What mathematics does programming teach? **David Tall** explains here how structured BASICs (such as the BASIC available on the BBC microcomputer) can be used to teach important ideas about functions.

Powerful mathematical functions

How will the computer be used in the mathematical classroom of the future? Clearly there is room for specially prepared software for drawing graphs, illustrating mathematical concepts, carrying out mathematical investigations and for appropriate programmed learning. But the computer itself, with a suitable programming language and no other software, is such a powerful mathematical tool that it cannot fail to cause significant changes in the mathematics curriculum. One only needs to consider the changes over the last few years to begin to imagine what might happen in the near future.

The computer revolution

Twenty years ago the study of four figure logarithm tables was an essential part of mathematics. It was quite impossible to perform many calculations without them. Five-figure tables were considered a *tour de force*. At that time the major support for numerical calculations other than tables were expensive hand- ▶

cranked mechanical calculators. I can remember demonstrating these machines when I first arrived at Warwick University fifteen years ago. Addition was quite easy, but multiplication required quite nifty techniques: moving along a place and cranking round the correct number of times. It never really caught on in schools, but many offices used them.

Ten years ago the electronic calculator had become widely distributed and provided the facility for handling logarithms and other standard functions to an accuracy of eight digits and more. Books of tables fast became mildly interesting historical documents, though they stayed in the examination syllabuses for a while longer.

The arrival of the computer in the classroom will change the scene even more dramatically in a way which cannot be ignored. A simple example will show the march of progress.

Consider the problem of adding up the first 100 terms in the harmonic sequence:

1+1/2+...+1/100.

Before the calculator this would have been almost impracticable. Logarithm tables or, better, a table of reciprocals, could be used to calculate each term but then they would have to be added up by hand or, perhaps, by "hand-crank".

With the advent of the electronic calculator most of these operations could be carried out on the machine. If the calculator had a memory, a running total could be kept and the other calculations might be effected using a reciprocal button. But it would still take an interminable time to add a hundred terms and a single mis-keying of a number would invalidate the whole job.

Programmable calculators made the matter far easier by allowing an algorithm to be written to carry out the sum. This might be similar to an algorithm for a computer using BASIC:

```
LET S=0: FORN=1 TO 100 : S=S+1/N: NEXT N :
PRINT S
```

But this is still somewhat inflexible. To change the number of terms or the formula for the term would require a change in the program. The arrival of more modern computers gives yet more flexibility through the introduction of user-defined functions. In this article we shall look at the flexibility of such functions and how they provide a powerful facility for use in mathematics.

User-defined functions

All computer languages worth their salt include routines for calculating specific functions such as sine or cosine and most versions of BASIC allow the definition of a function in the form

```
10 DEF FNV(x)=x*0.15
```

(This function multiplies the value of x by 0.15, giving the calculation of VAT at 15%. For example, FNV(3) equals 0.45.)

To calculate the gradient of the graph of f(x)=sinx+cosx from x to x+h, requires the formula

(f(x+h)−f(x))/h.

In most versions of BASIC one may type

```
1000 DEF FNf(x)=SIN(x)+COS(x)
```

and specify values such as

```
x=2:h=0.1
```

to obtain the gradient with:

```
PRINT (FNf(x+h)−FNf(x))/h.
```

In some versions of BASIC the user-defined function may not be active until the line has been executed in a program. In others, including BBC BASIC, the function becomes available as soon as the line has been typed in, turning the computer into a kind of "super-calculator".

Evaluating expressions

One of the ideas that made BBC BASIC exciting when it first appeared was the EVAL operator which is used to evaluate expressions. Thus if

```
F$="SIN(x)+x ↑ 2"
```

the numerical value of the expression may be calculated for the current value of x using EVAL(F$).

This facility is now more widely available. It is provided, for instance, in the versions of BASIC available for the Nimbus and Spectrum and in BBC COMAL (some using the term VAL instead of EVAL).

Evaluation may be used to advantage in function definitions. For instance,

```
1000 DEF FNf(x)=EVAL(F$)
```

followed by

```
F$="SIN(x)+COS(x)"
```

causes the function FNf(x) to give the value of SIN(x)+COS(x). But now we can change the function

by changing the expression F$ whenever we wish. We can type

```
x=2:h=0.1:F$="x ↑ 2"
```

followed by

```
PRINT (FNf (x+h)−FNf(x))/h
```

to give the gradient of $f(x)=x^2$ from x=2 to x+h=2.1.

This particular expression could be obtained directly by the calculation

```
PRINT (2.1 ↑ 2−2 ↑ 2)/.1
```

but conceptual power of the function definition has great advantages. Adding the lines

```
10  INPUT"f(x)="F$
20  h=.001
30  FORx=−4 TO 4 STEP 0.1
40  PRINT x, (FNf (X+h)−FNf(x))/h
50  NEXT
60  END
```

to the function definition gives a short program to calculate a table of gradients for any function between −4 to 4. This may then be used for drawing an approximate graph of the derivative, either by hand when students are becoming acquainted with the idea or, later, as part of a more sophisticated graph-drawing program that also draws numerical gradients.

Although a function definition only returns a single value, it can have any number of input parameters which may be numbers or strings. For instance one may type a second definition

```
2000 DEF FNg (x,h)=(FNf (x+h)−FNf (x))/h
```

into a computer already having the definition of FNf (x) and then

```
PRINT FNg (2, 0.1)
```

will give the gradient of the function from x=2 to x=2.1. Typing

```
FOR n=1 TO 10:PRINT FNg (2,1/2^n) :NEXT
```

will print out a sequence of values demonstrating the limiting process concerned with calculating the gradient from 2 to 2+h as h gets small.

Multi-line functions

Early implementations of BASIC limited a function definition to a single line statement. Later versions (including that on the BBC and Nimbus) allow multi-line definitions which fit naturally with more general functions. For example, a function may be given by different formulae over different parts of the domain, say

$f(x)=x\sin(1/x)$ for $x \neq 0$,

$f(0)=0$

This may be programmed as a user-defined function thus:

```
1000 DEF FNf(x):IF x=0 THEN =0 ELSE
=x*SIN(1/x)
```

The idea easily extends to a number of different formulae over different parts of the domain so that the function given by

$f(x)=1$ for $x<0$

$f(x)=\sqrt{(1-x^2)}$ for $0 \leq x \leq 1$

$f(x)=0$ for $x>1$

can be programmed as

```
4000 DEF FNf(x)
4010 IF x<0 THEN =1
4020 IF x>1 THEN =0
4030 =SQR(1−x ↑ 2)
```

or compactly in a single line as

```
4000 DEF FNf (x):IF x<0 =1 ELSE IF x>1 =0 ELSE
=SQR(1−x ↑ 2)
```

Conceptually this is extremely important in the classroom where functions are often seen as being necessarily given by a single formula. Thus f(x)=sinx is a function, but

$f(x)=\sin x$ (for $x<0$) and $f(x)=x^2$ (for $x \geq 0$)

is often considered, not as one function, but as *two* separate functions stuck together. The fact that one may write it as

```
2000 DEF FNf (x):IF x <0 THEN =SIN(x) ELSE
=x ↑ 2
```

and find that the single command PRINT FNf(x) will return appropriate function values for all x, positive or negative, is conceptually a great leap forward.

Procedural Functions

In general, a function can involve a number of ▶

intermediate calculations. A powerful function definition involving numbers and strings is:

```
10000 DEF FNsum (k,t$):s=0:FORn=1TOk:
s=s+EVAL(t$):NEXT:=s
```

This is the definition of a function FNsum (k,t$) to add up k terms of a series whoe nth term is the formula in t$. It sets the initial value of s to zero and adds on the evaluated expression t$ from n=1 to k. The final expression "=s" ends up defining the function as the value of s when all the calculations have been done. For instance, if t$="1/n" then FNsum (100,"1/n") evaluates the sum of the first 100 terms of the harmonic series, effortlessly performing a calculation that would have taken many hours away from the computer.

This flexible little program can be used to sum the first 1000 cubes as FNsum (1000, "n ↑ 3"), or the even numbers from 2 to a hundred by FNsum (50, "2*n"). By taking appropriate expressions for t$ it can be used to investigate the convergence or divergence of series.

The area under a graph y=f (x) from x=a to b may be calculated approximately by dividing the interval into n equal steps of width w=(b−a)/n. Using the mid-ordinate rule the height of the k−th strip is

```
h=f (a+(k−1/2)*w)
```

and the area of the strip is h*w.

If the expression for the function f(x) is contained in a string f$ (e.g. f$="2*x") then total area approximation from a to b using k strips can be programmed as:

```
5000 DEF FNarea (f$,a,b,n)
5010 s=0:w=(b−a)/n
5020 FORk=1TOn:x=a+(k−1/2)*w:s=s+w*EVAL
(f$):NEXT:=s
```

For instance, the approximate area under the graph f(x)=sinx from a=0 to b=π is given by

```
PRINT FNarea ("SIN(x)",0,PI,20)
```

This gives the result as 2.002. . . (compared with the value 2 found by antidifferentiation).

If one wished to concentrate on the area as a function of the number of strips, then

```
6000 DEF FNa(n)=FNarea ("SIN(x)",0,PI,n)
```

gives a mathematical *sequence*, FNa(n), of approximations to the area. For large numbers it does take a long time to calculate, but commands such as

```
FOR n=200 TO 1000 STEP 200:PRINT FNa(n) :
NEXT
```

illustrate the limiting process as the values get closer to the limiting value 2.

Alternatively, one might wish to think of the area function in a different way, say as an approximation to the definite integral of a function from a fixed point a to a variable point x. This could be done by taking a=0 and calculating an appropriate number of strips from a=0 to x. The following definition does precisely this by taking 5 steps in each interval and an extra 5 for luck (to cover the case when x is nearly zero):

```
7000 DEF FNI(f$,x)
7010 n=INT (5*ABS(x))+5
7020 =FNarea (f$,0,x,n)
```

The command PRINT FNI("2*x",2) gives 4, FNI ("2*x",3) is 9, and so on. The function is designed to work when x is negative too, so PRINT FNI("2*x",−2) also gives 4.

Conceptually this is again a great advance. It shows that the area function is indeed a *function* in the mathematical sense of the term.

There is currently a persistent belief amongst most students (and some teachers) that a "real answer" to a problem must be given as a mathematical *formula*, not a numerical *process*. This view is enshrined in the opinion that it is acceptable for the solution of a quadratic equation to be given by the standard formula, but not by an algorithm that gives a succession of better approximations. In industry, numerical methods are often used, not only for the pragmatic reason that they give a satisfactory result, but also for the more important reason that there are problems soluble by a numerical calculation which are not amenable to an analytic solution.

Recursive Functions

An oft-quoted example of a recursive function is the factorial, defined using the interpretation:

"If n=0, n! is 1, but if not, n!=n*(n−1)!".

Provided that n is a non-negative integer, this gives a definition of factorial n by working down,

```
n!=n*(n−1)!=n*(n−1)*(n−2)! . .
```

until we eventually reach 0! on the end of the product. In BBC BASIC, the definition of "n factorial" is

```
2000 DEF FNfactorial(n)
2010 IF n<>INTn OR n<0 ="not a whole number!"
2020 IF n=0 =1 ELSE = n*FNfactorial (n−1)
```

Here the condition n<>INTn OR n<0 excludes numbers which are not integers or are negative. If n=0

the value is returned as 1, or else the function is called again and again until zero is reached.

The definition could be used in the FNsum function mentioned earlier; for instance

```
PRINT FNsum(20,"1/FNfactorial (n-1)FN factorial (n-1)")
```

would give the sum of the first 20 terms in the expansion of e. To make a practical teaching point, one could obtain a good approximation to the exponential function using

```
3000 DEF FNexp(x)=FNsum(20,"x ↑ (n-1)")
```

though computer languages have more efficient methods of performing this particular calculation.

The Euclidean Algorithm

Another powerful recursive definition may be used to work out the highest common factor of two integers m,n. The method advocated by Euclid is a simple one:

If n=0 then the hcf is m

otherwise divide n into m to get remainder r

the hcf of m,n is also the hcf of r,n

Now the process can be repeated with r,n replacing m,n until a point is reached when the remainder is zero and the hcf is found. For instance, to find the hcf of 366 and 842:

Divide 366 into 842 to give remainder 120.

The hcf is the hcf of 120 and 366.

Divide 120 into 366 to give remainder 6.

The hcf is the hcf of 6 and 120.

Divide 6 into 120 to give remainder 0.

The hcf is the hcf of 0 and 6, which is 6.

In BBC BASIC the definition is

```
10000 DEF FNhcf (m,n)
10010 IF INTm<>m OR INTn<>n THEN ="not defined"
10020 IF m=0 THEN =n ELSE =FNhcf (m MODn,n)
```

Typing

```
PRINT FNhcf (366,842)
```

gives the hcf as 6. If you wish to see the calculations unfolding during the recursive process, then just insert a print statement into line 10020 by typing:

```
10020 IF m=0 THEN =n ELSE PRINT m,n :=FNhcf (m MODn,n)
```

The command PRINT FNhcf(366,842) will then give all the intermediate results before printing the highest common factor.

Broadening the idea of a mathematical function

The various facilities for user-defined functions will modify our view on the nature of a mathematical fuction. Set theory has (officially) broadened the notion already, but there are still many students in school and at university whose mental image of a function is as a single formula. Without practical experiences to the contrary this image may persist. This article has suggested practical experiences which may lead to the understanding of a function FNf(x) as a single entity, even when it uses complicated procedures to carry out the calculation.

How many students (and teachers) regard sinx as a genuine function (although it is actually calculated by a procedure using power series approximations) but are not happy to regard a numerical solution of a differential equation as a genuine function in the same way? As we become more familiar with user-defined functions this prejudice should be significantly reduced, bringing the theoretical mathematics of the classroom closer to the practical mathematics required in the real world.

Looking into the future

It is possible that in the very near future computers will have even more friendly user-defined functions that allow them to be used as a "super-calculator" in mathematics. BBC BASIC is regrettably flawed by the fact that typing in new lines of a program destroys all the current variables, seriously weakening its use in immediate mode.

It should also be manifestly obvious that it is rather a pain typing line numbers for each new function. A better-designed language would allow us to type in function definitions without line numbers and edit multi-line definitions in the same way as in Logo. There would be more power to our elbow if we could just type

```
DEFINE f(x)=SIN(x)
```

and then use f(x) as a usual mathematical function. A multi-line function could look something like this:

```
DEFINE f(x)
IF x <0 THEN =-1
IF x= 0 THEN =0
```

▶

```
ELSE =1
END DEF
```

(with carriage returns at the end of each line). It could be later displayed by a single command, say DISPLAY f, and edited by another, say EDIT f.

New versions of BASIC on the QL computer and the Nimbus, and the implementation of COMAL on the BBC still require line numbers, but improve on the notation by allowing one to dispense with the FN symbol. In such languages, once f(x) is defined, one may write

```
PRINT (f(x+h)-f(x))/h
```

instead of the cumbersome

```
PRINT (FNf(x+h)-FNf(x))/h.
```

Any other combination of user-defined functions f,g,. . . could be calculated, such as f(x)/g(x), f(g(x)) f(g(a(x)+b(x))) and so on. The power would be enormous.

One of the major issues that should be debated in *Micromath* is the design of a computer language, suitable for mathematics, which is immediately usable by beginners, yet powerful for long term expert use.

Curriculum implications

The kind of power intimated in the previous sections is irresistible. It will cause the computer to become essential mathematical equipment and bring with it the necessity to introduce mathematical aspects of computing into the 16+ and A-level mathematics syllabuses. To allow a proper integration of computers into mathematics, the mathematics classroom of the future will require the provision of a number of computers to be used as and when they are appropriate, without the necessity of advanced booking, of wheeling equipment from one room to another, or of using a specialized computer room. At the moment lack of curriculum materials and the expense of computers make this impracticable, but the continuing improvement in computer specifications and the dramatic reductions in cost make it a future reality for which we must now make active preparations. ■

David Tall
Mathematics Education Research Centre
University of Warwick