BASIC numerical mathematics

J C Mason MA, D Phil, FIMA

First published 1983

British Library Cataloguing in Publication Data

Mason, J. C.
BASIC numerical mathematics.
1. Basic (Computer program language) 2. Engineering mathematics—Computer programs
I. Title
510'.28'542 TA330

ISBN 0-408-01137-8

Typeset by Phoenix Photosetting, Chatham, Kent
Printed and bound in Great Britain by Whitstable Litho Ltd., Whitstable, Kent.

Preface

This book has two broad aims, as its title suggests: to introduce the reader to numerical mathematics, and to do so in the context of BASIC language programming. It is thus hoped to meet the needs of a wide range of engineers, mathematicians, and scientists who wish to solve practical problems on a computer.

More specifically, the book is one of a series geared to the requirements of undergraduate engineers, both in its choice of topics and in its style of presentation. In this context it has two principal tasks: to present BASIC numerical mathematics for its intrinsic value to engineers, and to provide a foundation for applications of numerical mathematics in other books of the BASIC series.

In particular the book aims to cover to a reasonable depth a selection of the most fundamental topics, namely nonlinear equations, finite differences, interpolation, differentiation and integration, as well as a variety of elementary mathematical calculations. Books on further numerical topics will appear later in the BASIC series, including a text on BASIC Matrix Methods, covering direct and indirect methods for simultaneous linear equations and applications to data fitting and function approximation.

Although it is primarily intended as an introductory text for first and second year undergraduates, the book would also serve as a reference book for advanced undergraduates, and its more elementary sections could be understood by good mathematics and science students in schools.

The book contains not only a large number of well-documented BASIC programs for executing a number of fundamental mathematical methods required by engineers, but also a fairly detailed discussion of the mathematical principles behind these methods. As a textbook it is tailor-made for a course in which numerical mathematics and computing are 'integrated', but it could also provide a means of spicing up a numerical mathematics course by adding optional computer exercises or of strengthening an engineering problems course by offering not only 'package programs' but also the mathematical foundations underlying them.

In writing the computer programs for this book certain decisions had to be made. In particular, BASIC was chosen as the programming language for the book, indeed for the whole series of books, because it is a simple and convenient language which is justifiably popular amongst engineers and widely available on minicomputers. The design of the BASIC language makes it possible to write programs, test them, modify them, and run them with very great ease. Readers should therefore be able to use and modify the programs in the book for their own use without difficulty.

However, one notable drawback of BASIC lies in its poor subroutine and subprogram facilities; indeed it is not normally possible to call upon one program within another program. This and other drawbacks are pointed out in the book as they arise. Although subprogram facilities can generally be dispensed with at the elementary level of this book, readers would find such facilities helpful if they wished to develop programs or use them within their own programs. In time such facilities may well become available in BASIC on larger computer systems.

In order to promote better understanding, individual instructions in our BASIC programs have wherever possible been written to resemble corresponding mathematical equations, occasionally at the expense of a little efficiency. The resemblance becomes clearer if the reader erases all REM statements, which are included only for explanatory purposes.

If readers wish to look up programs in the book, they will find them within the body of the text rather than before or after the relevant theory. However, complete lists of programs with titles and page references may be found in the Contents List. Programs are generally preceded by Algorithms (with the same numbering) which detail the logical steps to be carried out, and followed by Sample Runs which demonstrate the programs in action, and by Program Notes which provide necessary additional information and advice.

So as to make the programs easy to use and attractive for demonstration and teaching purposes we have designed them to be 'interactive' Following the keyboard instruction RUN, the user is asked for all relevant data, and then all input and output takes place at the keyboard. However, this also means that, if typed output is to be taken away by the user, the keyboard must be equipped with a 'hard-copy' facility. Readers who prefer to use 'batch-mode' programs, which include all their own data, may easily modify the programs by using appropriate READ and DATA statements in place of INPUT statements (see Programs 1A and 1B where such modifications are made).

As far as the mathematics in the book is concerned, we have tried to strike an appropriate balance between numerical methods and numerical analysis. It is clearly essential for engineers to have an armoury of methods at their disposal. However, they also need to know enough numerical analysis to be able to assess the relative merits of alternative methods and to understand the errors that all methods necessarily produce.

We have also tried to whet the reader's appetite in all the problem areas covered, even though in some cases the shortness of the book and the assumed inexperience of the reader has prevented us from making use of the 'best' methods available. On balance we believe that readers are better off with an imperfect method, provided that they are made aware of its limitations, than with no method at all.

A large number of problems are posed at the ends of chapters, covering both mathematical and computational aspects. Many of the questions are designed to help readers understand and use the theory and programs in the book. However, several questions are also given which aim to increase the reader's mathematical knowledge and put new perspectives on the theory, and several opportunities are given for readers to extend and develop programs to solve new problems.

It remains for us make some specific remarks about the contents. The book starts with three introductory chapters. Chapter 1 provides a brief but reasonably thorough description of BASIC, sufficient to understand and code the programs in the book. Chapter 2 gives an overall view of numerical mathematics, and discusses with simple examples all the factors which need to be taken into account in applying a numerical method to solve a mathematical problem. While Chapter 3 develops BASIC programs to perform a set of elementary mathematical calculations, several of which will be familiar to the reader from school mathematics (e.g. the solution of quadratic equations) and a number of which are used to advantage within programs in subsequent chapters.

Chapter 4 then proceeds to a thorough discussion of nonlinear algebraic equations including the bisection method, fixed point iteration and Newton's method, but with no treatment of complex roots. Finally Chapter 5 is concerned with the processing of tables of data in order to derive information from them, including the interpolation of values between tabulated values, the evaluation of derivatives (i.e. gradients), and the calculation of integrals (i.e. areas under the graph). This chapter includes a comprehensive introduction to finite differences, which are the principal tool in the analysis.

Although the book is primarily intended for undergraduate use, we suggest that introductory numerical mathematics/computing courses in schools might be based on Chapters 1 to 3, together with judiciously chosen sections of Chapters 4 and 5 (centering perhaps around the bisection and Newton methods, the Lagrange and Newton interpolation formulae, and the trapezium and Simpson integration rules).

We are grateful to colleagues at the Royal Military College of Science at Shrivenham for helping with this book in a variety of ways.

In particular we are indebted to Mrs Jan Price for preparing the manuscript with astonishing speed and precision, and to Mr D. C. Stocks, Dr M. J. Iremonger, and Mr P. D. Smith, for reading, checking or criticizing various parts of it.

Contents

Chapter 1

Introduction to BASIC

"In a few minutes a computer can make a mistake so great that it would take many men many months to equal it", according to Merle L. Meacham. Nevertheless, when such mistakes can be avoided, the computer can be an invaluable tool in the solution of mathematical problems. In this book we must therefore strive not only to understand numerical mathematics but also to write correct BASIC computer programs.

1.1 Computer programs and programming languages

A computer program is a set of instructions which a computer is able to interpret and execute. These instructions are designed to perform a particular task, in our case the numerical solution of a mathematical problem. In order that the instructions may be recognised, they must be written in a standard programming language (such as BASIC, FORTRAN, ALGOL, PASCAL, COBOL, etc) for which an 'interpreter' (or 'compiler') is available on the computer. The interpreter transforms each instruction in the programming language into a set of fundamental instructions in a 'machine language' instantly recognisable to the computer. The programming language is designed to be convenient and practicable for the user, and BASIC is one such programming language.

1.2 The BASIC approach

All of the programs in this book are written in BASIC. The name BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code, and was developed at Dartmouth College USA as a general-purpose language. The main advantages of BASIC are that it is easy to learn, convenient to use, and particularly well suited to 'conversational' programming in which the user interacts with the computer throughout the running of the program.

The simple version of BASIC used in this book has a number of disadvantages, and these mainly concern its lack of structure in comparison with languages like FORTRAN or PASCAL. For example it is not usual in BASIC to distinguish between integers and other numbers, to have variables of double length (for more accurate calculations), or to use one program as a subroutine or subprogram for another program. Moreover BASIC has a particular disadvantage in numerical analysis, which relates to its apparently commendable feature of rounding to integer values any numbers that are very close to integers. This makes it difficult to test the conditioning of any problem that has integer data, and inadvisable to use integer data as test data in gauging the rounding error in any program. However, these disadvantages are not too important a consideration for elementary programs such as those given in the following chapters.

This book is not intended as an instruction manual in BASIC. For that purpose the reader is referred to References 1–3 at the end of the Chapter or to one of many similar works. One of our aims, however, is to help the student learn BASIC by applying it to solve mathematical problems, especially those that occur in science and engineering. This aim can be met by the reader if he studies and tests the programs in the book, and also tries to write his own programs based on some of the problems given at the ends of the chapters. Although the book does not give every detail of the grammar of BASIC, a description of the main features of BASIC is given below.

1.3 The elements of BASIC

1.3.1 *Program structure and sequencing*

A BASIC program is a sequence of statements which define a procedure for the computer to follow (rather like a cooking recipe for a chef to follow). As it follows this procedure, the computer allocates values to each of the variables encountered and changes them where instructed. Statements used in the program are of a number of types, which will be discussed in more detail in following sections. They include REM statements (for making program notes), DIM statements (for allocating subscripted variables), INPUT or READ statements (for defining data), assignment statements (for doing mathematics), conditional statements (for controlling the action of the program) and PRINT statements (for printing out results).

Every statement must be preceded by a line number. On running the program, all statements are executed in the sequence that corresponds to these line numbers. For example, the program

```
100 X = 1          is executed as      100 X = 1
400 GO TO 200                          200 X = X + 1
300 PRINT X                            300 PRINT X
200 X = X + 1                          400 GO TO 200
```

The use of numbering greatly simplifies correcting and editing (see Section 1.5).

1.3.2 *Mathematical expressions*

In mathematics it is necessary to evaluate expressions which involve numerical constants, variables (e.g. x), and functions (e.g. sin). All constants are treated identically in BASIC, whether they are integer (e.g. 36) or real (e.g. 36.1). They may be entered in either fixed point form (e.g. 36.1) or floating point form (e.g. 0.361E2), although the computer prints out numbers in fixed point form unless they are small or large. The constant π is often available by typing PI or the π key, but for clarity the numerical value (3.14159265 . . .) will be used in this book unless otherwise stated.

Variables which fulfil the role of letters in algebra, may be named by any one of the letters A to Z, or by any letter followed by a digit (e.g. A3, P7, etc). Each variable is allocated a location in the computer memory, and it takes the numerical value recorded in that location. This numerical value is substituted for the corresponding variable whenever that variable occurs in an expression, and so it is important to ensure that the correct value is given to a variable initially.

The function square root may be evaluated via the built-in computer function SQR, $\sqrt{x}$ being replaced by SQR(X). The argument in brackets (X) may be any number, variable, or mathematical expression. Other built-in functions include SIN(X), COS(X), LOG(X), EXP(X), ABS(X), and INT(X) which represent, respectively, $\sin x$, $\cos x$, $\ln x$ (i.e. $\log_e x$), e^x, $|x|$, and the integer part of x. For trigonometric functions (SIN, etc) the argument is assumed to be measured in radians.

Mathematical expressions are formed from constants, variables and functions by inserting arithmetic operations such as plus, times, etc. These operations have a hierarchy, which determines the order in which they are performed by the computer, and it is as follows:

to the power of (^)
multiply (*) and divide (/)
add (+) and subtract (−).

If two or more operations have the same hierarchy, then the computer works from left to right. Brackets always take precedence and should be used to provide clarity and avoid ambiguity. The first left bracket is paired with the last right bracket, and so on. Hence

$$\frac{a+b}{3c}$$

becomes either

(A+B)/(3*C) or (A+B)/3/C.

Some examples of correct and incorrect BASIC expressions are as follows:

Mathematical expression	*Correct BASIC*	*Incorrect BASIC*
$x \sin x$	X * SIN(X)	X SIN(X)
$\frac{1-r^n}{1-r}$	(1-R^N)/(1-R)	1-R^N/1-R
$\ln(1+\sqrt{x})$	LOG (1 + SQR(X))	LOG (1 + SQR(X)
$\left\lvert\frac{1+\sin x}{x}\right\rvert$	ABS((1 + SIN(X))/X)	ABS (1 + SIN(X))/X

1.3.3 *Assignment statements*

An assignment statement takes the form

line number [LET] variable = mathematical expression

The word LET here is usually optional, and will be omitted throughout this book. Square brackets are used in this chapter to indicate optional items. For example a root of a quadratic equation

$$x_1 = (-b + \sqrt{b^2 - 4ac})/(2a)$$

may be obtained by a statement such as

```
100 X1 = (-B + SQR(B^2 - 4*A*C))/(2*A)
```

It is important, however, to realise that an assignment statement is not itself an equation. It is an instruction to give the variable on the left hand side the numerical value of the expression on the right hand side. Thus we may have a statement such as

```
50 X = X + 1
```

which increases by 1 the value of X.

There is no mathematical statement in common usage which is precisely equivalent to the assignment statement

X = Y

However, in this book we shall use the symbol ':=' to denote 'becomes', so that the precisely equivalent mathematical statement is

$x := y$

The symbol ':=' is used in place of '=' for assignment statements in the ALGOL language.

1.3.4 *Input*

In *conversational programming* the user specifies values of variables by INPUT statements at 'run-time'. The statement has form

Line number INPUT variable 1 [, variable 2, . . .]

e.g.

```
20 INPUT A, B, C
```

When the program is run the computer prints ? on reaching this statement, and waits for the user to type values for the variables, e.g. ? 5, 10, 15 which is interpreted as A = 5, B = 10, C = 15 in the above example.

An alternative form of data input is used if there are many data, or if the data are unlikely ever to be changed, or if the user does not want to converse with the computer. In this case a statement of the form

line number READ variable 1 [, variable 2, . . .]

e.g.

```
20 READ A, B, C                                (1.1)
```

is used in conjunction with a statement (or number of statements) of form

line number DATA number 1 [, number 2, . . .]

e.g. either

```
21 DATA 5, 10, 15                              (1.2)
```

or

```
21 DATA 5  )
22 DATA 10 }                                   (1.3)
23 DATA 15 )
```

On executing a READ statement, values are assigned to variables from the DATA statements in the order in which the latter occur in the program. If (1.1) is followed by either (1.2) or (1.3), then A, B and C are allocated values 5, 10, and 15.

1.3.5 *Output*

The output of data (for checking purposes) and the results of calculations etc is done by a statement of form

```
line number PRINT list
```

where the list may contain variables or expressions e.g.

```
200 PRINT A, B, C, A*B/C
```

or text enclosed in quotes e.g.

```
10 PRINT "VALUES OF A, B, C:";
```

or a mixture of both e.g.

```
300 PRINT "X = "; X, "Y = "; Y
```

The items in the lists are separated by commas or semi-colons. Commas ensure a tabulation in columns about 14 spaces wide, while a semi-colon suppresses such spacing. If a semi-colon is placed at the end of a list, it suppreses the line feed. If the list is left blank then a blank line is printed, and this is a useful way of spacing out results.

Note the necessity to use PRINT statements to copy out all numbers which are input by INPUT or READ/DATA statements, so that there is a true record of them. PRINT statements should also precede INPUT statements for explanatory purposes, since ? on its own is not informative. For example the pair of statements

```
10 PRINT "WHAT IS X";
20 INPUT X
```

leads to the computer output

```
WHAT IS X?
```

in reply to which the value of X is typed in.

1.3.6 *Conditional statements*

It is often necessary to take a course of action if, and only if, some condition is fulfilled. This is done with a statement of form line number IF expression 1 $\underset{\text{operator}}{\overset{\text{conditional}}{}}$ expression 2 THEN line number

where the possible 'conditional operators' are

=	equals
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

For example a program could contain the following statements if it is to stop when a zero value of N is input.

```
20 INPUT N
30 IF N <> 0 THEN 50
40 STOP
50 . . .
```
(1.4)

Note the statement STOP which ends the run of a program. This is not to be confused with the statement END which is the (optional) last statement occurring in the program listing.

1.3.7 *Loops*

There are several ways in which a program may be made to repeat some of its procedure, and the simplest is to use the statement

line number GO TO line number

For example, if the statement

```
60 GO TO 20
```

is added to the instructions (1.4), then the program will execute statement 50 for a sequence of input values of N until a zero is input.

The most common way of doing loops is to start with a 'FOR statement'

line number FOR variable = expression 1 TO expression 2
[STEP expression 3]

where the STEP is assumed to be unity if it is omitted, and end the loop with

line number NEXT variable

The same variable is used in both FOR and NEXT statements, and its value should not be changed in the intervening statements.

A loop is used if, for example, N sets of data X, Y have to be read and a mathematical expression such as $\sin(X + Y)$ calculated in each case. e.g.

```
10 READ N
20 PRINT "X", "Y", "SIN(X + Y)"
30 FOR I = 1 TO N
40 READ X, Y
50 PRINT X, Y, SIN(X + Y)
60 NEXT I
```

Loops may also be used to calculate sums and products of a list of expressions, and this is discussed in the following chapter in connection with the symbols Σ and Π.

1.3.8 *Subscripted variables*

It is frequently desirable in mathematics to use a variable with a subscript, such as x_i, so that many cases can be covered by a simple formula. For example, we might write

$$x_i = i^2 \qquad (i = 1, 2, 3, \ldots, 10) \tag{1.5}$$

to specify that the x_i are the squares of the integers up to 10 ($x_1 = 1, x_2 = 4, x_3 = 9, \ldots, x_{10} = 100$). In a BASIC program x_i is represented by X(I), the subscript being placed in brackets, and a specific numerical value must be assigned to I in the program, perhaps by a FOR loop. For example (1.5) may be calculated from

```
10 FOR I = 1 TO 10
20 X(I) = I^2
30 NEXT I
```

It is also permitted for a variable to have two or more subscripts attached to it. For example a matrix element a_{ij} may be represented by A(I, J).

Since a subscripted variable has more than one value associated with it (while a non-subscripted variable has just one), it is necessary to provide computer storage space for as many values as might be needed. This is done by a 'dimension statement' of the form

```
line number DIM variable 1 (integer 1)
[, variable 2 (integer 2), . . .]
```