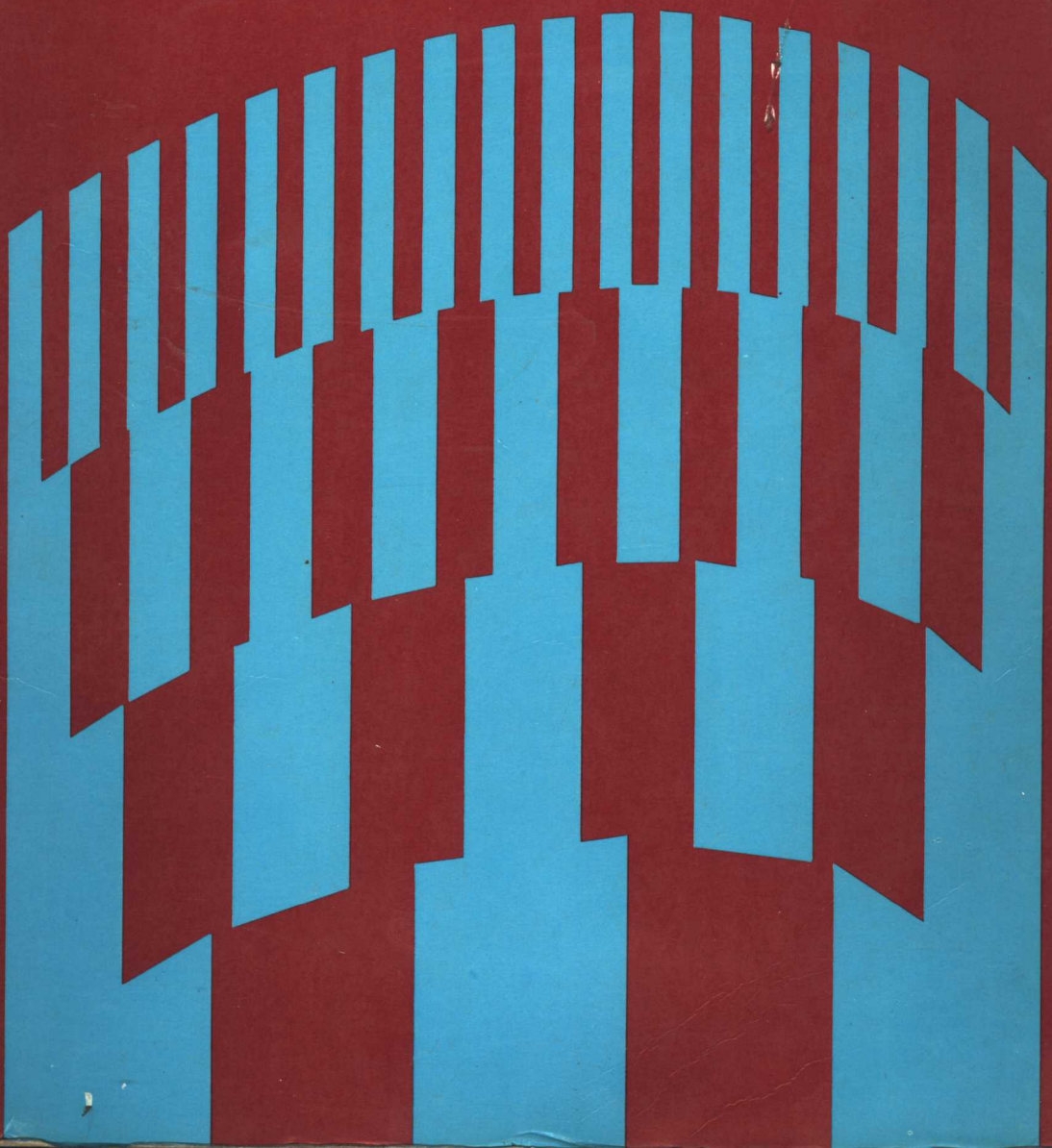
Basic Numerical Methods

An Introduction to Numerical Mathematics on a Microcomputer

R. E. Scraton



Basic Numerical Methods

An Introduction to Numerical Mathematics on a Microcomputer

R E Scraton

Reader in Numerical Analysis, University of Bradford

Edward Arnold

First published in Great Britain 1984 by
Edward Arnold (Publishers) Ltd, 41 Bedford Square, London WC1B 3DQ

Edward Arnold, 300 North Charles Street, Baltimore, Maryland 21201, USA

Edward Arnold (Australia) Pty Ltd, 80 Waverley Road, Caulfield East,
Victoria 3145, Australia

Scraton, R. E.
BASIC numerical methods.
1. Numerical analysis—Data processing
2. Basic (Computer program language)
I. Title
519.4 QA297

ISBN 0-7131-3521-2

Computer typeset by SB Datagraphics,
printed and bound by Spottiswoode Ballantyne, Colchester and London

Preface

When computers were first introduced, they were used almost entirely for the numerical solution of mathematical problems. The earliest computers were mostly to be found in the Mathematics departments of large universities. Nowadays computers, especially microcomputers, are big business. They are sold in quite large numbers to businessmen for accounting, data handling and word-processing; and in much larger numbers to children (of all ages) for games playing. Mathematicians now form an insignificant part of the total market, but they should be the last to complain about this: if computers were used only by mathematicians they would be many times their present price! For less than £200 one can now purchase machines which computer experts might haughtily dismiss as 'toys', but which are capable of serious and complex mathematical work: the author regularly uses a Sinclair Spectrum for his own research work.

One effect of the microcomputer revolution is that many budding mathematicians in their teens now have access to computing power which professional mathematicians could only dream about a few years ago. It is to these budding mathematicians, as well as budding scientists and engineers with a mathematical inclination, that this present book is addressed. Much of the book should be intelligible to the sixth-former studying A-level Mathematics, although as a whole the book is more suitable for the first year undergraduates. The author has tried to achieve a balance between computational practice and the underlying mathematical theory, but no doubt everyone has their own view on what the correct balance should be. The reader should not be afraid to skip some of the heavier mathematical work at a first reading; but must also remember that real competence in computational work cannot be attained without an understanding of the underlying theory.

It is assumed that the reader is sufficiently familiar with BASIC to understand the programs given throughout the book. These programs should be entered on the computer, tested (on as many problems as possible), and stored for future use. In these programs, REM statements have been used sparingly. It is rather tedious to type in lengthy and non-essential program lines, and in the author's experience many students simply leave them out. The most useful REM statements are those which a user inserts, which he or she feels are necessary for an understanding of the program: readers are invited to use these as liberally as they wish.

Although we have not specifically used the term in the text, we have tried to emphasise the importance of robustness in numerical methods. It is no use writing a program which works only for favourable problems. We should try to anticipate what might go wrong in less favourable circumstances. Ideally, we should like our programs to work for all relevant problems, but in numerical work we only rarely achieve this ideal; we have to be satisfied with programs which work 'nearly always'. No doubt some readers

will take delight in finding problems for which the programs fail. If they go on to ask *why* the programs did not work—was the problem ill-posed, or was the program inadequate for the job—they will learn a great deal from this exercise.

A few exercises are given at the end of each chapter. Generally these are intended to lead the reader to further understanding of the subject, rather than merely provide repetitive examples. Answers to numerical questions are given at the end of the book, as well as illustrative programs for some of the programming exercises. Some of the exercises in the later chapters require fairly lengthy programs, and the reader who can cope with these successfully is well on the way to becoming a competent numerical programmer. The author will be delighted if just a few of his readers find this more satisfying than playing space invaders!

R E SCRATON
1984

Note on BASIC

Since nearly all microcomputers work in BASIC, it might be thought that a program written for one computer could readily be transferred to another. Alas this is not so! Every manufacturer has adapted and extended BASIC for his particular machine. Many of the extensions are to handle matters such as colour, sound or high resolution graphics, which were not envisaged by the original authors of BASIC; others incorporate programming structures which are advantageous although not strictly necessary. Whatever the cause, most programs require substantial rewriting before they can be used on a different machine.

BASIC is also intended to be an easily understood language. It uses everyday words such as 'IF', 'LET' and 'PRINT', and it should be possible to read through a program like a piece of English text. Unfortunately many modern programs make extensive use of instructions such as POKE, USR, DEFFIL or VDU, which are totally meaningless except to the users of a particular machine.

All of this creates problems in a book such as the present. The programs given here are intended to be both instructive and useful: they should show the reader how a particular numerical method is implemented, as well as enable him or her to use the method on whatever computer is available. In order to achieve this we have used a subset of BASIC which is readily understood and common to most modern versions of the language. The programs were written originally for the Commodore PET computer, but they will run with minimal alteration on most other computers. On some computers the programs can be shortened, for example by

(i) omitting LET wherever it occurs,
(ii) omitting either THEN or GOTO in a statement IF . . . THEN GOTO . . .,
(iii) omitting the brackets in a function such as SQR(X),
(iv) replacing NEXT R by just NEXT.

The reader will need to know which of these changes are permissible on his or her machine; if in doubt enter the program as written. No use has been made of structures such as IF . . . THEN . . . ELSE or REPEAT . . . UNTIL, as these are not available on all machines.

In this book we have used arrays with suffices starting at zero. Some computers—notably the Sinclair Spectrum—use arrays starting with suffix 1, so some amendment is needed to programs using arrays before they can be run on these machines. The easiest way to change the programs is to increase all array suffices by one: thus DIM A(N) is replaced by DIM A(N + 1), A(R, S) is replaced by A(R + 1, S + 1), and A(0) is replaced by A(1). This should always work, though it sometimes leads to rather inelegant programs. The Sinclair Spectrum again allows only single letter array names,

so some of the arrays in Chapter 8 will need to be renamed: for instance K1(), K2(), K3() could be replaced by J(), K(), L(). Multiple dimension statements are not permitted on some machines, including the Spectrum: so DIM A(N, N + 1), X(N) must be written as DIM A(N, N + 1): DIM X(N).

It should also be realised that different machines do their arithmetic in different ways, so the numerical results given here (obtained on the PET) will not always be reproduced on other computers. Usually the difference will only be in the last significant figures, but where, for instance, we have deliberately taken the small difference between two nearly equal numbers, there could be substantial changes.

It is to be hoped that most readers will take the programs listed here as models to illustrate the implementation of a numerical procedure, and will rewrite them as necessary for their own computer. In this way they can take advantage of any permitted programming structures, and also make use of graphics or any other features they feel appropriate.

Contents

1

Numbers, Errors and Accuracy

1.1 Computer Accuracy

In the few years since microcomputers were introduced, they have become very much part of our everyday life. They may be found everywhere, from the amusement arcade to the managing director's office. Computers are especially important to the mathematician, for they enable him or her to carry out long and complicated sequences of operations which would otherwise be impossible, and this facility is no longer restricted to the privileged few who have access to an expensive main-frame computer.

A long sequence of calculations gives many opportunities for errors. There is no point in embarking on a complex calculation unless we can be reasonably sure that our answers will be correct, and that they will be sufficiently accurate for our purpose. The handbook for your microcomputer probably claims that the machine works to an accuracy of eight or nine significant figures, which is certainly accurate enough for most purposes. But this does not mean that every answer printed out by the computer is correct to this accuracy. On the Commodore PET, for instance, the instruction

```
PRINT INT(5-SQR(9))
```

gives the answer 1 instead of 2. The instruction

```
PRINT SQR(999999999)-SQR(999999998)
```

gives the answer

```
1.37Ø9Ø683E-Ø5
```

or 0.000 013 709 068 3. It is very tempting to take this answer as correct to the accuracy given, but as we shall see later the correct answer is 0.000 015 811 388 3.

At this stage you might feel tempted to return your computer to the manufacturer with a rude letter, but even the most expensive computers are capable of similar mistakes. We have to learn to use the computer in such a way as to avoid errors of this kind, and this is by no means easy. In this first chapter we shall see how to avoid some of the more common pitfalls.

1.2 How Numbers are Stored on the Computer

Most early calculators (and some early computers) worked in **fixed point** notation. They might be set, for instance, to display two figures before the decimal point and six figures after the decimal point. With this setting any answer larger than 100 would give an overflow error, whilst an answer less than 0.01 would be shown to only four significant figures. Nowadays nearly all scientific calculators work in **floating point**

notation, or more strictly in **floating point decimal**. In this notation the display

$$1.234\ 567\ 89\ +07$$

has to be interpreted as

$$1.234\ 567\ 89 \times 10^7 = 12\ 345\ 678.9.$$

These calculators can handle numbers from 10^{-99} to 10^{99}, all with the same number of significant figures. From the mathematician's viewpoint, the advantages of floating point over fixed point notation are obvious.

Computers work in **floating point binary** notation, although they display their results in fixed or floating point decimal. Most readers probably know a little about binary notation, and this will suffice for our present purpose. On most microcomputers the basic unit of storage is the **byte**, which consists of eight binary digits. We can consider a byte as holding an integer between 0 and 255 inclusive. All data held in the computer—numbers, strings or programs—are stored as combinations of bytes.

When a real number x is stored in the computer, it is first converted to the form

$$x = \pm a \times 2^b,$$

where a is a number such that $\frac{1}{2} \leqslant a < 1$ (known as the **mantissa**) and b is a positive, negative or zero integer (known as the **exponent**). Every real number (except zero) can be expressed uniquely in this form. Five bytes are used to store the number x, the first byte holding b and the remaining four holding a.

The first byte actually stores $b + 128$, so that b may take any value between -128 and $+127$. The computer can thus store numbers between 2^{-128} and 2^{127}, or roughly 2.9×10^{-39} and 1.7×10^{38}. Note that this is a smaller range than most scientific calculators can handle. The computer will stop with an appropriate error message if a number exceeds the upper limit. A number below the lower limit is simply taken as zero, with no error message, and this can sometimes cause unexpected results.

The mantissa a is stored as four bytes a_1, a_2, a_3, a_4, where

$$a = \frac{a_1}{256} + \frac{a_2}{256^2} + \frac{a_3}{256^3} + \frac{a_4}{256^4}.$$

This is very similar to the way in which we might write a decimal number to four decimal places, but here we are working in the scale of 256 instead of in the scale of 10. Since a lies between $\frac{1}{2}$ and 1, the first binary place of a_1 will always be 1, and need not be stored; instead this digit is used to store the sign of the number.

We can see that a is stored with a maximum error of $\frac{1}{2} \times 256^{-4} \simeq 1.2 \times 10^{-10}$. It is therefore correct to more than nine but less than ten decimal places. It follows that numbers are stored in the computer to an accuracy of somewhere between nine and ten significant figures.

1.3 Some Consequences of the Way Numbers are Stored

Apart from the addition, subtraction and multiplication of simple integers, nearly all operations on the computer yield only approximate answers, and it is important to be aware of this. Thus when the PET computer is asked for SQR(9) it does not

obtain the exact answer 3, but a number slightly greater than 3. We are not immediately aware of this, however, for the instruction

```
PRINT SQR(9)
```

gives the answer 3. In fact the value for SQR(9) when rounded to nine (decimal) significant figures is 3.000 000 00, which the computer prints simply as 3. On the other hand 5 − SQR(9) gives a number slightly less than 2, so INT(5 − SQR(9)) gives 1 instead of 2. Other microcomputers may give the correct answer to this problem, but it is usually possible to find similar problems on which they fail.

It is important to realise that an instruction such as

```
100 IF SQR(9)=3 THEN PRINT "OK"
```

will not cause any printing; as far as the computer is concerned, SQR(9) is *not* equal to 3. In general we should avoid instructions beginning IF X = Y unless X and Y are simple integers: even if PRINT X and PRINT Y yield the same result the computer may not accept that X and Y are equal! It is better to write

```
100 IF ABS(X-Y)<Q THEN ...
```

where, say, Q = 1 E − 9 (i.e. Q = 10^{-9}).

Another unexpected result is illustrated by the following simple program:

```
10 REM SINE TABLE 1
20 FOR X=0 TO 1 STEP .1
30 PRINT X,SIN(X)
40 NEXT X
```

This is intended to produce a table of x and sin x from $x = 0$ to $x = 1$ in steps of 0.1 (or, as we usually write, for $x = 0(0.1)1$). In fact the output from this program looks like this:

```
0           0
.1          .0998334167
.2          .198669331
.3          .29552020 7
.4          .389418342
.5          .479425539
.6          .564642473
.7          .644217687
.8          .717356091
.9          .78332691
```

You will see that the output stops short at the value .9. After each circuit or the FOR–NEXT loop, the value of X is increased by 0.1, and the computer returns for another circuit of the loop provided that this new value of X is less than or equal to 1. The number 0.1 cannot be stored exactly on the computer, and in fact it is stored as a number slightly greater than 0.1; the effect of this is to bring the above program to a premature end. It is usually safer to make sure that the step-lengths in FOR–NEXT loops are integers, thus:

```
10 REM SINE TABLE 2
20 FOR R=0 TO 10
30 LET X=R/10
40 PRINT X,SIN(X)
50 NEXT R
```

This program gives the required result.

The next program is designed to calculate the binomial coefficient

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}. \tag{1.1}$$

The subroutine starting in line 1ØØ calculates the value of M! and returns it as F; the rest of the program should be reasonably obvious.

```
1Ø REM BINOMIAL COEFFICIENT 1
2Ø INPUT"N,R";N,R
3Ø LET M=N:GOSUB 1ØØ:LET B=F
4Ø LET M=R:GOSUB 1ØØ:LET B=B/F
5Ø LET M=N-R:GOSUB 1ØØ:LET B=B/F
6Ø PRINT "BINOMIAL COEFFICIENT IS";B
7Ø STOP
1ØØ LET F=1:IF M<2 THEN RETURN
11Ø FOR S=2 TO M:LET F=F*S:NEXT S
12Ø RETURN
```

The program may be tested with small values of n and r, thus:

```
N,R? 4,2
BINOMIAL COEFFICIENT IS 6


N,R? 1Ø,3
BINOMIAL COEFFICIENT IS 12Ø


N,R? 2Ø,7
BINOMIAL COEFFICIENT IS 7752Ø
```

On the other hand, if we set $n = 35$, $r = 3$, we get the result

```
N,R? 35,3

?OVERFLOW ERROR IN 11Ø
```

even though the value of the required binomial coefficient is only 6545. The reason, of course, is that in the above program we calculate $n!$ as an intermediate result, and $35! \simeq 1.033 \times 10^{40}$, which is too large for the computer to handle. We can, however, write

$$\binom{n}{r} = \binom{n}{r}\left(\frac{n-1}{r-1}\right)\left(\frac{n-2}{r-2}\right)\cdots\left(\frac{n-r+1}{1}\right) \tag{1.2}$$

and using this formulation we get the program:

```
1Ø REM BINOMIAL COEFFICIENT 2
2Ø INPUT"N,R";N,R
3Ø LET B=1
4Ø IF R=Ø THEN GOTO 8Ø
5Ø FOR S=Ø TO R-1
6Ø LET B=B*(N-S)/(R-S)
7Ø NEXT S
8Ø PRINT "BINOMIAL COEFFICIENT IS";B
```

This gives the required result:

```
N,R? 35,3
BINOMIAL COEFFICIENT IS 6545
```

In fact this last program works for any values of n and r for which $\binom{n}{r}$ is within the capacity of the machine.

The formulae (1.1) and (1.2) are mathematically equivalent, but the latter proves to be more satisfactory from the computational viewpoint. It is often the case that the most obvious mathematical formulation is not the most satisfactory for computational work, and part of the art of numerical work is to choose the most suitable formulation.

1.4 Errors in Floating Point Arithmetic

We have already seen that a number x is stored as $\pm a \times 2^b$, where $\frac{1}{2} \leqslant a < 1$. Since a is stored to a length of four bytes (or equivalently to 32 binary places), there is a possible error in a in the interval $\pm\frac{1}{2} \times 256^{-4} = \pm 2^{-33}$. Thus the value of a stored in the computer is actually $a + E$, where $|E| \leqslant 2^{-33}$. It follows that the value of x stored in the computer is

$$\pm(a + E) \times 2^b = \pm a \times 2^b\left(1 + \frac{E}{a}\right) = x(1 + \varepsilon),$$

where

$$\varepsilon = \frac{E}{a}, \quad \text{and} \quad |\varepsilon| \leqslant \frac{1}{a} \times 2^{-33} \leqslant 2^{-32},$$

since $a \geqslant \frac{1}{2}$. So a number x is actually stored in the computer as $x(1 + \varepsilon)$, where $|\varepsilon| \leqslant \rho = 2^{-32}$. This result is true for any microcomputer which uses the five-byte representation of numbers described in Section 1.2. For other computers, or indeed for any calculator which uses floating point notation, the only difference is in the value of ρ: the smaller the value of ρ, the more accurate the machine. You might like to work out the value of ρ for your calculator.

Now suppose that we have two numbers x_1 and x_2, which are stored in the computer as $x_1(1 + \varepsilon_1)$ and $x_2(1 + \varepsilon_2)$, where $|\varepsilon_1| \leqslant \rho$ and $|\varepsilon_2| \leqslant \rho$. Suppose also that we wish to multiply these two numbers together to obtain a number $y = x_1x_2$. The value obtained for y will be

$$\begin{aligned} x_1(1 + \varepsilon_1)x_2(1 + \varepsilon_2) &= x_1x_2(1 + \varepsilon_1 + \varepsilon_2 + \varepsilon_1\varepsilon_2) \\ &\simeq y(1 + \varepsilon_1 + \varepsilon_2), \end{aligned}$$

since $\varepsilon_1\varepsilon_2$ is small compared with the other terms involved. Thus y is obtained as $y(1 + \delta)$, where $\delta \simeq \varepsilon_1 + \varepsilon_2$, and so $|\delta| \leqslant 2\rho$. This gives an upper limit to the magnitude of the error in y, but in many cases the error will be much smaller than this: for instance if ε_1 and ε_2 are of opposite signs they will partially cancel one another. Nevertheless we must allow for the possibility of a slight loss of accuracy whenever we multiply two numbers together, since the limit for $|\delta|$ is twice the limit for $|\varepsilon|$.

If, instead, we calculate $y = x_1/x_2$, the value obtained is

$$\frac{x_1(1+\varepsilon_1)}{x_2(1+\varepsilon_2)} = \frac{x_1}{x_2}(1+\varepsilon_1)(1-\varepsilon_2+\varepsilon_2^2-\varepsilon_2^3+\ldots),$$

using the binomial expansion of $(1+\varepsilon_2)^{-1}$. Ignoring terms involving products of two or more ε's, this gives

$$y(1+\varepsilon_1-\varepsilon_2).$$

Thus again y is obtained as $y(1+\delta)$, where in this case $\delta \simeq \varepsilon_1 - \varepsilon_2$. Since ε_1 and ε_2 can be either positive or negative, we again have $|\delta| \leqslant 2\rho$; thus the possible loss of accuracy on division is similar to that on multiplication.

Addition and subtraction are a little more difficult to handle. If $y = x_1 + x_2$ then y is computed as

$$x_1(1+\varepsilon_1) + x_2(1+\varepsilon_2) = x_1 + x_2 + x_1\varepsilon_1 + x_2\varepsilon_2 = y(1+\delta),$$

where

$$\delta = \frac{x_1\varepsilon_1 + x_2\varepsilon_2}{x_1 + x_2}. \tag{1.3}$$

We know that $-\rho \leqslant \varepsilon_1 \leqslant \rho$, $-\rho \leqslant \varepsilon_2 \leqslant \rho$; so if x_1 and x_2 are both positive we can write

$$-\frac{x_1\rho + x_2\rho}{x_1 + x_2} \leqslant \delta \leqslant \frac{x_1\rho + x_2\rho}{x_1 + x_2}$$

or $|\delta| \leqslant \rho$. Since the limit for $|\delta|$ is the same as the limit for $|\varepsilon|$ there is no loss of accuracy when two positive numbers are added together. We can easily extend the above argument to the case when x_1 and x_2 are both negative, but not to the case when x_1 and x_2 are of opposite signs, i.e. when we are either effectively doing a subtraction.

If x_2 is a number close to $-x_1$, the denominator $x_1 + x_2$ in equation (1.3) is nearly zero, so δ could be indefinitely large. It follows that we can get a substantial loss of accuracy when 'subtracting' two nearly equal numbers. It is probably true to say that this is the most common cause of loss of accuracy in computational work. 'Subtraction' in this sense may either be the addition of two numbers of opposite sign or the subtraction of two numbers of the same sign. Whenever we use the symbols + or − in a computer program we should remember that this could be a possible source of error!

In Section 1.1 we mentioned the evaluation of SQR(999999999) − SQR(999999998). This is a clear example of the subtraction of two nearly equal numbers. The instruction

```
PRINT SQR(999999999), SQR(999999998)
```

gives the answers

```
31622.7766          31622.7766
```

showing that the two numbers are identical to nine significant figures. Since the computer holds these numbers only to a little more than nine significant figures, it is obvious that we can expect no accuracy at all when we subtract them; but the computer

nevertheless insists on giving us a full nine-figure answer. What may appear obvious when presented in this way will be far from obvious when it occurs in the middle of a lengthy computer program; yet it could well destroy any accuracy in our final results. It is imperative that we formulate our problems so as to avoid any possibility of subtracting nearly equal numbers. We can reformulate the above problem by setting $p^2 = 999\,999\,999$, $q^2 = 999\,999\,998$, then

$$p - q = \frac{p^2 - q^2}{p + q} = \frac{1}{p + q};$$

we can thus evaluate $p - q$ without any loss of accuracy by means of the instruction

```
PRINT 1/(SQR(999999999)+SQR(999999998))
```

which gives the correct answer

```
1.58113883E-Ø5
```

1.5 Simple Interval Arithmetic

So far we have been concerned only with errors which arise in the computer, but we must be equally aware of errors in the data which we feed into the computer. If we ask the computer to evaluate 4.56/1.23 we shall get the answer 3.707 317 07, but it would be futile to quote this answer to nine significant figures if the numbers 4.56 and 1.23 are accurate only to three. If this is the case 4.56 denotes a number between 4.555 and 4.565, whilst 1.23 denotes a number between 1.225 and 1.235. Their quotient could be as small as $4.555/1.235 = 3.6883$, or as large as $4.565/1.225 = 3.7265$. Thus the answer lies between 3.6883 and 3.7265, and is accurate to rather less than three significant figures.

What we have just done is a simple example of **interval arithmetic**. It is convenient for this purpose to introduce the notation $\{p, q\}$ to mean 'a number between p and q'. We can then write

$$4.56 = \{4.555, 4.565\} \qquad 1.23 = \{1.225, 1.235\}$$

and so

$$4.56/1.23 = \{3.6883, 3.7265\}$$

$$4.56 \times 1.23 = \{5.5799, 5.6378\}$$

$$4.56 + 1.23 = \{5.78, 5.80\}$$

$$4.56 - 1.23 = \{3.32, 3.34\}.$$

Check the above answers for yourself, and make sure you understand how they are worked out. Note that in multiplication and addition we obtain the lower limit by combining 4.555 with 1.225, whereas in division and subtraction we combine 4.555 with 1.235. In every case we have to decide which combination gives the smallest possible result, and which gives the largest.

The idea is easily extended to more complicated expressions, for example

$$\begin{aligned}
&\frac{1.23 + 4.56}{2.78 \times (9.87 - 8.72)} \\
&= \frac{\{1.225, 1.235\} + \{4.555, 4.565\}}{\{2.775, 2.785\} \times (\{9.865, 9.875\} - \{8.715, 8.725\})} \\
&= \frac{\{5.78, 5.80\}}{\{2.775, 2.785\} \times \{1.14, 1.16\}} \\
&= \frac{\{5.78, 5.80\}}{\{3.1635, 3.2306\}} \\
&= \{1.7891, 1.8334\}.
\end{aligned}$$

The answer, in this case, is accurate to only two significant figures, and we should quote it as 1.8.

It is often believed that data which are accurate to three significant figures will give answers accurate to three significant figures. Whilst this may be a rough-and-ready rule in some circumstances, it is certainly not strictly true, as can be seen in the above examples. The most obvious deviation from this rule is again in the subtraction of nearly equal numbers; for example

$$\begin{aligned}
9.87 - 9.86 &= \{9.865, 9.875\} - \{9.855, 9.865\} \\
&= \{0.00, 0.02\},
\end{aligned}$$

giving virtually no accuracy in the answer. There are many other instances, for example

$$\tan 89.7° = \{\tan 89.65°, \tan 89.75°\} = \{163.7, 229.2\},$$

with an accuracy of only one significant figure.

Interval arithmetic gives us a foolproof way of assessing the possible error in data manipulation. It deals only with extreme values, and in practice most answers will lie well within the limits obtained. Clearly it is impractical to do all data manipulation using interval arithmetic, but used occasionally it can help to identify sources of error.

Exercises

1 Obtain both roots of the quadratic equation

$$x^2 - 23\,456x + 7 = 0$$

correct to nine significant figures.

2 Statisticians often need to calculate the binomial probability

$$P = \binom{n}{r} p^r (1-p)^{n-r}, \quad (0 \leqslant r \leqslant n, \quad 0 < p < 1).$$

Write a program to evaluate P for any specified values of n, r and p. Try to ensure that you get no overflow errors, and that you do not get the answer zero unless $P < 10^{-38}$.