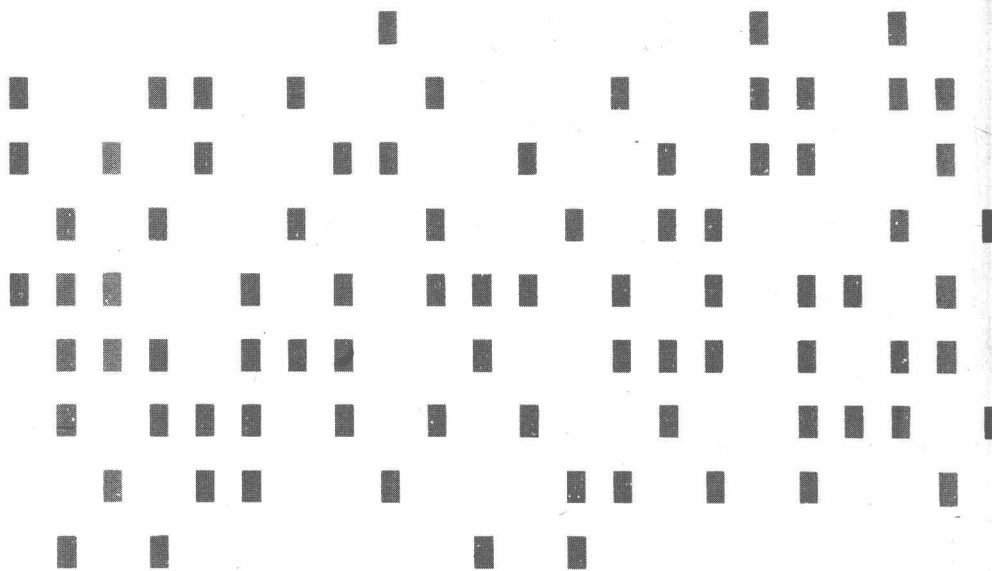


Numerical



John Wiley & Sons, Inc., New York • London

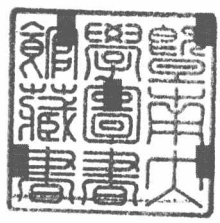
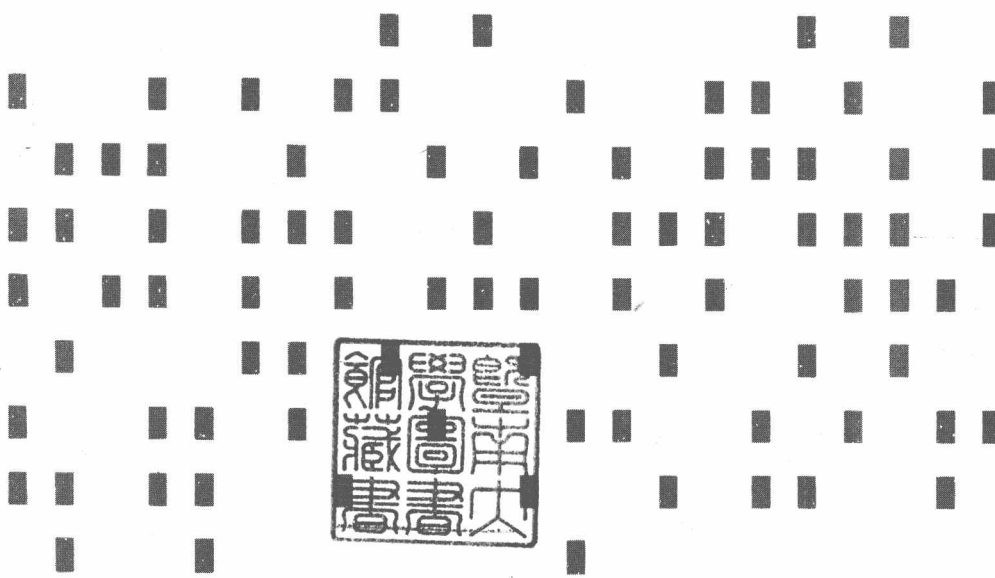
024

E709

3 23173

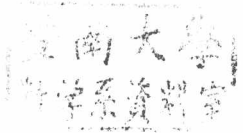
外文书库

Analysis



NATHANIEL MACON

*Professor of Mathematics and Director of Computer Laboratory
Department of Mathematics, Auburn University*



一九六四年 九月 十日

Numerical Analysis

To Alfred T. Brauer

Preface

Numerical analysis, by general definition, is the branch of mathematics concerned with developing and evaluating techniques of employing computers to solve problems. Much day-to-day computing is done by intuition and trial and error. Numerical analysis is concerned with narrowing the gap between science and artistry in computing, although that gap will never be completely closed. We seek to establish methods for obtaining answers and ascertaining their accuracy. When the numerical analyst uses his science, he can rely on the accuracy of his computations and determine whether his results are meaningful.

This book is designed as a textbook for a one-semester first course in numerical analysis or as a volume for independent study by one who wishes to grasp the rudiments of the subject. It can readily be incorporated into a course on the philosophy and techniques of computing that includes numerical analysis. Although the book was written with high-speed computation in mind, it is a mathematical textbook and not a collection of algorithms.

An introductory chapter on computers is followed by chapters on approximation by Taylor's series, iterative methods of solving equations, numerical methods of linear algebra, interpolation with polynomials, numerical differentiation and integration, and numerical solution of ordinary differential equations. Other topics, such as the uses of orthogonal functions and the numerical solution of partial differential equations, are omitted in the interest of brevity. Certain important methods are omitted because accurate and honest descriptions which can be readily presented to students with limited backgrounds in algebra and advanced calculus are difficult to formulate. The most regrettable such omission is a discussion of the Runge-Kutta methods.

The book requires only a knowledge of freshman mathematics and calculus as exemplified by the textbooks of Allendoerfer and Oakley and of Johnson and Kiokmeister, respectively. Students having some

acquaintance with elementary matrix theory, differential equations, and advanced calculus can expect to master the material in the later chapters more rapidly than those with only a minimum background. It is hoped that every reader will learn to code an automatic digital computer if he has not already done so. Coding in an algebraic language, such as FORTRAN or ALGOL, is to be preferred, since there is no need for the reader to concern himself with the difficulties or peculiarities of a specific machine.

The book provides more detailed and accurate discussions of methods than is customary in elementary textbooks on numerical methods. Laborious numerical examples are conspicuous by their absence; the student is encouraged to study a few typical algorithms critically, produce flow charts for them, code them in some algebraic language, and—if at all possible—run them on a computer.

I believe that one of the most useful skills any student of mathematics can acquire is that of constructing simple, meaningful numerical examples. Both the theoretician and the problem solver will find no easier or quicker way to gain insight into a mathematical structure, discover the cause of an unexpected phenomenon, or isolate a blunder in a code than by constructing an appropriate example. Several such examples are given early in the text, but the reader will find it increasingly necessary to supply his own as his study progresses.

Students should recognize that the mathematical literature, though necessarily concise and occasionally inscrutable, is one of the principal tools of the mathematical profession. This book is written in such a way that the reader must provide a great deal of the exposition himself, much as he would when reading a mathematical journal or treatise. Considerable care has been taken to assure that the gaps are not too great and that the material is within the grasp of anyone with a reasonable knowledge of first-year calculus. Thus it has been necessary from time to time to include material of an auxiliary character, especially in connection with the treatment of linear algebra. Because of limitations of space I have occasionally referred the reader to standard textbooks for proofs of theorems on such matters as the expansion of a determinant by cofactors and the location of the zeros of the Legendre polynomials. Suggested sources of material for further study appear frequently, since a book of this nature is necessarily limited in scope.

The overwhelming majority of numerical problems confronted in science and engineering fall into two classes. First are those which can be disposed of in a few minutes on a half dozen or so sheets of paper. The others are of sufficient complexity or length to require a high-speed computer. The greatest use of desk calculators today is probably in

facilitating the "debugging" of codes prepared for electronic computers. Although the desk calculator does not play quite the same role in modern numerical analysis as does the slide rule in a course in analytic trigonometry, it is better pedagogical strategy to ignore it completely than to risk overemphasis at the expense of the development of analytical skills and feeling for what automatic computers can and cannot do.

Perhaps unfortunately, there is very little emphasis in this book on physical motivation and on interpretation of results in terms of scientific problems. In most cases the instructor will find it a simple matter to provide a heuristic discussion in the course of his classroom lectures.

NATHANIEL MACON

The Hague, Netherlands
January, 1963

1

Basic Concepts

It is our purpose in this chapter to review briefly the working of high-speed electronic computers. The subject is too complicated to allow a complete or rigorous discussion in a short space. We hope only to present a few fundamentals that affect the character of the problems and methods we are to study.

We hope that the reader will learn to code at least one automatic digital computing machine. We can learn to code any machine without actually seeing it; in fact, programmers usually begin coding a new machine as soon as the designers have decided on a machine language, so that a number of programs will be completed by the time the first machine is built and ready for use. Learning to code will help us to understand what computers can and cannot do and to realize that the use of a computer requires precise formulation of the problem and complete accuracy in the formulation of the code.

There are two major classes of electronic computers, *analog* computers and *digital* computers. Our interest is directed almost entirely toward digital computers. The analog computer, simpler in concept, is dealt with briefly in the next section.

ELECTRONIC ANALOG COMPUTERS

Analog computers do not work with numbers themselves; numbers are represented in them as measurements of continuous physical variables such as voltages across terminals or lengths of rods. It is often possible, by using physical laws, to represent a complicated mathematical relation in a relatively simple mechanism. Such a mechanism is called an analog computer.

As an example we consider a crude device for solving equations of the form

$$x^3 + x - k = 0$$

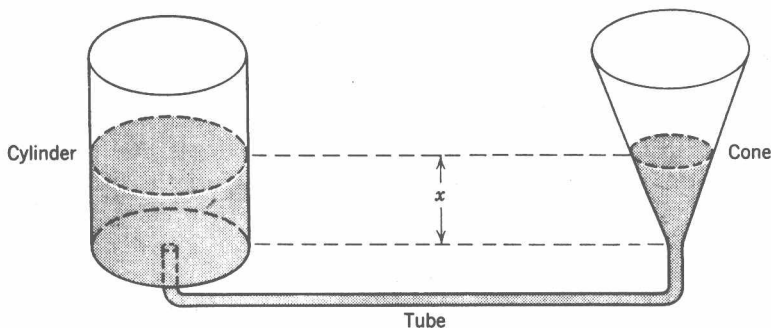
where k is a given constant. The device is not electronic, since, having been developed in 1898, it predates electronics as we know it today. The inventor, A. Demanet, made use of the fact that the volume of a cone of proper taper is equal to x^3 cubic centimeters, where x is the altitude of the cone in centimeters, and that the volume of a cylinder of unit cross-sectional area is equal to x cubic centimeters, where x is again the altitude in centimeters.

Demanet's computer consists of a cone having its radius and altitude in the ratio $\sqrt{3/\pi}$ and a cylinder of radius $1/\sqrt{\pi}$ centimeters connected by a tube as in the accompanying figure. When k cubic centimeters of liquid are poured into one of the vessels, the equation for the total volume of liquid in both vessels is

$$x^3 + x = k$$

and the common depth x to which the liquid settles is a root of the equation

$$x^3 + x - k = 0$$



This rudimentary computer illustrates some important characteristics of analog computers. It is quite simple and can be inexpensively constructed; on the other hand, it has a rather limited range of application and is of little use in solving other problems. Finally, it has an inherent limitation: Processes of measurement introduce errors into the computation. Even if the cone and cylinder were perfectly constructed, inaccuracies in the measurement of the input variable (the volume of liquid) and the output variable (the depth of liquid in the containers) would remain.

In an electronic analog computer various physical or numerical quantities are usually represented by voltages. These voltages are operated upon by computing circuits within the machine. Circuits are available that will multiply a voltage by a positive constant, invert the sign of a voltage, generate a voltage proportional to the sum of two or more voltages, generate a voltage proportional to the product of two input voltages, and generate a voltage proportional to the integral of an input voltage with respect to

time. It is up to the operator of the computer to see that these circuits are connected in such a way that they represent the problem he wishes to have solved.

In summary, an analog computer is a machine that represents numbers by continuously varying physical magnitudes, such as voltages or depths of liquid in containers. Each step or operation is performed by a separate unit, all units operating simultaneously. Many analog computers are characterized by simplicity and comparatively low cost. Versatility and high precision are often lacking. Within the limitations of a given computer, however, it is possible for the operator to set up complex problems rather quickly and vary the input conditions at will.

ELECTRONIC DIGITAL COMPUTERS

We now leave analog computers and turn to a discussion of digital computers, which contrast greatly with analog computers. Digital computers are more complicated but more versatile; their applications range from the simplest clerical operations to the most difficult scientific calculations. Computers range in size from portable, battery-operated machines to giant systems weighing many tons. They range in cost from a few thousand to several million dollars. (In some circles the "megabuck" is taken to be the unit of cost.) Despite these enormous variations, digital computers tend to differ from each other in degree rather than fundamentals and for the most part our discussion is applicable to all sizes and types.

Digital computers are so named because they deal with symbols (or digits) rather than with continuously varying voltages or other quantities. The symbols differ from those people ordinarily use because information must be represented in a form that is legible to the computer mechanism. The choice of symbols and their meaning is made by the designers of a computer. The important thing is that information relayed to a machine be represented by symbols which form a language of communication between people and machines.

A digital computing system consists of three types of functional unit: Input-output devices, storage devices, and central processing units.

The input-output devices serve as a channel of communication between the computer and the outside world, which may consist of people or, in the case of a control operation, other machines of various kinds. The input-output system may be thought of as a translator which speaks both the language of the digital computer and the language of the outside world.

The storage devices constitute a sort of electronic filing system. Symbols are read into storage by input units and are then available for processing. Each location, position, or section of storage is numbered so that stored

symbols can be located by the computer as needed. The computer may rearrange the symbols stored in it, and it may also take the original symbols from storage, calculate new ones, and place the results back in storage. Symbols can be transferred from storage to an output unit for printing or other display. The capacity or size of the storage is an important factor in determining the size and cost of a digital computing system. In some computers millions of symbols may be stored simultaneously; in others, storage of only a thousand or so is possible. Access time, or the average time required for a unit of information to be transferred from storage to the central processing unit, also varies widely. In older machines particularly access time is rather long and is the principal factor limiting the speed of the machine.

Central processing units are mechanisms that carry out the manipulation of symbols once they are read into storage. A typical unit consists of two parts:

1. An arithmetic and logic unit
2. A control unit

An arithmetic unit is capable of performing the ordinary operations of arithmetic when it receives an instruction to do so from a control unit. It also has a degree of logical ability—the ability to test various conditions, such as the relative size of two numbers in storage, and to take action called for by the outcome of the test. Control units are responsible for the initiation and monitoring of the operation of the computer. Their function is to cause the machine to perform specified operations on specified groups of symbols in a specified sequence. In the process, control units maintain a constant flow of symbols between the storage and the arithmetic units, activate circuits in the arithmetic units, and start or stop the action of input and output devices, all according to a procedure originated by the human operators of the computer. Such a procedure is called a program.

STORED PROGRAMS

We have seen that the operation of a digital computing system involves a number of planned steps spelled out by the human operators of the machine. Each step must be written out in terms of operations that the computer can perform. Each operation is coded as an instruction in a form that can be interpreted by the computer and is placed in the storage unit as a part of a stored program.

The necessity of storing programs is almost obvious. The basic unit of time in most modern computers is the microsecond—one millionth part of a second. Operations take place at a rate of perhaps several hundred thousand each second. It is essential that instructions be available to the

central processing unit at comparable speeds, and this is possible only when fully automatic control is provided, the program is stored internally, and the computer has access to instructions at electronic speeds.

The symbols stored in a computer are partitioned into definite groups or entities called *words*. Associated with each word is its location in storage, or its *address*. The addresses of the words are analogous to the page numbers in this book—they specify the place in which certain information is to be found but are not normally a part of the information itself. Thus a word consists of a group of symbols located in storage that are comprehensible to the central processing unit.

These words are subject to one of two interpretations by the central processing unit; a numerical interpretation or an operational interpretation. When a word is subject to a numerical interpretation, the group of symbols constituting the word is decoded in some fashion specified by the designers of the computer. The method used to symbolize data is known as a code. This code symbolizes a set of digits, an algebraic sign (plus or minus), and the location of a decimal point. Two forms of numerical indication are common. In one a sequence of significant digits, or mantissa, an algebraic sign, and an exponent are encoded. For example, a configuration like $-314159(1)$ might stand for -0.314159×10^1 or -3.14159 . Using this scheme the number 50310.2 or 0.503102×10^5 would be written $+503102(5)$. Since the location of the decimal point is freely variable, the scheme is said to use a *floating point*. If on the other hand the location of the decimal point is assumed to be the same for all numerical words, the scheme is said to use a *fixed point*. The bother of keeping track of the location of the decimal point is left to the program, and thus to the programmer.

When words are transferred from storage to the control section of a central processing unit the symbols are interpreted as instructions. These instructions are used by the control section to determine the interplay among storage units, arithmetic and logic units, and input-output devices. Each instruction consists of two or more parts:

1. An operation part that designates read, write, add, multiply, subtract, test sign, move data, and so on.
2. An address part that designates the address of the information or device needed for the specified operation.

Every operation the computer can perform has a unique code symbol which is to be found in the operation part of an instruction, where it can be interpreted by a control unit. Each word in storage has an address associated with it which specifies its location in a storage unit. The address part of an instruction does not contain the information proper, only the address where it is to be found.

The normal sequence of operation of a computer is as follows. All words to be required during a computation, data and instructions, are read into storage by an input unit. The computer locates the first instruction it is to perform either on command from a console or by looking in a pre-determined location in storage assigned for this purpose. This first instruction is executed. The computer then locates the next instruction and executes it.¹ This process continues automatically until the computer encounters an instruction telling it to stop.

The distinction between instructions and data in storage is made only by whether a word is brought into an arithmetic and logic section of the central processing unit or into a control section. Consequently the computer can operate on its own instructions if they are brought into the arithmetic and logic unit as data, and it can be programmed to alter its own instructions according to conditions encountered during the course of computation. It is this ability to process instructions that provides the remarkable flexibility and "logical ability" of stored-program computers.

This ability means that a single set of instructions can be used to operate on multiple sets of data stored in different locations. As each group of data is processed, the program is modified so that it subsequently refers to the next group. It also allows for the selection of alternatives in the program on the basis of conditions arising during processing. To illustrate, suppose that a calculation involves the evaluation of the function

$$y = x \quad \text{for } x < 1$$

$$y = -2x + \frac{3}{2} \quad \text{for } x \geq 1$$

This can be done by instructing the machine to select one of two sequences of instructions according as x is less than or not less than 1.

DEVELOPING A PROGRAM

Despite the diversity of the problems some form of digital computation can attack, the course of preparing a problem for computation tends to vary rather little from problem to problem. There is of course great variation among problems in difficulty and sophistication, and there is a great

¹ Location of the next instruction may be done in a variety of ways. In some machines instructions contain an "instruction address" giving the location of the next instruction in the sequence. In others, instructions are stored sequentially so that each instruction has an address one unit higher than the preceding. The pattern of sequential selection of instructions may be altered at any point by means of a "branch" instruction analogous to the "continued on page thus and so" instruction given to magazine readers when the sequence of pagination is broken.

deal of overlap and interplay among the various stages of solution, which we outline next.

The first and often the most difficult stage in preparing a program is the formulation of the problem. What are the objectives of the program? Is there any reason to suppose that these objectives can be reached with the means we have at hand? How can the language of the problem be translated into mathematical terms?

Numerical analysis begins after the problem has been formulated and perhaps after we have decided whether or not the problem has a solution and whether or not it may have more than one. We are then faced with the reduction of the mathematical problem to a numerical procedure—with numerical analysis. Here we may find relevant mathematical theorems with proofs or references to proofs, an error analysis, a discussion of the circumstances under which the procedure may be expected to perform well or poorly, comparisons with other techniques, and references to relevant literature.

Although the subsequent steps in preparing a problem for a computer are not an essential part of numerical analysis it is important to understand them. Numerical analysis, like all mathematics, is influenced to an extent by its applications, one which is computing.

The numerical analyst investigates a numerical method; the programmer describes it; the computer executes it. *Programming* is translating a numerical procedure, clearly and unambiguously, into a sequence of commands that can be read and executed by a digital computer. In most situations the programmer has a choice of languages in which to write his description. Specifically, the natural language of the machine is always available. In addition there are a number of artificial languages, such as FORTRAN² and ALGOL.³

A machine language statement normally appears in a numeric form, such as

21 07000 09800

The first two digits, 21, may be interpreted as an operation code and the remaining as two addresses, 07000 and 09800. The machine might interpret the command as “add the number stored in the location whose address is 07000 to the number stored in the location whose address is 09800.” Presumably the result would then be available in some definite place in an arithmetic unit. It is not uncommon to find that operation codes have been

² D. D. McCracken, *A Guide to FORTRAN Programming*, John Wiley, New York, 1961.

³ H. Bottenbruch, “Structure and Use of ALGOL 60,” *Jour. A.C.M.*, vol. 9 (1962), 161–221.

assigned mnemonic equivalents, such as ADD for 21. The machine, on receiving a command in its mnemonic form, can be instructed to consult a table and assign the corresponding numeric code.

It is evident that conversion of a mathematical discussion complete with derivations and proofs into a sequence of machine language commands can be a prodigious undertaking. The artificial languages are a product of our efforts to utilize the speed and logical ability of a computer to reduce the task.

The ability of a computing machine to manipulate symbols is brought into play by means of a *compiler*. A compiler is a sequence of machine language instructions which, when executed, will translate a procedure written in a given artificial language, the *source program*, into an equivalent procedure written in machine language, the *object program*. The object program may be executed to produce "answers."

A program written in an artificial language consists of a sequence of *statements*. Although two types of statements appear in such a language, we are concerned with only one of them, executable statements. They describe the procedure. The other type of statement is used in relating facts about the source program. For example, the END statement, which is last statement in any FORTRAN program, indicates that the end of the source program has been reached. By contrast, a STOP statement is executable and is eventually translated into an equivalent machine language command.

The executable statements in the FORTRAN language are arithmetic statements, input-output statements, and control statements. A typical *arithmetic statement* is

$$X = (A + B)/(C * B) - 4.1 * A ** C \quad (1)$$

In algebraic language this statement is equivalent to $X = ((A + B)/CB) - 4.1A^c$, where A, B, and C are the names of constants stored in a computer. The statement should properly be called a substitution statement; the FORTRAN compiler will generate from it a sequence of machine language commands which, if executed, will result in replacement of the current value of X by the value of the expression to the right of the equals sign. In addition to the substitution symbol (the equals sign), we have the symbols for the basic arithmetic operations indicated by +, -, *, and /, and exponentiation indicated by **. The compiler also provides for the use of certain common mathematical functions, provided their preassigned names are used. For example, the statement $X = \text{SQRTF}(Y)$ will cause the compiler to generate instructions which on execution will replace the current value of X by the positive square root of the current value of the variable Y as calculated by a square-root program which has already been written and

incorporated in the compiler. A typical list of such functions, along with their names, might include:

Function	Name (FORTRAN)
Square root of x	SQRTF(X)
Exponential e^x	EXPF(X)
Natural logarithm $\log x$	LOGF(X)
Sine function (x in radian measure)	SINF(X)
Cosine function	COSF(X)
Arctangent (result in radian measure)	ATANF(X)
Absolute value $ x $	ABSF(X)

Control statements are the logic statements in the FORTRAN language. Their primary use is in controlling the sequence in which commands are executed. For this purpose statement numbers are introduced to those statements that may be referenced by control statements. The numbers are positive integers, not necessarily consecutive, written at the left of the FORTRAN statements. As an example of a control statement, the command GO TO 23 will result in generation of a machine language branch instruction transferring control to the first of the sequence of machine language commands corresponding to statement number 23. The statement IF(ATANF(X) - B ** X) 5, 10, 15 will be interpreted as "if $\arctan X - B^X < 0$, go to statement 5; if it equals zero, go to statement 10; if $\arctan X - B^X > 0$, go to statement 15." To a great degree the usefulness of an artificial language depends on the flexibility and variety of the control statements available.

The *input-output* statements of a language are used to assign names to variables to be read or written, their formats on cards or tape, and to specify the input-output devices to be used. For instance, READ 1 A, B, C might mean "read three numbers from a card according to the formats specified in statement 1. The locations in which they are stored are assigned the names A, B, and C, respectively." Other such instructions are WRITE, PRINT, READ INPUT TAPE, and WRITE TAPE. A programming manual should be consulted for details.

In summary, the sequence of events that takes place in using an artificial language is as follows. The procedure is described in the artificial language, and the description is read into a computer and translated, by means of a compiler program, into an object program. This machine language program may then be executed to effect the procedure.

FLOW CHARTS

The transition from numerical analysis to programming can generally be facilitated by a flow chart. Not only do detailed lists of mathematical