Introduction

This is a book about techniques that have been developed in the past century or so—mostly by physicists and astronomers—for the numerical solution of various problems that arise in the physical sciences and engineering, and which have no (known) analytical solution. Although some of the methods are new, most are to be found in standard books on numerical analysis, or in technical journals such as *Journal of the Society for Industrial and Applied Mathematics*, *Journal of Computational Physics*, *Computing in Science and Engineering*, and so forth.

"I put in for reincarnation, but they said if you don't know computers forget it."

In the distant past—within the lifetime of the author, but doubtless long before most of his intended audience was born—there were no digital computers at all. Only four types of calculating machines to augment the human brain had at that time been invented:

1. the ancient Chinese abacus, that in the hands of an expert rivaled the electromechanical calculator for speed on arithmetic operations of addition, subtraction, multiplication, division or even square roots;

2. Napier's "bones" (that is, the slide rule) for low-precision multiplication, division and trigonometry;

3. the electromechanical calculator, an intricate construction of gears, cams and wheels far outstripping the most elaborate Swiss watch for complexity, and able to perform addition, subtraction and multiplication;

4. the analog computer—either mechanical or electrical—that could integrate mathematical functions as well as perform the usual operations of simple arithmetic.

Computational methods of the pre-digital computer era were therefore intended to be carried out by one or more of the above methods, or with pencil and paper. Some very remarkable and complex calculations were performed by these means in astronomy, statistical inference, and of course the nuclear fission bomb[1].

Fast, inexpensive digital computers have rendered practical calculations of a scope undreamed of by earlier generations. Algorithms[2] that would have been completely impractical using the primitive means of a generation ago are, with today's computers and programming languages, not only practical but often optimal.

Because the methods expounded in this book are intended for digitial computers of the Von Neumann type (that is, the program resides in the same memory as the data, and the calculations are performed *seriatim* on a single computer, rather than in parallel on several computers simultaneously), the subject of programming will of necessity be discussed in connection with the algorithms. In particular I shall try to illustrate good programming practice by making the accompanying illustration programs clear, structured, well-documented and maintainable—as opposed to employing tricks that speed execution at the cost of clarity. Most of the illustration programs will be in Forth, with a few in FORTRAN or BASIC. The chief reason for this is that Forth is the language I program in by preference, while I am fluent in the other two. Although I have a nodding acquaintance with Lisp and C, I would not feel competent to write programs in either that I could unblushingly present as examples of good style.

Forth is a relatively easy language to acquire. Students familiar with FORTRAN, BASIC, Pascal or C (languages that permit assignment statements in the form of mathematical formulae) should be able to follow the computational part of the Forth programs since I have extended the language by adding a FORmula TRANslator[3] that adheres to FORTRAN rules of precedence.

Thus adding a line like

```
f" a=b+c/cosh(w^3)-sinh(w) "
```

to a Forth subroutine translates the formula string to (and compiles) the Forth code

```
b F@ c F@ w F@ F^3 FCOSH F/ F+ w F@ FSINH F- a F!
```

in a manner invisible to the user (unless he or she chooses to display the decompiled result). The program that extends Forth in this way is both short and simple—that is one of the reasons I enjoy using this language, and hope the student may find it equally enjoyable.

---

1. For an account of some of the numerical techniques and computing machinery applied to the Los Alamos effort, see "Los Alamos from Below" in R.P. Feynman, *Surely You're Joking, Mr. Feynman* (W.W. Norton & Co., New York, 1985).
2. This word is derived from an Arab mathematician's name, *Al Khuwarezmi* ("the man from Khwaresm", a city in Central Asia which was destroyed by Ghengis Khan's Mongol armies).
3. The name FORTRAN is derived from FORmula TRANslator—this is my little joke. Mine translates formulas into Forth on the fly and compiles them as Forth statements.