer.

Bearing in mind that potential users of programs need have no familiarity with or interest in microcomputers (and may even feel hostile towards them) good program design is essential. Thus, with BASIC programs that need to input the definition of a function, we shunned the easy approach of asking the user to type in a program line defining the function, on the grounds that even learning simple tasks like this could prove off-putting. Instead the function definition is input as a string, then POKED into a dummy function definition that has been padded out with statement separators, taking care to encode keywords correctly. It is now easy to scan the string for occurrences of INT and SGN and extract these to place in other function definitions in order to check continuity of the function.

Queen's University

One area where great programming care is needed is with input from and output to the user, especially the former. All messages output should be in plain language (plain, that is, to the average pure mathematician) and input should be expected in the same form. It is essential to allow the opportunity to correct typing errors (except for single character input, and even there verification should be sought if the outcome would be irretrievable). Thus we don't use READLN to input a number as this does not allow correction of typing errors and, unless input/output checking is off, gives an EXECUTION ERROR #10 as soon as an illegal character is typed. Instead the number is input as a string and decoded in the program. This also allows the input to be echoed to the graphics page instead of the text page, useful with graphics-oriented programs.

The writing of real numbers in Pascal also leaves something to be desired. Most pure mathematicians are not particularly number oriented and have few routine dealings with floating point notation. The Pascal fixed point format specifies a fixed number of *decimal places* to be output when the normal precision required is to specify the number of *significant figures*. Again the home-produced routine we use to do this can output to the graphics page as well as to the text page. Note also that too much precision, even when it can be justified numerically, can disguise the message. I am convinced that it is much quicker to take in the relative sizes of the sequence of numbers:

1, 20, 300, 5000

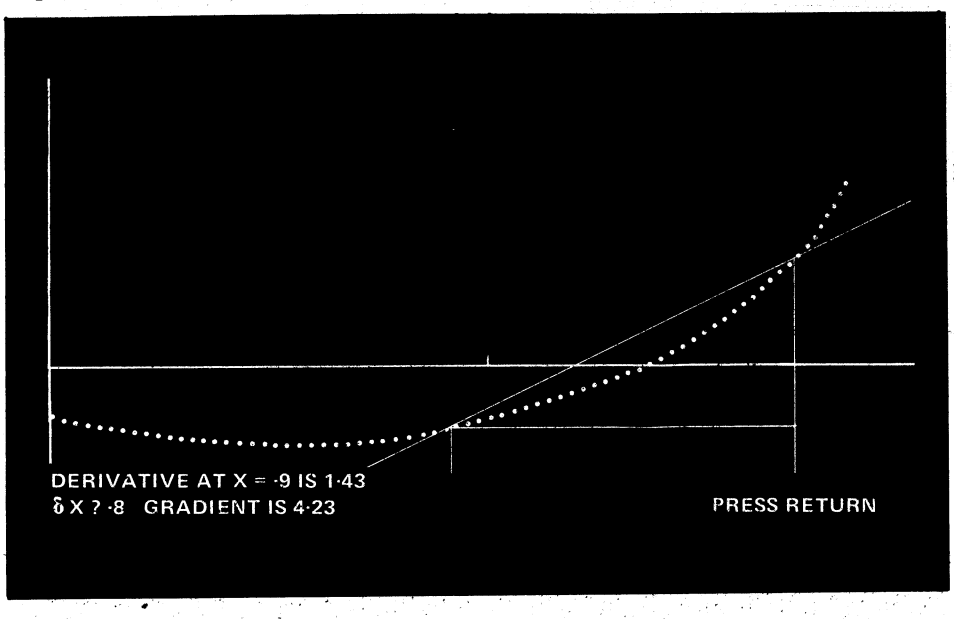
than of the sequence:

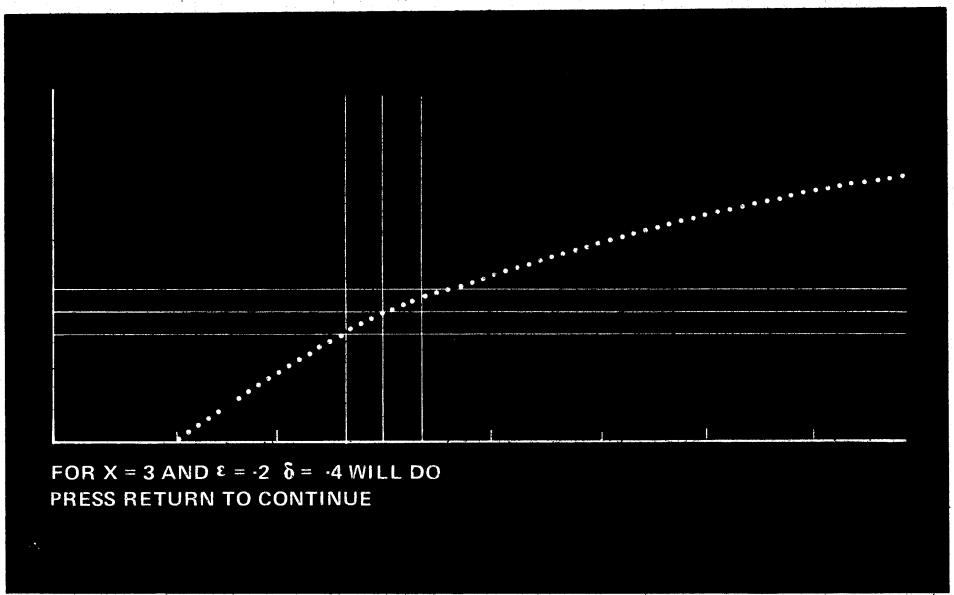
1.023, 21.742, 327.461, 5179.782.

Programs should be self-explanatory once running. Users do not want large instruction sheets to wade through before they can use a program; and programmers working in their spare time will have little desire to document their programs thoroughly. One of the advantages of Pascal over BASIC is that, as long as the program gives adequate explanation at runtime, little documentation is required. It is tempting for an author to assume that he will always be present to explain, for example, the input format needed for a particular program. But even nowadays academics do occasionally move to a different institution and the hope is that programs will be exchanged, so the program has to do it all.

Program exchange is something that must occur if there is not to be a vast duplication of effort in the Universities, Polytechnics, Teacher Training Colleges and, to some extent, the Schools (for there is overlap between 6th form teaching requirements and first year University courses, especially when we consider the needs of those University students who are not mathematical specialists). Such exchange already occurs between mainframe installations at educational establishments, but the greater physical spread of microcomputing facilities makes exchange more difficult for them. These exchanges should certainly be organised on a subject basis, but some other restriction is probably necessary. If it were to be restricted to machines of a single type it would probably prevent schools from exchanging programs with universities. Possibly a regional basis could work (especially in Northern Ireland) but then the problem arises either of translating programs or trying to standardize on a machine type. I am not sure which would be the more difficult, but certainly the problems of translation would be less between different implementations of, say, USCD Pascal than of BASIC.

Whatever the difficulties involved, some solution is essential if the present impetus is not to die under the sheer weight of effort needed to rewrite all the teaching programs desired in each and every educational establishment in the country. ■





• *Dr. Wickstead is at the Department of Pure Mathematics, The Queen's University of Belfast, Northern Ireland.*