MASTER

THE USE OF MAN-COMPUTER INTERACTION
IN ROOT-FINDING AND NUMERICAL PROBLEM-SOLVING

Ph. D. Thesis Submitted to Iowa State University, August 1970

Gerald David Ripley and R. J. Lambert

Ames Laboratory, USAEC

Iowa State University

Ames, Iowa 50010

PREPARED FOR THE U. S. ATOMIC ENERGY
COMMISSION UNDER CONTRACT NO. W-7405-eng-82

Date Transmitted: October 1970

LEGAL NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

IS-2416

LEGAL NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

Available from

Clearinghouse for Federal Scientific and Technical Information

National Bureau of Standards, U.S. Department of Commerce

Springfield, Virginia 22151

Price: Printed Copy \$3.00; Microfiche \$0.65

THE USE OF MAN-COMPUTER INTERACTION
IN ROOT-FINDING AND NUMERICAL PROBLEM-SOLVING

Gerald David Ripley and R. J. Lambert

ABSTRACT

The current man-computer communication gap is examined in this research. Causes of this gap are identified and current approaches to the solution of this problem are reviewed.

The main portion of this research concerns the design and development of problem-oriented (high-level operator) interactive problem-solving systems. There are several reasons for attempting to formalize such systems, which until now have mainly been built as the need arose on an ad hoc basis. The design specifications for such a system should aid in identifying the true characteristics and value of such interactive systems. From a more practical standpoint, such a "blueprint" should be invaluable to those wishing to build an interactive system for a specific application.

The design specifications are independent of implementation language and computer hardware. The type of system being described is a highly modular system consisting of a number of modules, or "basic elements". The user may increase the "degree of automation" of the system by defining new "user-

defined" elements in terms of basic or other user-defined elements. The system program is designed to be readable and easily extendible by other programmers.

Finally, a graphic high-level interactive root-finding system was implemented as an example of the above type of system. The system is written in PL/I and is implemented on an IBM 360/65 computer. An IBM 2260 alphanumeric cathode ray tube serves as the on-line terminal of the system.

A number of equations solved with this system (i.e., values of X found such that $f(X) = 0$, where f is a real-valued function of one real variable) are given, along with comparisons with an existing automatic (non-interactive) root-finding system.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
CHAPTER 1 INTRODUCTION	1
Overview	1
The Communications Gap and Resulting Problems	2
Current Work	5
Automatic systems	7
Interactive systems	9
The problem of comparison	15
CHAPTER 2 DESIGN OF A HIGH-LEVEL INTERACTIVE PROBLEM-SOLVING SYSTEM	19
Introduction	19
Structure of a High-Level System	20
and the elements of an interactive high-level operator	24
Elements	24
Element properties	26
Mathematical elements	27
Investigative elements	28
Bookkeeping and special elements	29
Element extensions	29
Communication	32
Implementation Considerations	34
Hardware	35
Software	37
Programming Considerations	40
CHAPTER 3 AN INTERACTIVE ROOT-FINDING SYSTEM	43
Goals of the Root-Finding System	43
The Implementation	46
Elements	49

"Goals of the Root-Finding System" by defining new "..."

	Page
Mathematical elements	49
Investigative elements	55
Bookkeeping and special elements	56
Element extensions	61
Communication	65
CHAPTER 4 EXAMPLES USING THE ROOT-FINDING SYSTEM	67
CHAPTER 5 CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK	131
Conclusions	131
Future Work	133
BIBLIOGRAPHY	135
ACKNOWLEDGMENTS	140
APPENDIX A: TUTORIAL INFORMATION FOR THE ROOT-FINDING SYSTEM	141
APPENDIX B: ADDITION OF A NEW ELEMENT TO THE SYSTEM	162

CHAPTER 1

INTRODUCTION

Overview

Anyone who has had contact with today's electronic computer has been astounded at the tremendous power of this machine. Indeed the ability to carry out hundreds of thousands or even millions of arithmetic operations per second is alone enough to stagger the mind. Certainly the computer is a revolutionary machine in many fields of science today, and promises to leave few areas of human endeavor untouched by its capabilities.

And yet for those who have had direct contact with a scientific computer--scientists and engineers, programmers and analysts--awareness of such power has often been accompanied by a great deal of frustration concerning the manner in which we must communicate with this machine.

It is the thesis of this author that this communication gap need not exist--that the scientist or engineer can, via a terminal connected directly to a computer, effectively use the full power of the computer toward his goals.

The purposes of this research are threefold:

- (1) Examination of the communication gap, its causes, and current attempts to narrow this gap.
- (2) Specification of the properties of a high-level interactive problem-solving system,

independent of application.

- (3) Implementation of an interactive root-finding system, described in the manner set forth above.

The process of solving a numerical problem with a computer is discussed in this chapter. The problems in this process are identified, followed by a review of current approaches to their solutions.

The Communications Gap and Resulting Problems

The standard way of attempting to solve a numerical problem with a computer is to first select an appropriate algorithm. This selection should be done on the basis of what is known about the problem. For example sparseness and size of the coefficient matrix may influence selection of an algorithm for solving a system of linear equations.

After the algorithm is selected and programmed the problem is coded into a form acceptable by the algorithm routine. Such items as error tolerance, time and size estimates are specified by the user. Then the input data and the algorithm are submitted to the computer operations desk.

Then the wait begins. The user must push all his thoughts about this problem from the front of his mind. Such thoughts include excitement of getting the problem solved and how the solution will be used, and anxieties concerning choice of algorithms. Was that time estimate large enough,

or was it too large? Was all the input data punched in the right card columns? (A requirement trivial to him but paramount to the computer.)

His program is eventually run and returned to him. If his problem was an easy one, and if the algorithm was in fact appropriate and is able to handle easy problems well, and if his input data was properly formulated, and if a number of other details went well, such as time and size estimate, system operation, user's ability to interpret the output, then the problem is solved, and the user is happy and now tries to retrieve the mental state he was in when the problem arose.

What is more likely to happen, of course, is that at least one of the above "if"s will not be true. Hence a frustrating series of computer runs follows, with each run preceded by attempts to determine what went wrong and how to fix it. These "debugging" runs involve traces of program flow and data dumps placed deeper and deeper into the program as debugging proceeds. In other words, the user is actually engaged in interactive batch problem-solving.

The scientist or engineer in this predicament is often not an analyst or a programmer and quite reasonably has no intentions of temporarily becoming one. His numerical problem is most likely not his final hurdle, rather a temporary "roadblock" in his overall problem. The longer this roadblock remains, the more damage is likely to occur

to the scientist's problem-solving ability. His image of his overall problem, how it is progressing, and momentum and its ability to generate insight will all be affected. And this is not to mention the effect on the scientist of an increasing amount of disgust (for the computer, the programmer, the analyst, or even himself) generated after each unsuccessful computer run and analysis and repair session. The programmer often becomes a "patch artist", doing only what is necessary to eliminate the error messages. This frustrating mode of problem-solving also affects the analyst. Both the batch and automatic features make it very difficult and time-consuming to experiment with an algorithm and its input data to try to improve the algorithm.

This situation is ably identified and commented on by Licklider (1965):

In the general run of computer applications today, the heuristic aspects of problem solving are almost wholly separated from the algorithm aspects. The heuristic contributions are made by human problem solvers, before their programs get into a computer. Then the heuristic contributions cease abruptly, and the execution of algorithms begins. This extreme separation of the two aspects of information processing would be a source of amazement, I think, if conventional digital computing were examined carefully by a student of problem solving who had not been conditioned by the development of computing during the last 20 years.

Of course this is not the first time the man-computer communication problem has been recognized. Indeed, its recognition has preceded such significant advances as assembly

language, higher level "problem-oriented" languages, and translation-time and run-time error checking and correcting. As in the past efforts to improve this problem will be based on the following belief: the man-computer communications gap should be narrowed by extending the computer's ability to communicate with man, rather than vice-versa.

Current Work

In looking at this communication gap closely, two distinct problems emerge: time lag between attempts to solve a problem, and the strict, unbending demands of most computer programs. Examples of the latter range from incorrectly inputted data to nonconvergence of an algorithm on a particular problem.

The problem of time lag is being partially solved by the use of remote terminals and time-sharing.

A remote terminal is a communication device, such as a typewriter, teletypewriter, or cathode ray tube, connected (usually either by direct wire or by telephone) to a central computer. By being remote from the computer, a terminal can be conveniently located near the user's place of work.

Time-sharing is the sharing of the computer's capabilities with a number of users. Time-sharing is typically implemented today by "time-slicing", where each user has the attention of the computer's control processing unit for a short fixed length of time. The tremendous speed at which

a computer operates compared to human reaction time often makes it appear as though each terminal user has complete control of the computer. For example, if a time slice were defined to be 0.0001 seconds for a given computer, then 20 concurrent terminal users would be "serviced" by the computer approximately 500 times per second $((10,000 \text{ slices/second}) / 20 \text{ users})$.

Penalties paid for time-sharing include the time and space required by the computer (as opposed to the human) to shift from one user's problem to another. The benefits include instant feedback for each user, economic individual use of a large computer, and, as we will see, an entire new mode of problem solving--online interactive computing.

The second major problem in the man-computer communication gap concerns the strict demands placed on a user by most computer programs. The problem to be solved must be inputted error-free and in format usually precisely specified by the program. A more demanding requirement is that the user must select a routine able to solve the problem completely (with no errors in the process, such as division by zero). Often such a priori estimates as execution time, storage space, and convergence criteria are inadequate, resulting in wasted computer and human time.

Two different approaches to this problem of inflexibility are currently being pursued.

Automatic systems

The first approach is the building of a fully automatic program (or collection of programs, or "polyalgorithm" (Rice, 1967)) to completely handle any problem in a given class of problems, such as systems of linear equations. Such a collection of programs may be implemented for an offline (batch) environment, or for an online (usually terminal-accessed) environment. An excellent paper on the construction of such automatic programs has been written by Rice (1967). Seitz et al. (1968) also discuss goals of automatic systems in their paper on the description of one such system, AMTRAN.

Typically the user of such an automatic system inputs his problem in the form required by the system, possibly with some additional information. Thus, depending on how natural the input allowed and how versatile the system is, the user might specify to, say, an automatic root-finding system, one of the following:

(a) $F = X * \text{LOG}(X) - 1$

or (b) FIND THE ZEROES OF $F(X) = X * \text{LOG}(X) - 1$

or (c) FIND 1 ZERO OF $F(X) = X * \text{LOG}(X) - 1$

USING THE SECANT METHOD

WITH $\text{EPS} = 1/10^{**20}$ IN NO MORE THAN 100 ITERATIONS

where * means multiplication and ** means exponentiation.

Automatic systems and languages with high level operators have, besides their individual good points, several advantages

in common. Since the routines select the method(s) to use in solving the problem, these systems are mathematically "user-independent"--i.e. the user need not know anything about the method of solution. Yet those systems which allow optional qualifying input (e.g., NAPSS - see Rice and Rosen (1966)) allow the user who knows what he wants to so specify. These systems can be easy to learn, depending on how natural the input is, i.e., how close input is to the original notation of the problem.

No human intervention is required in an automatic problem solving routine, which is certainly more convenient than a routine requiring human interaction. Designers of interactive systems will often run into such complaints as "I don't have time to interact today", or, "I don't feel up to interacting today", as well as a host of other human factors problems.

It should finally be noted that automatic problem-solving systems are in many cases a desirable goal. Such cases include situations where the automatic system works correctly and there is little to be learned by human intervention in the process.

Automatic problem-solving systems and offline languages with automatic problem-solving facilities include MIRFAC, Gawlik (1963); POLYHOOK, Champagne (1965); AUTOMAST, Ball and Berns (1966); NAPSS Polyalgorithm for automatic solution of non-linear equations, Rice (1969a); POSE, Schlesinger and

Sashkin (1967); A Programming Language for Linear Algebra, Burley (1967); Automatic Integration Package for Ordinary Differential Equations, Dill et al. (1968); and PDEL, Cardenas and Karplus (1970).

Interactive systems

The second approach taken in attempting to reduce the severe requirements of today's problem-solving routines is through interactive systems. An excellent discussion of this area and survey of interactive systems as of 1966 has been written by Mills (1967).

An interactive system is by implication on-line, and so provides instant response and feedback to the user. It may also, depending on its design objectives, allow the user to interact during, not just before and after, attempted solutions to a numerical problem. The idea in such systems is that the user can sometimes do better than a strictly automatic routine and can often learn more about his problem and its solution by interacting.

Although a number of people have declared the human to be an essential active element in problem solving processes (see for example, Marchuk and Yershov (1966) and Smith (1969)), possibly no one has said it as well as Licklider (1965) in his article on "Man-Computer Partnership."

Bright humans shine in the setting of goals, the generation of hypotheses, the selection of criteria - the problem-solving phases in which one has to lay down the guide lines, choose approaches,

follow intuition, exercise judgment or make an evaluation. These aspects are called heuristic, meaning that they lead toward or facilitate invention or discovery (Newell and Simon (1964)). Both the heuristic and the algorithmic aspects of problem solving are seen throughout science and technology and wherever problems have to be solved or decisions have to be made. They are indeed, the complimentary aspects of thought.

Because of the forced separation of heuristic from algorithmic aspects, conventional digital computing is limited in application to those problem areas in which such separation can be made. Those areas of application are extensive. However, the domains in which the separations can be made easily are essentially the domains in which the problems have already been solved. Along the frontier of technology, of science, of intellectual understanding, there are areas in which the separation of ways to handle information can be made only with great difficulty, and there are areas in which it cannot be made at all.

On the frontier, man must often chart his course by stars he has never seen. Rarely does one recognize or discover a complex problem, formulate it, and lay out a procedure that will solve it--all in one great flash of insight. Usually it is necessary to go through several or many steps of planning, formulating, calculating, evaluating, and replanning--sometimes progressing, sometimes retreating to mount a new attack, sometimes bogging down in what may seem to be endless iteration or recursion or search before hitting upon the path that leads to satisfaction. Heuristic and algorithmic activities are tightly intermeshed.

More specifically, in another article Licklider and Clark (1962) identify the unique virtues of man and of the computer:

The fundamental aim in designing a man-computer symbiosis is to exploit the complementation that exists between human capabilities and present computer capabilities:

- a. To select goals and criteria--human;

- b. to formulate questions and hypotheses--human;
- c. To select approaches--human;
- d. To detect relevance--human;
- e. To recognize patterns and objects--human;
- f. To handle unforeseen and low-probability exigencies--human;
- g. To store large quantities of information--human and computer; with high precision--computer;
- h. To retrieve information rapidly--human and computer; with high precision--computer;
- i. To calculate rapidly and accurately--computer;
- j. To build up progressively a repertoire of procedures without suffering loss due to interference or lack of use--computer.

Miller (1969) gives a list of pertinent human capabilities in his article entitled "Archetypes in Man-Computer Problem-Solving".

Often times the human contribution to the numerical problem-solving process is largely one of pattern-recognition, a process at which the computer is currently very poor. There are, however, several general situations where useful interaction can be identified:

- (1) Type of problems where some degree of interaction seems to be vital, such as curve-fitting (Smith, 1969) and root-finding (this dissertation, Chapter 3).
- (2) When an automatic routine is having trouble, interaction allows monitoring and detection of

the problem. By monitoring results from varying data and algorithm parameters, interaction becomes a valuable tool for improving an automatic routine (Burgess, 1965).

- (3) Interaction can prove useful in building an automatic routine, such as in several of the phases identified by Rice (1967) in construction of an automatic routine:

- development of strategy,
- experimentation with a variety of realistic problems,
- development of common sense, and
- extensive testing and refinement.

The amount of interaction varies from one interactive system to the next. Generally speaking, the more interaction allowed by a system, the lower the level of communication. This is not to imply that a high-level operator cannot contain interaction. For versatility, however, the tendency is to allow most interaction to occur between rather than during execution of high-level operators.

The most interactive system gives the user access to all or nearly all the computer's capabilities. A system might, for example, be an interactive assembly language allowing execution of statements immediately after they are defined, and hence allowing a great deal of interaction with the computer (i.e., as often as between execution of each machine

language instruction). At the other extreme an interactive system might allow only high-level operations, such as a linear algebra system allowing only matrix inversion, eigenvalue extraction, and determinant evaluation. Some systems are somewhere between these two extremes, allowing certain high-level operators as well as some basic programming capability. (Such operator levels are also noticeable in off-line systems.)

These two extremes of high and low degree of interaction, often referred to as procedure-oriented and problem-oriented systems, generally correspond to a wide and narrow range of applications, respectively. They also generally correspond to a longer and shorter length of time required for a user to learn them, respectively. This is because (a) the system of high-level operators generally has fewer operators than the more versatile basic-programming type system, and (b) the high level system generally has more natural, or problem-oriented, input notation since it is usually defined for a narrow range of applications. These and other general characteristics are summarized in Table 1.1.

Input to an interactive system is usually handled in one of two ways. It can be compiled, which means the input is translated into machine language, and then executed by the computer. Compilation may be done entirely by the interactive system, or by an existing compiler after the input is first translated into source code acceptable by the compiler.

Table 1.1. Characteristics of low-level and high-level interactive systems

Low-level systems	High-level systems
Great amount of interaction and versatility	Narrow range of applications
Wide range of applications	Short learning time
Allows on-line extensions	Natural notation
Large amount of programming required	Little or no programming required
	Fast execution of high-level operations
	Short development time

For example, a system might translate input into FORTRAN and call a standard FORTRAN compiler.

Input to an interactive system may also be interpreted, which means no machine code is generated. Rather the input itself, or the input translated into an intermediate code, is executed by the system. Interpreted code typically executes more slowly, is translated more easily, and is dynamically changed more easily than compiled code.

A number of low-level or procedure-oriented interactive languages have been developed, including JOSS, Shaw (1964); APL, Iverson (1962); and CPS, IBM (1969a).

The following are some interactive systems consisting mainly of high-level operators: Machine-aided design of context-free grammars, Evans (1965); a collection of graphic

statistical programs, Dixon (1967); COMSTAT, Edwin and Edwin (1968); and PEG, Smith (1969).

There are several interactive systems which are somewhere between the two extremes of low and high-level operator systems. Such systems contain some low and some high-level operators: Culler-Fried system, Culler and Fried (1964); MAP, Kaplow et al. (1966); NAPSS, Rice and Rosen (1966); and AMTRAN, Seitz et al (1968).

Numerical applications using graphical output, and sometimes graphical input have been developed in several areas, including the following:

Statistics (Dixon, 1967)

Data Fitting (Smith, 1969)

Complex root-finding (Larkin, 1964).

These authors cite a real need for graphic display during the problem-solving process.

The problem of comparison

Very little study has been done on which type of system should be built and when, whether it be automatic or interactive, and off-line or on-line. Another pertinent question concerns the user--which system should he use? Although few answers are available, a few results will be presented, along with many things that a builder of a problem-solving system must consider.

Sackman (1968) discusses time-sharing versus batch

processing with low-level programming systems and has summarized and commented on five independent experimental studies made of subjects programming in both the batch and online modes. He points out that these five studies are very dissimilar and that a number of important factors are not held constant from one study to the next. Such factors include subject background (mathematical and computer), nature and degree of difficulty of the problems the subjects were to solve, the computer system, and programming languages used. Nevertheless, the results of these studies are "all we have" and provide some interesting statistics. These results are summarized in Table 1.2.

Sackman points out that very little has been done on the following topics vital to the off-line vs. on-line question:

- (1) human creativity,
- (2) distinctive characteristics of conversational interaction vs. fast batch systems,
- (3) error analysis of user performance on-line and off-line, and
- (4) case histories on the real-time pattern of problem solving.

Even if it is decided to build an interactive system, there are many considerations facing the system designer. Nickerson et al. (1968) discuss various human factors and the design of time-sharing systems. Among the problems he

Table 1.2. Summary of studies by various investigators comparing programming in a time-sharing and batch environment (Sackman, 1968)

Investigator(s)	Man-hours	Computer time	Costs	User preference
Erikson	Time-sharing ^a 1.9:1	Time-sharing 3.4:1	Time-sharing	Time-sharing
Gold	Time-sharing 1.2:1	Batch 5.7:1	Approx. same	Time-sharing
Grant and Sackman	Time-sharing 1.6:1	Batch 1.4:1	Approx. same	Time-sharing
Schatzoff, Tsao and Wing	Batch 2.1:1	Time-sharing 1.1:1	Batch 1.5:1	Not reported
Smith	Instant ^b 1.2:1	Batch 1.5:1	Approx. same	Instant
Median for all studies	Time-sharing 1.2:1	Batch 1.4:1	Approx. same	Time-sharing preferred

^aThis entry shows an advantage for time-sharing and indicates that 1.9 times as many man-hours are needed for same job in batch mode.

^b"Instant" batch is treated in this table as a simulated version of time-sharing.

mentions are the following:

psychological principles guiding the design of interactive languages,

whether there should be one or several problem-solving languages,

if there should be several languages, how to establish problem classes or user types,

how much English a language should include,

computer system response time versus user productivity,

charging algorithms for use of the system,
problems of servicing both frequent and
infrequent users, and
maximum number of concurrent users allowed.

Some important considerations which affect either the
prospective builder or prospective user of a system (or
both) are:

- (1) system development time,
- (2) simplicity and "understandability" of
the system itself,
- (3) user convenience, learning time,
- (4) execution time - both user's and computer's,
and
- (5) amount of additional information to be
gained from interaction.

CHAPTER 2

DESIGN OF A HIGH-LEVEL INTERACTIVE PROBLEM-SOLVING SYSTEM

Introduction

This chapter concerns the design and development of problem-oriented (high-level operator) interactive problem-solving systems. There are several reasons for attempting to formalize such systems, which until now have mainly been built as the need arose on an ad hoc basis. The design specifications for such a system should aid in identifying the true characteristics and value of such interactive systems. From a more practical standpoint, such a "blueprint" should be invaluable to those wishing to build an interactive system for a specific application.

Emphasis on design rather than on specific implementations is important in light of the dynamic and heterogeneous properties of the computing industry. Such properties include programming languages, personnel, and computer hardware including remote terminals.

Specifically, the design should have the following properties:

- (1) Language independence, although certain characteristics of the system implementation language will be classified as either required or desirable.
- (2) Equipment independence. All that should be

required is an appropriate terminal (although time-sharing is usually implied because of the economic factor).

The implemented system should be

- (1) of a high level operator type, easy to use by non-programmers, where the user does most of the guidance through the problem solution phase and yet has a capability to vary the degree of automation of the system;
- (2) programmed such that it is easily readable and understandable by other programmers for ease of design and maintenance of the system;
- (3) easily extendible, including the addition of high-level operators and change in any storage requirements.

In this chapter the structure of a type of high-level interactive problem-solving system is given. The "element approach" of this type of system is discussed in general and some element design criteria are given. Aspects of communication in such a system are discussed, and the last two sections present some implementation and programming considerations.

Structure of a High-Level System

The type of high-level system to be described here is one of highly modular design. Among other assets, a modular

system offers

- (a) greater readability and understandability by other programmers,
- (b) easy extension of the system, and
- (c) a degree of automation which the user can vary by combining these modules into new "user-defined" modules.

This modular approach is one way of allowing a "variable degree of automation", although other ways are possible (see the discussion on element extensions in the "Elements" section in this chapter).

The system consists of three components (see Figure 2.1):

- (1) a control routine,
- (2) a user display routine, and
- (3) a set of modules, or "basic elements".

The control routine handles most communications with the user (other than any interaction internal to a basic element) and initiates element executions. The control routine decodes a command and any associated parameters input by the user. This is an order to execute either a basic or a user-defined element. The control routine then branches to some code which does any required pre-call preparation, such as assigning parameter default values, and then executes the desired element (usually a separate routine). Following execution of an element, control always returns to the control routine. Figure 2.2 shows the flow of action within the control routine.

Figure 2.1. General flowchart of the interactive system

Figure 2.2. Flowchart of control routine (flow is downward unless indicated)

The user display routine is the interface between the rest of the system and the user's terminal. Messages to the user (system output) and commands and responses from the user (system input) pass through the user display routine. This is usually a small routine designed for a specific type of terminal and is of little concern to the system builder.

The basic elements make up the basic set of tools of the system. Each element is used by the user to do a specific task, such as input a function, display a matrix, or solve a differential equation by a certain method. These elements are determined by the particular type of problem the system is to solve, for example, solution of ordinary differential equations. Although not shown in Figure 2.1, basic elements may internally contain interaction. Elements are discussed in more detail in the next section.

Elements

The basic elements of an interactive problem-solving system are the high-level operators of the system. They are used in a manner similar to the use of low-level operators in low-level systems. That is, each operator (element) operates on its operands (input parameters) and produces certain resulting values (output values). Table 2.1 shows some examples of operators.

These elements are programmed by the system builder when he has determined what basic tools the user will need.

Table 2.1. Some low-level and high-level operators

Operator	Level	Operands	Results
+	Low	X,Y	Algebraic sum of X and Y
COS	Low	X	Cosine of X
TUTOR	High		Display of system tutorial information
HI	High	XMIN,XMAX EPS,MAXITERS	Root candidate X from using half-interval method to find a root in the interval [XMIN, XMAX], number of iterations, total number of function evaluations, error flag

The beauty of this "element approach" is that each element is completely described to both the rest of the system and the user by a few simple characteristics, or properties. Hence such an interactive system is completely defined by a concise table which lists each element and its properties. This also allows new elements to be easily added to the system as the need arises.

The properties which characterize an element are listed below. Then the types of elements in an interactive system are discussed. These element types are mathematical, investigative, bookkeeping and special, and element extensions.

Element properties

The following are the types of properties which completely describe an element to the rest of the system:

- (1) Algebraic inputs to and outputs from an element.

Names, data types, and possible default values for the input parameters are specified. An "error flag" should be included in the outputs if there are any Type 1 errors that may occur during execution of the element.

- (2) Errors. Two types of errors can occur during execution of an element, Type 1 and Type 2.

Type 1 errors are those which the designer of the element wishes to intercept and handle specially. An example of a Type 1 error is when an element which applies the half-interval root-finding method is given improper starting values (no sign change in the function). Another Type 1 error might be improper input supplied during interaction within an element. Detection of Type 1 errors is explicitly coded into the element by the designer. Any output error flag values are also set in this manner.

Type 2 errors are those detected by the computer system or the programming language which the element designer does not wish to handle specially. A Type 2 error might be attempted division by

zero. A type 2 error abnormally terminates execution of an element, control returns to the control routine and an error message is formed and displayed.

- (3) Displays. These are any interactive displays for interaction occurring within an element and output displays, including any error messages. Displays are discussed in more detail in the section on "Communication".

These properties completely describe an element to the rest of the system; and when augmented with a description of what the element does, these properties also completely describe the element to the user.

Mathematical elements

These are the elements which will be directly used to solve the type of problem for which the system is designed. The problem solution phase is first identified as a sequence of basic steps. Then one or more methods, or algorithms, to solve each step are selected by the system designer. Each such method is coded as a basic element of the system. In this type of interactive system the user is to be able to increase the degree of automation of the system by combining basic elements rather than by, for example, increasing the degree of automation within an element. Hence, the basic elements of the system should tend to be simple in

nature as opposed to an element containing much interaction, several algorithms, and complex logic. Also for this reason an element is usually an indivisible unit--that is, enterable only at the beginning of the element, rather than somewhere in the middle. If, however, enterability is allowed other than at the beginning of an element, then each such entry point will be treated as just another element. The designer should keep in mind that the user is to do most of the guidance through the problem solution phase, and that the complexity often found in a non-interactive routine may not be necessary.

Investigative elements

Included as basic elements should be certain investigative and verification tools. An investigative tool might be, for example, the ability to plot a function or an indicator of the ill-conditioning of a system of linear equations. Another such tool might be a SHOW element for displaying certain algebraic values such as element output values. A verification tool might be a matrix-vector multiplication to compare $A \cdot X$ with B in solving the system of equations $A \cdot X = B$ or a certain vector norm to check $||A \cdot X - B||$.

Bookkeeping and special elements

In any system there will be a need for certain bookkeeping elements. Examples of these might be

- (a) a SUMMARY element which displays a current summary of results of the solution process
- (b) a TIMER element to set or check a timer to allow the user to keep track of how much time he has used
- (c) a STOP element for shutting the system down
- (d) a TUTOR element to describe the system.

The system may also require certain special elements, such as input or display of a matrix, or definition of a function. Such special elements are, of course, strictly a function of the particular application.

Element extensions

An element extension, or user-defined element, is an element defined in terms of other basic or user-defined elements. This gives the user a capability of dynamically varying the degree of automation of the system. And while the solution process can be automated in this way in any area and to any degree, the basic elements are always available. This allows the user to increase the computer's role in the solution process at any place or time, while still retaining the ability to execute single basic elements when required.

This, of course, is not the only way of allowing a variable degree of automation in an interactive system. A system might allow, for example, the user to increase or decrease the amount of automation within an element. However, the element extension is a simple yet powerful tool, and gives a great deal of versatility to an interactive high-level problem-solving system. A system must be able to accommodate a wide range of user backgrounds--backgrounds including both mathematical experience and experience with the problem-solving system (see Nickerson et al., 1968).

The way in which a user can define his own elements is specified by the system designer through a syntactical and semantic description. The system requires a translator to convert description of a user-defined element into an intermediate code. An interpreter for this code is also required. The translator may be obtained by using, for example, the notion of a compiler-compiler (see Schorre (1964) and O'Neil (1968)). A compiler-compiler is a program used to generate a new compiler, or language translator. Input to a compiler-compiler is the syntactic and semantic description of the language for which a compiler is desired. Output from a compiler-compiler is a compiler for the new language.

In the case under consideration, the new language is the one with which the user will write new elements. The generated compiler will itself become an element in the system, referred to by the user when he wishes to define a

new element. The generated compiler will actually be a translator rather than a compiler since its output is intermediate code rather than machine code. This intermediate code is then interpreted, whenever the defined element is referenced by the user, by an interpreter contained in the control routine.

There are other ways to generate such an element extension translator (see Chapter 3, this dissertation), although the compiler-compiler promises to be a powerful tool for this application. It requires a formal syntactic and semantic description of the new language. Also any change in the language need only be followed by a run of the compiler-compiler to generate a new translator. Kulsrud (1968) uses this approach in the generation of a graphic language. An overview of certain compiler-compilers has been written by Theys (1970). An excellent survey of these techniques can be found in the Feldman and Gries (1968) article, "Translator Writing Systems".

It should be noted that an element translator is almost entirely independent of the type of problem being solved, and hence independent of the particular system in which it is used. Virtually the identical translator can be used in any system which uses the same extension language.

Communication

Communication with the user is a topic of great importance in the design of an interactive system. Nowhere is the problem of satisfying both the inexperienced and the experienced user greater than in the area of user communication. This area consists of specification of user input, element output and interaction messages, Type 1 and Type 2 error messages, and system tutorial information.

The type of system being described is to be easily used by scientists and engineers--people who are working directly in the problem area. Hence input as natural as possible should be the designer's goal. Little has been done in the area of the use of an existing natural language such as English in man-computer communication. Fraser (1967) reports on some work done in this area, although much work needs to be done.

A problem that occurs in specifying legal user inputs is that of spelling. To solve this problem a system may attempt spelling error correction or may ask the user to select his input from a "menu" of all possible legal inputs.

The most important point in design of input is that input should be convenient for the user rather than for the system designer! An example of this conflict of user versus designer convenience is the definition of a "generalized step" or "partitioned domain" function in the system described in Chapter 3. The following is the classical mathematical form

- of such a function (and the form chosen in Chapter 3):

$$f(x) = \begin{cases} \sqrt{-x} & \text{if } x < 0, \\ x & \text{if } 0 \leq x < 10, \\ 10 & \text{if } x \geq 10. \end{cases}$$

However the following form is considerably easier for a program to decode:

$$f(x) = \begin{cases} \text{if } x < 0, & \sqrt{-x} \\ \text{if } 0 \leq x < 10, & x \\ \text{if } x \geq 10, & 10 \end{cases}$$

since the "if" part of the specification must be examined prior to execution of the given expression. The designer must keep in mind that the man-computer communication gap is best narrowed by extending the computer's ability to communicate with man, not vice-versa.

General messages and Type 1 error messages (the type intercepted explicitly) must either be full, complete messages or be part of a multilevel message system. The latter type might be a rather terse, concise message issued first, and if the user did not understand this message or wanted more explanation, he could so indicate and receive a "second-level", more complete message.

A certain amount of advice might also be included in these messages. This would allow the system to service a wider range of user mathematical backgrounds. And in a multilevel message system one level (say level 3) could be reserved for "advice". Hence this information would not be displayed

unless asked for and so would not inconvenience a more experienced user.

The tutorial information should explain the entire system, both overall and each element in detail. Again a multilevel system would be advantageous, with more detailed descriptions and many examples appearing in higher levels. This information will probably be stored in auxiliary storage and retrieved sequentially. It would be convenient to allow the TUTOR element to have a character-type parameter which selects the topic, with perhaps the default being the beginning of the tutorial information. The user might input "TUTOR 'GAUSS'" to display information on GAUSS, an element used to perform Gauss elimination.

Implementation Considerations

In this section various aspects of the actual hardware and software used to implement an interactive system are discussed. It should be noted that this discussion is highly dependent on the state of the art. Although several aspects listed below as "desirable" either do not exist today or are rare, it is difficult to predict computer technology very far into the future. Often, too, a prospective system builder has little or no choice of facilities, and must use what he has. The following considerations are given for the sake of completeness of system design, as a general indication of the requirements of an interactive system.

Hardware

An interactive system is by implication an on-line system, and hence usually implies the necessity of time-sharing for economic reasons. Small computers are often much cheaper than large ones, making it possible to avoid time-sharing by allowing a single user to use the entire machine. However, small computers are often lacking in features desired for effective interactive systems, such as amount of storage and language support.

The speed at which a user is serviced is a more important consideration than one might think. Few users will tolerate a system in which they must wait more than a few seconds for anything less than a heavy computation request. Such wait times are a function of such items as computing speed (or machine "cycle time"), users concurrently using the system, concurrent background (batch) jobs, priorities of all active jobs, terminal-computer data transfer rates, and terminal display rates.

Computer stores are also an important consideration for interactive systems. Usually several levels of stores for data and/or programs exist in a computer system, such as high speed main storage, low speed main storage, drum, disk, and tape storage. Each level is typically slower, larger, and cheaper than the preceding level. An interactive system might, for example, reside in and be executed from low speed main storage, with tutorial information and certain error

messages in disk storage. The system might occasionally use some high speed main storage when heavy computational elements are referenced by the user.

Dynamic accuracy specification is an attractive feature for a problem-solving system. The user might decide he needs more accuracy in one or more phases of the solution process. Most computers today require accuracy specification (number of significant digits to be carried in the machine) to be done at programming time. And even then such specification is usually limited to two choices--"single" or "double" precision--rather than, say, "n decimal digits" of precision.

There are several types of terminals currently commercially available. It is not the purpose here to fully discuss the characteristics and advantages of various terminals since many such discussions exist in the literature (see Machover, (1967) for a discussion of graphic terminals, for example). Rather the purpose is to briefly indicate the nature of such equipment and some considerations for use in an interactive system.

Most existing terminals have a keyboard containing most of the characters found on a standard typewriter and usually a few special purpose keys. Alphameric display and graphic display terminals have a cathode ray tube (CRT) for their recording device, while terminals such as teletypes and other typewriter-type terminals use paper. Alphameric displays can display any of the characters appearing on the

keyboard, and the CRT is usually organized into 10 to 20 rows and 50 to 100 columns. Graphic displays have certain plotting, picture drawing, and/or vector drawing capabilities, with various additional types of input (e.g., light pen, "joystick", and even foot pedals). Most graphic displays are very expensive (often 25 to 50 times the price of a typewriter-type or alphanumeric display terminal), although some displays are now beginning to be produced with capabilities and prices somewhere between those of alphanumeric and graphic displays.

Some important considerations for terminals are cost, display rate, and the characteristics of the type of problem to be solved. For example, a rough point plot can be generated on an alphanumeric display in one or two seconds compared to minutes on a typewriter-type terminal. CRT displays are fast and quiet, but generally have no hard copy output facilities. Actually both a display tube and paper output make a very useful terminal combination.

Software

The programming language in which the interactive system is written must have several characteristics. These include easy character string manipulation, error recovery ability and adequate computational ability.

In languages that do not conveniently support character string manipulation, such as many versions of FORTRAN, manipulation facilities might be written by a programmer, but

then efficiency becomes an important question. Such simulation of character string facilities may be too expensive to use, and in any case requires effort in other than the main area of interest--the interactive system.

Error recovery is essential in an on-line environment--it is clearly unacceptable for the computer to kick the user off the machine every time he causes an error to occur. The interactive system must intercept errors, inform the user, and let him take appropriate action.

The following are several programming features considered desirable for certain applications:

- (1) Multiple entry points. These are useful for an element which may actually be a part of another element.
- (2) Versatile file manipulation. This can greatly ease messaging (e.g., with various types of direct access files).
- (3) Recursion. This can be invaluable for certain types of translators when the syntax productions are recursive. Chapter 3 has two examples of the use of recursion in the interactive root-finding system in parts of the function translator and the element extension translator. Recursion can be simulated, but this often requires a great deal more work for the programmer.

- (4) Compile-time or macro facilities. This again can ease programming of translators.
- (5) Arrays of operators. This feature could, for example, reduce one interpreter used in the root-finding system described in Chapter 3 from around 150 statements to five or ten statements.
- (6) Special features. Certain other features may also be desirable, depending on the particular application. A system for linear algebra, for example, would be more easily written in a programming language having convenient array manipulation facilities.

Several computer system characteristics may be desirable, including a terminal monitor and program overlay facilities. Overlay facilities allow a segmented program to take up a smaller amount of main storage by keeping in main storage only the segment currently being executed. This can greatly reduce the large storage costs incurred in an on-line system, with some increase in execution costs for the segment swapping required. It should be noted that the modular "element approach" of the system makes it particularly easy to use overlays.

Programming Considerations

In this section some programming techniques and characteristics of the interactive system program are discussed. These are considered essential for the creation of an effective, readable, and easily extendible system.

Execution efficiency is an important consideration. This does not mean that the program should be "streamlined" to the greatest degree. Quite the contrary--such streamlining often does more harm by destroying readability of a program than any gains received by increased execution times. Rather efficiency should be achieved by study of the most expensive parts of the system in execution time prior to programming. Efficient algorithms should be chosen, usually by considering known efficiencies and results from further experimenting. For example, in the problem of finding zeroes of a nonlinear function, the most expensive item except for extremely simple functions is evaluating the function. Hence a system used to find zeroes should incorporate efficient function evaluation.

User error recovery within an element should, for certain elements, be such that the error does not terminate execution of the element. For example, it is an aggravation for the user who has incorrectly defined an element to re-enter the entire element definition after receiving the error message.

The system must be easily extendible. It may be desired

to add elements to widen the range of applications of the system, and elements to supplement existing methods of solution. All implementation limits, such as number of elements allowed and length of element and variable names, should be specified as variables in the program rather than constants. This allows such limits to be easily changed. Checks on whether these limits are being exceeded, incidentally, must religiously be made in the program for self-protection of the system.

Documentation and readability are two important properties of a system. Gone are the days in computing when minimum execution time and minimum initial programming time were the sole concerns of programmers. The rapid changes in the computing industry make documentation and readability essential. Such things as variable, label, and procedure names should be meaningful. For example, "GO TO ERROR_RETURN" is much more informative and costs no more in execution time than the statement "GO TO L1".

Wherever possible, message texts should be coded in the program at the same place the message is assigned. This documents the action causing the message. And finally, explicit comments should supplement these implicit documentation methods wherever necessary.

And last, but far from least, is the requirement of extensive testing, or debugging, of an interactive system.

This must be done by both system builder and system users.

This process uncovers not only many bugs in the system, but also design flaws and certain inconveniences which can often be cured by small changes in the system.

CHAPTER 3

AN INTERACTIVE ROOT-FINDING SYSTEM

In this chapter a high-level graphic interactive root-finding system is discussed. This system was developed at Iowa State University as an experimental high-level interactive system. The system is an example of the type of interactive system described in general terms in Chapter 2.

Goals of the Root-Finding System

The problem considered here is the following: given a function f , find values x' such that $f(x') = 0$. Such x' are said to be zeroes of the function f , or, equivalently, roots of the equation $f(x) = 0$. Here f is a real-valued function of one real variable.

It was felt that the development of an interactive root-finding system with a graphic capability would be useful in several ways. The person with the problem to be solved (the user) could aid in several phases of the root-finding process which are difficult to handle automatically. It may also give the user a great deal more insight into, and understanding of, his problem than simply a few numbers (roots) printed by an automatic program or even by an automatic root-finding system. For example a user of an automatic system may be convinced that roots of his equation lie in a certain interval. By restricting the automatic system to this interval, which

in the NAPSS system allows a dramatic increase in efficiency (Rice, 1969a), the user may miss roots he could have found using an interactive system by noticing the drop in function values near an end of the interval. He may also be interested in knowing that the function is very erratic in the neighborhood of some zeroes, or even in some other neighborhood. The user may be interested in the behavior of his function under certain perturbations of a parameter or a coefficient. In other words, a graphic interactive system can be useful when an automatic routine or system is having difficulty or when there is more of an interest than a list of some roots.

Some specific phases and ways in which interaction can be useful are the following:

- (1) Supplying starting values for root-finding algorithms. A "sufficiently" good starting value is required for any algorithm, where "sufficiently" depends on the algorithm and the given function. The ability to graph the function quickly and wherever desired combined with the user yields a simple yet versatile search tool, able to supply starting values often as close to a zero as desired.
- (2) Examining and guiding the way through trouble areas. Included in possible trouble areas are the following:
 - (a) clustered roots

- (b) multiple roots
- (c) discontinuities
- (d) asymptotes to zero
- (e) roundoff-sensitive functions
- (f) badly scaled functions

(functions with these properties are included in an extensive set of test functions given by Rice (1969c)).

- (3) Graphically and algebraically verifying a root candidate (results from execution of an algorithm). The importance of this area can perhaps best be emphasized by quoting two authors of automatic root-finding systems:

Reliability is the most critical attribute of a polyalgorithm for the automatic solution of a mathematical problem (Rice 1969b, p. 6).

One of the major failures of many routines is in the choice of methods for finally accepting an approximation as a root (Champagne 1965, p. 57).

- (4) Graphic investigations of the neighborhood of a root. This is quite difficult to do automatically because of the many different situations that can arise.
- (5) Eliminate the computer time and error involved in deflating the function (each time it is evaluated!) by dividing by previously-found roots. This can be done by simply avoiding

previously searched intervals (possible in automatic systems, but with a significant increase in logic).

- (6) Elimination of a priori estimates. The fact that the user can dynamically evaluate results and decide when to continue is an important practical feature inherent in on-line and especially interactive systems. Such factors as time, number of roots, and convergence tolerances need not be specified beforehand. Human adeptness at pattern recognition can play a large role here, such as for functions with periodic roots or periodic trouble areas.
- (7) The unexpected. In an interactive system, this major source of trouble and worry for the builder of automatic systems is greatly reduced. Building into a system the ability to know when it cannot solve the problem can be a hard job.

The Implementation

The root-finding system was implemented on an IBM 360/65 computer. The computer system environment is that of time-sharing and multiprogramming with a variable number of tasks. A number of terminals are connected to this computer: teletypes, IBM Selectric typewriters, card-reader/line-printers,

and cathode ray tubes (CRTs). However the speed of a Model 65 and the memory configuration in use allow very fast servicing of users (while still doing most of its work via batch programs concurrently). Memories include 512 K bytes (1 K = 1024; a byte is eight binary digits and is the amount of storage required to store one character) of high speed main memory, 1024 K bytes of low speed main memory, and drums, disks, and tapes. The 360/65 computer has single (6 to 7 decimal digits) and double precision (16 decimal digits) for floating point numbers. The root-finding system currently uses double precision and is executed from low speed main storage, with a tutor file and error file on disk.

Several types of terminals were considered. The alphanumeric IBM 2260 CRT was chosen mainly for its high speed display ability. Although the 2260 allows only rough point plots, this was found to be quite satisfactory since the user has the ability to "zoom in" on any questionable portion of the plot. The teletype and typewriter are much too slow for generating such plots. The graphic IBM 2250-type terminals were ruled out because of cost--to us and to most other installations in the country, making sharing of the system almost impossible.

The language in which the system is written is PL/1. The possibilities were FORTRAN, ALGOL, and PL/1. The poor character manipulation and error recovery facilities in IBM 360 FORTRAN IV disqualified this widely used language,

although had nothing better been available, these deficiencies could have been somewhat overcome by the heavy use of simulated facilities written as subroutines, at the expense of increased execution times. The ALGOL language has many useful and powerful facilities, but the inferior IBM implementation of ALGOL makes this a very inconvenient language to use, even ignoring the weaknesses in input and output. PL/1, on the other hand, has all of the essential and desirable features mentioned in Chapter 2 except arrays of operators. It is true that a string manipulation language could have been used just to handle input, but the PL/1 string-handling facilities were judged adequate for this job.

The system includes two small assembly language routines--one as the display interface between the PL/1 program and the CRT, and a routine to set and check a cpu-time (central processing unit) timer for the user's convenience.

A monitor (see IBM (1969b) supplied by IBM is used for all the 2260's connected to the Model 65 computer.

The system is segmented into four segments: control, math and plot routines, function analyzer, and element analyzer. Each segment takes about 20,000 bytes of storage, and is in main storage when required (overlayed).

Elements

The problem of interactively finding a root of a non-linear equation can be segmented into three steps:

- (1) search for possible root,
- (2) application of mathematical algorithm(s),
- (3) verification of root candidate.

The basic elements chosen to carry out these steps are listed along with their characteristics (as outlined in Chapter 2) in Table 2.1. The information in this table completely defines each basic element to the rest of the system. And contained within the TUTOR element should be enough information to describe the entire system to a user. This tutorial information is displayed in Appendix A. The discussion below of each element pertains to matters other than those directly concerning a user, such as internal workings of the element and the philosophy of the element.

Mathematical elements

The mathematical algorithms currently in the system are half interval method (HI), the secant method (SEC) or method of false position), and a minimization method (MIN)¹. These

¹It is interesting to note that this set of routines is very nearly the same as those used in Rice's (1969a) automatic root-finding system. This set of routines was developed independently of Rice's results. The author was, however, guided by Rice's comment that "the simpler routines seemed to be the most reliable" (Rice, J. R., West Lafayette, Indiana. Root-finding routines. Private communication. 1969).

Table 3.1. Basic element properties

Element	Description	Inputs	Outputs	Internal interaction
HI	Half-interval method of finding a root in the interval [XMIN, XMAX]	Algebraic: XMIN, XMAX, EPS (convergence tolerance--set to $1/10^{**20}$ if 0), MAXITERS (maximum number of iterations allowed--set to 100 if 0)	Algebraic: X(root candidate), EVALS (total number of function evaluations to date), ITERS (number of iterations made), FLAG (error flag: = 0 if method converged, = 1 if got too small an interval, = 2 if starting points do not differ in sign, = 3 if no convergence after MAXITERS iterations)	None
SEC	Secant method of finding a root in the interval [XSTART, XSTOP]	Algebraic: XSTART, XSTOP, EPS (same as in HI), MAXITERS (same as HI)	Algebraic: X(same as HI), EVALS (same as HI), ITERS (same as HI), FLAG (error flag: = 0 if method converged, = 1 if function values became too close together, = 2 if X passes XSTART, = 3 if X passes XSTOP, = 4 if no convergence after MAXITERS iterations)	None

Table 3.1 (Continued)

Element	Description	Inputs	Outputs	Internal Interaction
MIN	Method to minimize $ f(x) $ in the interval $[XMIN, XMAX]$ (may converge to a relative minimum)	Algebraic: XMIN, XMAX, EPS (same as HI), MAXITERS (same as HI)	Algebraic: X (same as HI), EVALS (same as HI), ITERS (same as HI), FLAG (error flag: = 0 if method converged, = 1 if got to small interval, = 2 if no convergence after MAXITERS iterations)	None
Investigative: PLOT	Plots the function F in the interval $[XMIN, XMAX]$	Algebraic: XMIN, XMAX	Algebraic: XL (value of X where F(X) was smallest), XH (value of X where F(X) was largest), EVALS (same as HI), Displays: a point plot of F in the interval $[XMIN, XMAX]$	None
SHOW	Displays the current values of any algebraic element inputs, of F(X) for any constant X, or any element arithmetic expression	Character: list of items to be displayed	Display: the list of items to be displayed and their current values	Any syntax error in the list is noted along with a request to correct it

Table 3.1 (Continued)

Element	Description	Inputs	Outputs	Internal interaction
Special: F	Element used to define the function F whose zeroes are sought	Character: (optionally) The single character '?' (means display previous function definition)	Algebraic: FLAG (error flag: = 0 if function definition was accepted, = 1 if function definition was not accepted due to a syntax error) Other: function F is defined	The first display requests the function definition. Each syntax error is pointed out with a request to correct it. Definition of the independent variable and all parameters are requested.
Bookkeeping: TUTOR:	System tutorial information	None	Displays: system tutorial information	Allows input of element reference at any time - this terminates the TUTOR element
SETTIMER	Sets a cpu-time timer	Algebraic: TOTALTIME	Algebraic: EXPIRE (is set to zero when timer expires)	None
CHECK TIMER	Checks the cpu-time timer set with SETTIMER element	None	Algebraic: TOTALTIME (as set by SETTIMER), TIMEUSED (TOTALTIME - current value of timer)	None

Table 3.1 (Continued)

Element	Description	Inputs	Outputs	Internal interaction
SAVE	Saves the current display by storing it in an auxiliary storage file	None	Display is saved	None
CONT	Continues execution of the used-defined element that was invoked last	None	None	None
STOP	Shuts down the system	None	Control is returned to the terminal monitor	None
Element extensions: ELEMENT	Element used to define a user-defined element	None	Algebraic: FLAG (error flag: = 0 if user-defined element was accepted, = 1 if user-defined element was not accepted due to a syntax error)	The first display requests the element to be defined. Any syntax error is noted with a request to correct it

methods are quite simple and well-known and are described in the TUTOR (Appendix A), hence the discussions will be mainly on possible uses and trouble areas.

The half-interval method works well when it can be used, which is when the function differs in sign at the endpoints of the starting interval. The only problem that can occur is when the interval in which the root candidate lies becomes too small before a root is found (i.e., before $|f(x)| < \text{EPS}$). Here "too small" of an interval means the two endpoints differ by only one place in the last digit carried by the machine and so the "midpoint" of the interval would be identical to one of the endpoints. Hence the method can do no better. At this point the user should investigate the neighborhood of the resulting root candidate to check for a possible discontinuity.

The secant method can be useful when there is no sign change about a suspected root. There are two known trouble areas. One problem is when the function values at two successive iterates become nearly the same. The secant method involves a division by the difference of these two function values. This will cause a machine overflow when the difference is small enough (i.e., the result of the division is too large to be stored in the computer--about 10^{75} for the IBM 360). The second difficulty occurs when the results of an iteration fall outside the original specified interval. This may not always be a problem--the routine may eventually

converge to a value inside the original interval. However this particular routine was designed in this way to insure that only the interval given was examined. When this problem occurs, better starting values should be supplied.

The minimization method used here is a simple "stepping" scheme, and although slow can be useful when other methods fail. The method steps through the given interval in one direction, sampling $|f|$ at these "step points" until an increase in $|f|$ is found. At this point the direction of sampling is reversed and the procedure is repeated with a smaller step size. The user must keep in mind that the method may find only a relative minimum of $|f|$ rather than the true minimum in the given interval.

Investigative elements

The elements PLOT and SHOW are the investigative tools of the system. The ability to plot the function is invaluable in both the search and verification phases of root finding, as discussed earlier in this chapter (see the section on "Goals of the Root-Finding System"). The powerful SHOW element can display any arithmetic expression involving constants, algebraic inputs or outputs of any element, repetition variables, and any function values. The SHOW element is implemented as a special entry point in the element analyzer routine due to its character parameters.

Bookkeeping and special elements

The only special element in the system (other than the "element extension" element, which is discussed in another section) is the element used to define the function whose zeroes are sought.

The problem of handling the user's function involves deciding what functions will be allowed and how they should be specified (syntax), and how the functions will be handled internally. This is a non-trivial problem in a system that is interactive and accepts a natural, or user-oriented, notation. One way of handling natural notation is to translate it into a language for which a compiler or interpreter already exists, and then have the compiler generate machine code or the interpreter interpret the (translated) function. This method was not used for several reasons. Existing compilers or interpreters were either not dynamically accessible or created other problems such as large space requirements--a great deal of the translator was required, even though the function involved a very small part of the language. Linking of the compiled function to the rest of the system created problems, as did the matter of the compile-time and run-time error handling.

A special compiler could be written to translate the natural input directly into machine code. For even a small function syntax, this can be quite a project, and can be quite difficult to extend.

A very attractive solution to the problem of handling the user's function would be by the use of a compiler-compiler as discussed in the section on "Element Extensions", Chapter 2. Although not used in the root-finding system, the compiler-compiler promises to be a very useful tool here for the same reasons stated in Chapter 2.

The method chosen to handle the user's function in the root-finding system is a syntax-directed analyzer and interpreter. The analyzer is called "syntax-directed" because it is written directly from, and operates in the manner specified by, the function syntax written in Backus Normal Form (see Table 3.2 where "C" means an arithmetic constant, "I" means an identifier, "A" means an array, "F" means a mathematical function, "S" means "STEP", and "L" means "LOOP"). The PL/1 recursive feature was most useful in writing the analyzer, since some of the function syntax productions specify recursion. The function syntax was chosen to handle a wide range of functions, including all but three of Rice's (1969a) 74 test functions (two of the three were classified as "pathological") and some other types as well.

The analyzer checks the syntax of the function inputted by the user and generates a code vector consisting of operator codes, operand addresses, and destination addresses. This vector is then used by the function interpreter each time the function is evaluated.

Table 3.2 Function syntax

```

<FUNCTION>::=<STATEMENT>
<FUNCTION>::=<FUNCTION>; <STATEMENT>
<STATEMENT>::=I=<ARITH EXPR>

<ARITH EXPR>::=<TERM>
<ARITH EXPR>::=<ADDSUB OP><TERM>
<ARITH EXPR>::=<ARITH EXPR><ADDSUB OP><TERM>

<TERM>::=<FACTOR>
<TERM>::=<TERM><MULDIV OP><FACTOR>

<FACTOR>::=<PRIMARY>
<FACTOR>::=<FACTOR>**<PRIMARY>

<PRIMARY>::=C
<PRIMARY>::=I
<PRIMARY>::=A(<ARITH EXPR>)
<PRIMARY>::=F(<ARG LIST>)
<PRIMARY>::=S(<STEP LIST>)
<PRIMARY>::=L(<LOOP INDEX>,<ARITH EXPR>)
<PRIMARY>::=( <ARITH EXPR>)

<ADDSUB OP>::=+
<ADDSUB OP>::=-

<MULDIV OP>::=*
<MULDIV OP>::=/

<ARG LIST>::=<ARITH EXPR>
<ARG LIST>::=<ARG LIST>,<ARITH EXPR>

<STEP LIST>::=<STEP>
<STEP LIST>::=<STEP LIST>,<STEP>

<STEP>::=<ARITH EXPR><DOMAIN>

<DOMAIN>::= IF <BOOL EXPR>
<DOMAIN>::= OTHERWISE

<BOOL EXPR>::=<ARITH EXPR><BOOL OP><ARITH EXPR><OPT COMPARE>

<OPT COMPARE>::=<BOOL OP><ARITH EXPR>
<OPT COMPARE>::=<EMPTY>

<LOOP INDEX>::=I=<ARITH EXPR> TO <ARITH EXPR><BY CLAUSE>

<BY CLAUSE>::= BY <ARITH EXPR>
<BY CLAUSE>::=<EMPTY>

```

For example, the function

$$f(x) = 2x + \cos(x + 3)$$

would be inputted as

$$f = 2 * X + \text{COS}(X + 3)$$

and the analyzer would generate the following code vector:

$$\underbrace{* \quad 2 \quad X \quad T_1}_{\text{group 1}} \quad + \quad \underbrace{X \quad 3 \quad T_2}_{\text{group 2}} \quad \underbrace{\text{COS} \quad T_2 \quad T_2}_{\text{group 3}} \quad + \quad \underbrace{T_1 \quad T_2 \quad T_1}_{\text{group 4}}$$

Each group of code is an operator followed by the operands and a destination (actually what is generated is an operator code number followed by operand addresses and a destination address). The way this is executed by the interpreter is:

- (1) 2 is multiplied by X and the product is stored in temporary location T_1 ;
- (2) X is added to 3 and the sum is stored in temporary T_2 ;
- (3) the cosine of the contents of T_2 is computed and stored back in T_2 ;
- (4) finally the sum of the contents of T_1 and T_2 is stored in T_1 , which now holds the function value.

This type of code generated by the analyzer is very efficiently executed. Although not as compact as other types, such as postfix or prefix notation (see, for example, Wegner 1968, pp. 241-242), it is executed quickly because the

interpreter need not distinguish between operators and operands, and temporary storage allocation and deallocation is done at analysis or translation time rather than at execution time. The interpreter is included along with the math routines and is not written as a separate PL/1 procedure for efficiency reasons. A tremendous amount of overhead is required by PL/1 to initialize the labels used in the interpreter. This would need to be done for every function evaluation had the interpreter been written as a procedure. A small amount of inconvenience is caused for the system builder by having to branch to the interpreter rather than calling it as a procedure, but the increase in efficiency of this method made the inconvenience well worth while. In Table 3.3 some execution time comparisons are made between the implemented analyzer and interpreter and the PL/1 (Version F) compiler.

The analyzer informs the user of the description and location of any syntax errors, allows him to correct them, and then retries the analysis and code generation from the beginning. The analyzer then requests identification of the independent variable and values of any parameters.

Bookkeeping elements are TUTOR, SETTIMER, CHECKTIMER, SAVE, CONT, and STOP. These elements are explained in the TUTOR (Appendix A). The tutorial information in TUTOR is kept in auxiliary storage (disk) and displayed sequentially in one-screen segments (960 characters). The SETTIMER and

CHECKTIMER elements use a small assembly language routine which uses a system macro to set and check a cpu-time (central processing unit time) timer. These two elements allow the user to get a feel for how much time he is using and to monitor various phases of the solution process.

Table 3.3. Comparisons of several sample functions showing execution time ratio of interpreted functions to compiled functions

Function	Ratio
$X + 3.5 * X$	3:1
$((((2 * X + 3.5) * X - 2) * X + 1) * X - 7) * X + 1$	7:1
$X + \text{MIN}(3.5, X, 2 * X) + 1$	5:1
Functions containing one or more math routines	1.5:1

Element extensions

The user may define his own elements in terms of basic elements or other user-defined elements by using the basic element named ELEMENT. This allows the user to dynamically increase the degree of automation of the system. The concept of element extensions is discussed in Chapter 2 ("Element Extensions" section), and the details of how a user-defined element is formed are given in the TUTOR (Appendix A). The formal syntax is given in Table 3.4, where "C" means an arithmetic constant, "I" means an identifier, and "F" refers

Table 3.4. Element extension syntax

```

<ELEMENT>::=<ELEMENT NAME><PARAMS>;<STATEMENTS>

<ELEMENT NAME>::=I

<PARAMS>::=<EMPTY>
<PARAMS>::= I
<PARAMS>::=<PARAMS> I

<STATEMENTS>::=<STATEMENT>
<STATEMENTS>::=<STATEMENTS>;<STATEMENT>

<STATEMENT>::=<IF STATEMENT>
<STATEMENT>::=<ELEMENT REFERENCE>

<IF STATEMENT>::= IF <BOOL EXPR> THEN <ELEMENT REFERENCE>;
               <IF TAIL>

<IF TAIL>::= ELSE <ELEMENT REFERENCE>
<IF TAIL>::=<EMPTY>

<ELEMENT REFERENCE>::=<ELEMENT NAME><ACTUAL PARAMS><REPETITION>

<ACTUAL PARAMS>::=<EMPTY>
<ACTUAL PARAMS>::=<ARITH EXPR>
<ACTUAL PARAMS>::=<ACTUAL PARAMS> <ARITH EXPR>

<REPETITION>::=,I=C,C,...<REPETITION STOP>
<REPETITION>::=<EMPTY>

<REPETITION STOP>::=,C
<REPETITION STOP>::=<EMPTY>

<ARITH EXPR>::=<TERM>
<ARITH EXPR>::=<ADDSUB OP><TERM>
<ARITH EXPR>::=<ARITH EXPR><ADDSUB OP><TERM>

<TERM>::=<FACTOR>
<TERM>::=<TERM><MULDIV OP><FACTOR>

<FACTOR>::=<PRIMARY>
<FACTOR>::=<FACTOR>**<PRIMARY>

<PRIMARY>::=C
<PRIMARY>::=I
<PRIMARY>::=F(<ARG LIST>)

```

Table 3.4 (Continued)

```

<ADDSUB OP>::=+
<ADDSUB OP>::=-

<MULDIV OP>::=*
<MULDIV OP>::=/

<BOOL_EXPR>::=<ARITH_EXPR><BOOL_OP><ARITH_EXPR>

```

to the function F .

Essentially an element extension is of the form

```
<element name>      <list of parameters>;  
  
<statement>;  
  
.  
.  
.  
<statement>;
```

where <element name> is the name of the element being defined and <list of parameters> is a list of zero or more parameters of the element. A <statement> is either an <element reference> or an <if statement>. An <element reference> is simply a reference (or "call") to an element followed by actual parameters of the element, optionally followed by a repetition specification. An actual parameter is an arithmetic expression, which may involve function values, algebraic element inputs and outputs, and repetition variables.

An `<if statement>` is of the form

[illegible]

where the bracketed part is optional.

There is no rule prohibiting an element from referring to itself, or "recursing". Indeed this technique can be very useful and concise, although the same result can usually be attained through repetition specification. For example, the element defined as

```
SHIFT X Y;
PLOT X Y;
SHIFT Y Y+(Y-X);
```

will, when referenced by SHIFT 1 3, generate the following plot commands:

```
PLOT 1 3
PLOT 3 5
PLOT 5 7
.
.
.
```

and stops only when the user inputs another command.

The above example could have been written using repetition as

```
SHIFT X Y;
PLOT Y+(Y-X)*(J-1) Y+(Y-X)*J,J=0,1,...
```

Although an element may refer to itself, recursion in the strict sense of the word is not supported. There is no "return" from an element that has been called recursively and hence no saving of the element environment at each recursion for use upon return. The idea behind an element

"referring to itself" is to "repeat the same process" (usually with different parameters) rather than to execute true recursion with all its implications.

These element extensions are handled by the system in exactly the same manner in which the user's function is handled. A syntax-directed analyzer analyzes the element being defined and generates a code vector. This code vector is executed by an interpreter, which resides in the control routine. See the description of the F element in the subsection on "Bookkeeping and special elements" for a discussion of these techniques.

Communication

The form of user input was designed to be as natural as possible. Such function types as SUM (Σ), PROD (Π), and STEP ("generalized step" or "partitioned domain") are specified as near to classical mathematical notation as is possible with a standard keyboard and one-dimensional input. The language in which a user defines an element is simple and natural. Only two types of statements are possible, both of which are certainly natural to the scientist or engineer: reference to a procedure (element) and the conditional statement "if... then... else...".

The message system is a one-level system. Type 2 error messages are stored in an auxiliary file (disk) and the proper message is retrieved (using PL/1 direct access file

facilities) when an error occurs.

The system tutorial information (in the TUTOR element) is also on one level and is always accessed sequentially from the beginning. The ability of the user to specify the subject he wishes to refer to would be useful, but would require a character parameter (the tutorial subject) for the TUTOR element. This would currently have to be handled specially by the system since all elements have arithmetic parameters (except SHOW, which currently must be handled specially). The tutorial information contained in the TUTOR element is given in Appendix A.

CHAPTER 4

EXAMPLES USING THE ROOT-FINDING SYSTEM

In this chapter some examples are given which use various features of the root-finding system. The figures are exact copies of the cathode ray tube displays and were obtained by use of the SAVE element, which saves any desired displays in auxiliary storage for future use.

The examples are mainly functions whose zeroes are sought. The functions contain many of the trouble areas for root-finding routines mentioned in Chapter 3. The power of element extensions in interactive root-finding will be demonstrated throughout these examples. Various other features of the system will also be displayed, such as error detection.

Results of solving these functions with the interactive system are compared with the results of solution with the NAPSS polyalgorithm for finding zeroes (Rice, 1969a). Time, accuracy, and number of roots found are compared. It was felt that comparison should be with an automatic system of some kind rather than with single root-finding algorithms since root-finding involves several processes--search for starting values, use of mathematical algorithms, and verification of root candidates. The NAPSS routine is a 1969 IBM 360 version. Because of the difference in the CDC 6500 and IBM 360 computers and FORTRAN languages, the results given here may not always be identical with those obtained by Rice.

Comparisons with the NAPSS system should be considered as only an indication of the type of results possible with interactive and automatic systems.

The figures often do not show all the displays used in solving a given equation, rather just those required to demonstrate various features. All results and comparisons with the NAPSS system are summarized in a table at the end of the chapter.

Input from the user is always begun in the lower left corner of the display. The only exceptions to this are during function and user-defined element definitions, where only the top line or so is written by the system. An "L" marks the lowest point found in a given plot. An "H" marks the highest point. These values are displayed with each plot, labeled as XL and XH, respectively. In these examples the default value of MAXITERS for the HI element was 25.

The first example (Figure 4.1) is a rather simple one¹ but is useful in demonstrating the entire root-finding process. In the first two displays the function F whose zeroes are desired is defined. Then the cpu-time timer is set for timing purposes. The function is then plotted in the search for starting values. The half-interval method is invoked using $X = 1000$ and 10000 ($1E5$) as starting values and the default values (signaled by inputting two zeroes) for EPS, the convergence tolerance, and MAXITERS, the maximum number of

¹It is assumed that the user has no knowledge of the roots, hence inspection of the function is not allowed.

T#01 INPUT FUNCTION:

$$F=(X-12000)*(X+21E6)$$

T#01 PLEASE IDENTIFY THE INDEPENDENT VARIABLE:

X

Figure 4.1. Simple function to show entire root-finding process

T#01 TIMER HAS NOW BEEN SET TO 30.00 SECONDS

SETTIMER 30

T#01 MAXIMUM F:
-2.519789732E+11

EVALS: 33
XH:
1.000000000E+00
XL:
0.000000000E+00

MINIMUM F:
-2.520000000E+11 L
XMIN: 0.000000000E+00

XMAX: 1.000000000E+00

Figure 4.1 (Continued)

T#01 MAXIMUM F:
-2.517900984E+11

EVALS: 66

XH:
1.000000000E+01

XL:
1.000000000E+00

MINIMUM F:
-2.51790120E+11 L

XMIN: 1.000000000E+00

XMAX: 1.000000000E+01

T#01 MAXIMUM F:
-2.310109921E+11

EVALS: 99

XH:
1.000000000E+03

XL:
1.000000000E+01

MINIMUM F:
-2.517901199E+11 L

XMIN: 1.000000000E+01

XMAX: 1.000000000E+03

Figure 4.1 (Continued)

T#01 MAXIMUM F:
1.856799506E+12

EVALS: 132
XH:
1.000000000E+05
XL:
1.000000000E+03

MINIMUM F:
-2.310110000E+11 L
XMIN: 1.000000000E+03

XMAX: 1.000000000E+05

T#01 IN HI METHOD, NO CONVERGENCE AFTER MAXIMUM NUMBER OF ITERATIONS SPECIFIED:

X	F(X)	ITERATIONS
1.200000229E+04	4.821789265E+04	25

EVALS: 159
XH:
1.000000000E+05
XL:
1.000000000E+03

MINIMUM F:
-2.310110000E+11 L
HI 1000 1E5 0 0 1.000000000E+03

XMAX: 1.000000000E+05

Figure 4.1 (Continued)

T#01 HALF INTERVAL METHOD CONVERGED:

X	F(X)	ITERATIONS
1.200000000E+04	0.000000000E+00	57

EVALS: 218

XH:

1.000000000E+05

XL:

1.000000000E+03

MINIMUM F:

-2.310110000E+11 L

HI 1000 1E5 0 100 1.000000000E+03

XMAX: 1.000000000E+05

T#01 MAXIMUM F:

2.101200000E+07

EVALS: 251

XH:

1.200100000E+04

XL:

1.199900000E+04

MINIMUM F:

-2.101199900E+07 L

XMIN: 1.199900000E+04

XMAX: 1.200100000E+04

Figure 4.1 (Continued)

T#01 MAXIMUM F:
2.101199609E+04

EVALS: 284

XH:

1.200000100E+04

XL:

1.199999900E+04

MINIMUM F:

-2.101200000E+04 L

XMIN: 1.199999900E+04

XMAX: 1.200000100E+04

T#01 YOU HAVE USED 2.12 SECONDS OF YOUR TOTAL 30.00 SECONDS
2.101199609E+04

EVALS: 284

XH:

1.200000100E+04

XL:

1.199999900E+04

MINIMUM F:

-2.101200000E+04 L

CHECKTIMER XMIN: 1.199999900E+04

XMAX: 1.200000100E+04

Figure 4.1 (Continued)

T#01 MAXIMUM F:
1.050602496E+09

EVALS: 317
XH:
1.205000000E+04
XL:
1.200100000E+04

MINIMUM F:
2.101200000E+07 L
XMIN: 1.200100000E+04

XMAX: 1.205000000E+04

T#01 MAXIMUM F:
1.681599365E+11

EVALS: 383
XH:
2.000000000E+04
XL:
1.300000000E+04

MINIMUM F:
2.101299610E+10 L
XMIN: 1.300000000E+04

XMAX: 2.000000000E+04

Figure 4.1 (Continued)

RIGHTSHIFT X Y;
PLOT X Y;
RIGHTSHIFT Y Y*10;

```

EVALS:      482
XH:
  1.000000000E+07
XL:
  1.000000000E+06

```

```

MINIMUM F:      . . .
2.173599062E+13 L . .
XMIN: 1.000000000E+06

```

XMAX: 1.0000000000E+07

Figure 4.1 (Continued)

T#01 MAXIMUM F:
1.209854492E+16

EVALS: 515

XH:

1.000000000E+08

XL:

1.000000000E+07

MINIMUM F:

3.096279505E+14 L

XMIN: 1.000000000E+07

XMAX: 1.000000000E+08

T#01 MAXIMUM F:
1.020987737E+18

EVALS: 548

XH:

1.000000000E+09

XL:

1.000000000E+08

MINIMUM F:

1.209854492E+16 L

XMIN: 1.000000000E+08

XMAX: 1.000000000E+09

Figure 4.1 (Continued)

T#01 YOU HAVE USED 4.47 SECONDS OF YOUR TOTAL 30.00 SECONDS
1.020987737E+18

EVALS: 548

XH:

1.000000000E+09

XL:

1.000000000E+08

MINIMUM F:

1.209854492E+16 L

CHECKTIMER XMIN: 1.000000000E+08

XMAX: 1.000000000E+09

T#01 MAXIMUM F:

-2.519999447E+11

EVALS: 581

XH:

-4.440892099E-16

XL:

-1.000000000E+01

MINIMUM F:

-2.522098799E+11 L

XMIN: -1.000000000E+01

XMAX: 0.000000000E+00

Figure 4.1 (Continued)

T#01 MAXIMUM F:
-2.522098565E+11

EVALS: 614

XH:
-1.000000000E+01
XL:
-1.000000000E+04

MINIMUM F:

-4.617800000E+11 L

XMIN:-1.000000000E+04

XMAX:-1.000000000E+01

T#01 MAXIMUM F:
-4.617799598E+11

EVALS: 647

XH:
-1.000000000E+04
XL:
-1.000000000E+07

MINIMUM F:

-1.101320000E+14 L

XMIN:-1.000000000E+07

XMAX:-1.000000000E+04

Figure 4.1 (Continued)

T#01 MAXIMUM F: H
9.790117480E+17

EVALS: 680

XH:
-1.000000000E+09

XL:
-1.000000000E+07

MINIMUM F:

-1.101319876E+14 -----L
XMIN:-1.000000000E+09 XMAX:-1.000000000E+07

T#01 SECANT METHOD CONVERGED:

X	F(X)	ITERATIONS
-2.100000000E+07	0.000000000E+00	16

EVALS: 698

XH:
-1.000000000E+09

XL:
-1.000000000E+07

MINIMUM F:

-1.101319876E+14 -----L
SEC -1E9 -1E7 0 0 1.000000000E+09 XMAX:-1.000000000E+07

Figure 4.1 (Continued)


```

T#01 YOU HAVE USED 5.30 SECONDS OF YOUR TOTAL 30.00 SECONDS
      X          F(X)          ITERATIONS
      -2.100000000E+07      0.000000000E+00      16
EVALS: 698
XH:
-1.000000000E+09
XL:
-1.000000000E+07

MINIMUM F:
-1.101319876E+14 -----L
CHECKTIMER E7 0 0 1.000000000E+09      XMAX:-1.000000000E+07

```

Figure 4.1 (Continued)

iterations allowed. These default values are 10^{-20} (1E-20) and 25, respectively. Convergence occurred when the half-interval method was retried for 100 maximum iterations. In the next few displays the neighborhood of the root is briefly examined and then the search moves to the right of the root. A user-defined element called RIGHTSHIFT is used to search farther and farther out to the right of the origin. This element makes it easy to check extremes of probable root areas, i.e., when the function values are getting larger and larger. RIGHTSHIFT is defined as:

```
RIGHTSHIFT X Y;
PLOT X Y;
RIGHTSHIFT Y Y*10;
```

The user then inputs the command

```
RIGHTSHIFT 1E6 1E7.
```

This automatically generates the following commands:

```
PLOT 1E6 1E7
PLOT 1E7 1E8
PLOT 1E8 1E9
```

with no further input from the user. He need only push the ENTER button to cause the next plot to appear. Stepping through the execution of this element in detail, when RIGHTSHIFT is referenced by the user as RIGHTSHIFT 1E6 1E7, the following commands are generated, according to the definition of RIGHTSHIFT:

```
PLOT 1E6 1E7;
```

```
RIGHTSHIFT 1E7 1E7*10;
```

hence the plot from $1E6$ to $1E7$ is generated, then RIGHTSHIFT is referenced with parameters of $1E7$ and $1E8$, and the process is repeated. The search is then carried to the left of the origin, where another root is found. Displays omitted here and in other examples include close examination of certain roots and their neighborhoods and searching out far enough until the user is convinced he has found all the roots he requires.

The second example (Figure 4.2) contains asymptotes to zero and a singularity: $F = 1/X$. Division by zero is attempted, and further investigation shows a likely singularity at zero. After 199 function evaluations an asymptote seems apparent. Further examination around the origin and to the left of zero could be carried out without taking much time. The NAPSS routine gave an erroneous root of $X = -.147E-7$ for this function although it labeled it as a "possible discontinuity". This may be a case where the program was not working as intended for the CDC computer, because the function is quite large at $X = -.147E-7$ and the NAPSS routine usually is more reliable than indicated here. The function $F = 1/\text{MAX}(\text{ABS}(X), 1E-6)$ was run on both systems. The solution phase went about the same as with $1/X$ on the interactive system. The NAPSS routine reported no roots after 1310 function evaluations, and terminated "because of an excessive number of asymptotic conditions".

T#01 **** FUNCTION ACCEPTED ****

$F=1/X$

PLOT 0 1

T#01 # 320 ZERO DIVIDE:
AN ATTEMPT HAS BEEN MADE TO DIVIDE BY ZERO

Figure 4.2. Asymptotic function

T#01 MAXIMUM F: H
1.000000000E+00

EVALS: 34

XH:
1.000000000E+00

XL:
2.000000000E+00

MINIMUM F:
5.000000000E-01

XMIN: 1.000000000E+00

XMAX: 2.000000000E+00

T#01 MAXIMUM F:
3.100000000E+01

EVALS: 100

XH:
3.225806452E-02

XL:
-3.225806452E-02

MINIMUM F:
-3.099998474E+01

XMIN: -1.000000000E+00

XMAX: 1.000000000E+00

Figure 4.2 (Continued)

T#01 MAXIMUM F: H
1.000000000E+00

EVALS: 166

XH:
1.000000000E+00

XL:
1.000000000E+01

MINIMUM F:
9.999996424E-02

XMIN: 1.000000000E+00

XMAX: 1.000000000E+01

T#01 MAXIMUM F: H
1.000000000E+00

EVALS: 166

XH:
1.000000000E+00

XL:
1.000000000E+01

MINIMUM F:
9.999996424E-02

RIGHTSHIFT 10 100 1.000000000E+00

XMAX: 1.000000000E+01

Figure 4.2 (Continued)

T#01 MAXIMUM F: H
1.000000000E-01

EVALS: 133
XH:
1.000000000E+01
XL:
1.000000000E+02

MINIMUM F:
9.99997914E-03

XMIN: 1.000000000E+01

XMAX: 1.000000000E+02

T#01 MAXIMUM F: H
1.000000000E-02

EVALS: 166
XH:
1.000000000E+02
XL:
1.000000000E+03

MINIMUM F:
9.99999311E-04

XMIN: 1.000000000E+02

XMAX: 1.000000000E+03

Figure 4.2 (Continued)

T#01 YOU HAVE USED 1.47 SECONDS OF YOUR TOTAL 30.00 SECONDS
1.000000000E-03

EVALS: 199

XH:

1.000000000E+03

XL:

1.000000000E+04

MINIMUM F:

9.999999020E-05

CHECKTIMER XMIN: 1.000000000E+03

XMAX: 1.000000000E+04

Figure 4.2 (Continued)

The next example (Figure 4.3) contains clustered roots. The following two user-defined elements were used to examine the neighborhood of the root found by the secant method:

```
CENTER X DELTA;
PLOT X-DELTA X+DELTA;
```

and

```
NHOOD X DEL;
CENTER X DEL;
NHOOD X DEL*2;.
```

The NHOOD element allows the user to inspect a very small neighborhood of a root, then allows him to "back away" from the root by generating plots centered about the root with an increasing radius. The NHOOD element is defined in this example in terms of CENTER, an element which generates a plot about the point X with radius DELTA. NHOOD is written recursively, although it could have been written iteratively as:

```
NHOOD X DEL;
CENTER X DEL*2**I, I=0,1,...;
```

or simply as:

```
NHOOD X DEL;
PLOT X-DEL*2**I X+DEL*2**I, I=0,L,...;
```

Use of these and other elements led to the discovery of two other roots close to the first one.

Figure 4.4 shows a function with a root located near a discontinuity. After a strange area is discovered near the

T#01 INPUT FUNCTION:

$$F = \text{ABS}(X-17)**2 * \text{ABS}(X-17.1)**1.8 * (X-20)$$

T#01 MAXIMUM F:
-7.232361250E+05

EVALS: 33
XH:
1.000000000E+00
XL:
0.000000000E+00

MINIMUM F:
-9.579014643E+05 L
XMIN: 0.000000000E+00

XMAX: 1.000000000E+00

Figure 4.3. Function with clustered zeroes

T#01 MAXIMUM F:
-1.669017969E+04

EVALS: 66
XH:
1.000000000E+01
XL:
1.000000000E+00

MINIMUM F:
-7.232361791E+05 L

XMIN: 1.000000000E+00

XMAX: 1.000000000E+01

T#01 MAXIMUM F:
-2.173296807E-13

EVALS: 99
XH:
2.000000000E+01
XL:
1.000000000E+01

MINIMUM F:
-1.669018335E+04 L

XMIN: 1.000000000E+01

XMAX: 2.000000000E+01

Figure 4.3 (Continued)

T#01 SECANT METHOD CONVERGED:

	X	F(X)	ITERATIONS
	1.700000000E+01	-9.693443934E-21	63
EVALS:	164		
XH:			
	2.000000000E+01		
XL:			
	1.000000000E+01		

MINIMUM F:

-1.669018335E+04 L

SEC 10 20 0 0 IN: 1.000000000E+01

XMAX: 2.000000000E+01

T#01 **** ELEMENT ACCEPTED ****

NHOOD X DEL;
CENTER X DEL;
NHOOD X DEL*2;

NHOOD 17 .01

Figure 4.3 (Continued)

T#01 MAXIMUM F:
-4.918415897E-09

EVALS: 197

XH:
1.700032258E+01

XL:
1.699000000E+01

MINIMUM F:
-5.663349043E-06 L

XMIN: 1.699000000E+01

XMAX: 1.701000000E+01

T#01 MAXIMUM F:
-1.955710971E-08

EVALS: 230

XH:
1.700064516E+01

XL:
1.698000000E+01

MINIMUM F:
-2.658233908E-05 L

XMIN: 1.698000000E+01

XMAX: 1.702000000E+01

Figure 4.3 (Continued)

T#01 MAXIMUM F:
-7.729977369E-08

EVALS: 263

XH:
1.700129032E+01

XL:
1.696000000E+01

MINIMUM F:
-1.412615706E-04 L

XMIN: 1.696000000E+01

XMAX: 1.704000000E+01

T#01 MAXIMUM F:
-3.018321308E-07

EVALS: 296

XH:
1.700258065E+01

XL:
1.692000000E+01

MINIMUM F:
-8.999558403E-04 L

XMIN: 1.692000000E+01

XMAX: 1.708000000E+01

Figure 4.3 (Continued)

T#01 MAXIMUM F:
-3.646974278E-07

EVALS: 329

XH:

1.709806452E+01

XL:

1.684000000E+01

MINIMUM F:

-7.159440337E-03 L

XMIN: 1.684000000E+01

XMAX: 1.716000000E+01

T#01 MAXIMUM F:
-3.399622983E-06

EVALS: 362

XH:

1.709290323E+01

XL:

1.668000000E+01

MINIMUM F:

-7.133238798E-02 L

XMIN: 1.668000000E+01

XMAX: 1.732000000E+01

Figure 4.3 (Continued)

T#01 MAXIMUM F:
-2.173296807E-13

EVALS: 428
XH:
2.000000000E+01
XL:
1.500000000E+01

MINIMUM F:
-7.603695667E+01 L

XMIN: 1.500000000E+01

XMAX: 2.000000000E+01

T#01 SECANT METHOD CONVERGED:

X	F(X)	ITERATIONS
2.000000000E+01	0.000000000E+00	15

EVALS: 544
XH:
2.500000000E+01
XL:
1.500000000E+01

MINIMUM F:

-7.603695667E+01 L

SEC 25 15 0 0 1.500000000E+01

XMAX: 2.500000000E+01

Figure 4.3 (Continued)

T#01 MAXIMUM F: H-.-.-.-.-.-.-.-.-.-.
0.000000000E+00

MINIMUM F:
-1.545150852E+01

XMIN: 1.7000000000E+01

XMAX: 2.00000000E+01

T#01 SECANT METHOD CONVERGED:

X	F(X)
1.710000000E+01	-7.003001779E-21

ITERATIONS
49

EVALS: 1048

XH:
1.709290323E+01
XL:
1.668000000E+01

MINIMUM F:

-7.133238798E-02 L

SEC 18 17 0 0 1.668000000E+01

```
XMAX: 1.732000000E+01
```

Figure 4.3 (Continued)

Figure 4.3 (Continued)

T#01 INPUT FUNCTION:

$F = \text{ABS}(X - 361) ** .7 * \text{ABS}(X + 157.2) ** 1.5 * (X - .1E-5) / \text{ABS}(X - .1E-10)$

T#01 MAXIMUM F: -----H
1.225239375E+05

EVALS: 33

XH:

1.000000000E+00

XL:

0.000000000E+00

MINIMUM F:

-1.216000781E+10 L

XMIN: 0.000000000E+00

XMAX: 1.000000000E+00

Figure 4.4. Function with a root near a discontinuity

T#01 **** ELEMENT ACCEPTED ****

ZOOM X RADIUS DECREMENT;
PLOT X-RADIUS X+RADIUS;
ZOOM X RADIUS/DECREMENT DECREMENT;

ZOOM 0 1 2

T#01 MAXIMUM F:
1.225239375E+05

. H

EVALS: 99

XH:

1.000000000E+00 -----

XL:

-3.225806452E-02

MINIMUM F:

-1.215740000E+05 L

XMIN: -1.000000000E+00

XMAX: 1.000000000E+00

Figure 4.4 (Continued)

T#01 MAXIMUM F:
1.225239375E+05

. H

EVALS: 165

XH:

1.000000000E+00 -----

XL:

-3.225806452E-02

MINIMUM F:

-1.215740000E+05 L

XMIN: -1.000000000E+00

XMAX: 1.000000000E+00

T#01 MAXIMUM F:
1.216912500E+05

. H

EVALS: 198

XH:

1.000000000E-01 -----

XL:

-3.225806452E-03

MINIMUM F:

-1.216347500E+05 L

XMIN: -1.000000000E-01

XMAX: 1.000000000E-01

Figure 4.4 (Continued)

T#01 MAXIMUM F: H
1.215971250E+05

EVALS: 231

XH:

1.000000000E-02 -----

XL:

-3.225806452E-04

MINIMUM F:

-1.219766875E+05 L

XMIN:-1.000000000E-02

XMAX: 1.000000000E-02

T#01 MAXIMUM F: H
1.214793750E+05

EVALS: 264

XH:

1.000000000E-03 -----

XL:

-3.225806452E-05

MINIMUM F:

-1.253695625E+05 L

XMIN:-1.000000000E-03

XMAX: 1.000000000E-03

Figure 4.4 (Continued)

T#01 MAXIMUM F: H
1.203841250E+05

EVALS: 297

XH: -----

1.000000000E-04

XL:

-3.225806452E-06

MINIMUM F:

-1.592955625E+05

L

XMIN: -1.000000000E-04

XMAX: 1.000000000E-04

T#01 MAXIMUM F: H
1.094401875E+05

EVALS: 330

XH: -----

1.000000000E-05

XL:

-3.225806452E-07

MINIMUM F:

-4.985448125E+05

L

XMIN: -1.000000000E-05

XMAX: 1.000000000E-05

Figure 4.4 (Continued)

T#01 IN HI METHOD, NO CONVERGENCE AFTER MAXIMUM NUMBER OF ITERATIONS SPECIFIED:

	X	F(X)	ITERATIONS
	5.960464478E-03	1.215851868E+05	25
EVALS:	476		
XH:	1.000000000E-05		
XL:	-3.225806452E-07		
MINIMUM F:			
	-4.985448125E+05		
HI	-1E5 1E5 0 0	1.000000000E-05	
			XMAX: 1.000000000E-05

T#01 HALF INTERVAL METHOD CONVERGED:

	X	F(X)	ITERATIONS
	1.000000000E-06	0.000000000E+00	90
EVALS:	568		
XH:	1.000000000E-05		
XL:	-3.225806452E-07		
MINIMUM F:			
	-4.985448125E+05		
HI	-1E5 1E5 0 100	1.000000000E-05	
			XMAX: 1.000000000E-05

Figure 4.4 (Continued)

origin, an element called ZOOM is used to "zoom in" on the suspected trouble area. This element allows inspection of a neighborhood of a point by generating plots centered about the point with decreasing radii. This example was given mainly to show the use of the ZOOM element; the half-interval method could have been invoked at any time. Even if there had been a discontinuity, the half-interval method would have helped isolate it.

The function $F=\text{SIN}(1/(X-5))$ is examined in Figure 4.5. A right parenthesis was omitted in the definition of the function. This was reported by the system and corrected by the user. The asymptote to the left of the origin and the trouble area around $X=4$ or 5 were located. The APPROACH element was designed to approach this area from the left cautiously. This idea was then incorporated into a "PLOTNSOLVE" element after observing the function behavior. The PLOTNSOLVE element is an important example of the significant ways in which the user can increase the degree of automation of the root-finding system. The PLOTNSOLVE element invokes the half interval method when the function takes on both positive and negative values in a plot, and hence combines two of the three steps in the root-finding process: search for starting values and invocation of mathematical algorithms. This example also shows the importance of pattern recognition. The five roots found by the NAPSS routine don't really tell much of a story--such as the existence of an

T#01 INPUT FUNCTION:

F=SIN(1/(X-5))

T#01 SYNTAX ERROR AT CHARACTER 14:
MISSING RIGHT PARAN ON FUNCTION SIN
(EITHER CORRECT YOUR DEFINITION OF F, OR ENTER NEW COMMAND ON BOTTOM LINE)
F=SIN(1/(X-5))

Figure 4.5. Function with a pattern of zeroes

T#01 MAXIMUM F: H .
-1.986693144E-01

EVALS: 33

XH:
0.000000000E+00

XL:
1.000000000E+00

MINIMUM F:
-2.474039197E-01

XMIN: 0.000000000E+00

XMAX: 1.000000000E+00

T#01 MAXIMUM F: H
-6.661725044E-02

EVALS: 66

XH:
-1.000000000E+01

XL:
-4.440892099E-16

MINIMUM F:
-1.986693144E-01

XMIN: -1.000000000E+01

XMAX: 0.000000000E+00

Figure 4.5 (Continued)

T#01 MAXIMUM F: H
-9.523663670E-03

EVALS: 99
XH:
-1.000000000E+02
XL:
-1.000000000E+01

MINIMUM F:
-6.661725044E-02
XMIN:-1.000000000E+02

L
XMAX:-1.000000000E+01

T#01 MAXIMUM F: H
-9.999499525E-06

EVALS: 132
XH:
-1.000000000E+05
XL:
-1.000000000E+03

MINIMUM F:
-9.950245731E-04
XMIN:-1.000000000E+05

L
XMAX:-1.000000000E+03

Figure 4.5 (Continued)

T#01 MAXIMUM F:
9.997837543E-01

EVALS: 165

XH:
5.645161290E+00

XL:
4.193548387E+00

MINIMUM F:
-9.457839727E-01

XMIN: 1.000000000E+00

XMAX: 1.000000000E+01

T#01 IN HI METHOD, NO CONVERGENCE AFTER MAXIMUM NUMBER OF ITERATIONS SPECIFIED:

X	F(X)	ITERATIONS
4.681689948E+00	-1.638083619E-06	25

EVALS: 192

XH:
5.645161290E+00

XL:
4.193548337E+00

MINIMUM F:
-9.457839727E-01

HI 1 10 0 0 XMIN: 1.000000000E+00

XMAX: 1.000000000E+01

Figure 4.5 (Continued)

T#01 IN HI METHOD, INTERVAL ENDPOINTS BECAME IDENTICAL TO FIFTEEN DECIMAL PLACES

	X	F(X)	ITERATIONS
	4.681690114E+00	-8.719671245E-16	56

EVALS: 250

XH: 5.645161290E+00

XL: 4.193548387E+00

MINIMUM F:

-9.457839727E-01

HI 1 10 0 100 IN: 1.000000000E+00

XMAX: 1.000000000E+01

T#01 **** ELEMENT ACCEPTED ****

APPROACH POINT START;

PLOT START (POINT+START)/2;

APPROACH POINT (POINT+START)/2;

APPROACH 5 4

Figure 4.5 (Continued)

T#01 MAXIMUM F: H . .
-3.894183040E-01

EVALS: 448
XH:
2.500000000E+00
XL:
3.750000000E+00

MINIMUM F:
-7.173560358E-01
XMIN: 2.500000000E+00

XMAX: 3.750000000E+00

T#01 MAXIMUM F: H . .
-7.173560858E-01

EVALS: 481
XH:
3.750000000E+00
XL:
4.354838710E+00

MINIMUM F:
-9.997837543E-01
XMIN: 3.750000000E+00

XMAX: 4.375000000E+00

Figure 4.5 (Continued)

T#01 MAXIMUM F:
9.996449351E-01

EVALS: 580

XH:

4.909274194E+00

XL:

4.871471774E+00

MINIMUM F:

-9.972934723E-01

XMIN: 4.843750000E+00

XMAX: 4.921875000E+00

T#01 MAXIMUM F:
9.996466041E-01

EVALS: 613

XH:

4.942036290E+00

XL:

4.929435484E+00

MINIMUM F:

-9.994130731E-01

XMIN: 4.921875000E+00

XMAX: 4.960937500E+00

Figure 4.5 (Continued)

T#01 **** ELEMENT ACCEPTED ****

```
PLOTNSOLVE POINT START;  
PLOT START (POINT+START)/2;  
IF F(XL(PLOT))*F(XH(PLOT))<=0 THEN HI START (POINT+START)/2 0 100;  
PLOTNSOLVE POINT (POINT+START)/2;
```

PLOTNSOLVE 5 4

T#01 MAXIMUM F: H
-8.414709568E-01

EVALS: 33
XH:
4.000000000E+00
XL:
4.370967742E+00

MINIMUM F:
-9.998204708E-01

XMIN: 4.000000000E+00

XMAX: 4.500000000E+00

Figure 4.5 (Continued)

T#01 MAXIMUM F:
7.568024397E-01

EVALS: 68

XH:

4.750000000E+00

XL:

4.500000000E+00

MINIMUM F:

-9.092974268E-01 L

XMIN: 4.500000000E+00

XMAX: 4.750000000E+00

T#01 IN HI METHOD, INTERVAL ENDPOINTS BECAME IDENTICAL TO FIFTEEN DECIMAL PLACES

X	F(X)	ITERATIONS
4.681690114E+00	-8.719671245E-16	51

EVALS: 123

XH:

4.750000000E+00

XL:

4.500000000E+00

MINIMUM F:

-9.092974258E-01 L

CONT

XMIN: 4.500000000E+00

XMAX: 4.750000000E+00

Figure 4.5 (Continued)

T#01 MAXIMUM F: H
 9.994507432E-01 . . ' '
 EVALS: 156
 XH:
 4.786290323E+00 -----
 XL:
 4.87C967742E+00
 MINIMUM F:
 -9.945987463E-01
 XMIN: 4.750000000E+00 XMAX: 4.875000000E+00

T#01 IN HI METHOD, INTERVAL ENDPOINTS BECAME IDENTICAL TO FIFTEEN DECIMAL PLACES

	X	F(X)	ITERATIONS
	4.840845057E+00	6.103769872E-15	50

 EVALS: 210
 XH:
 4.786290323E+00 -----
 XL:
 4.870967742E+00
 MINIMUM F:
 -9.945987463E-01
 CONT XMIN: 4.750000000E+00 XMAX: 4.875000000E+00

Figure 4.5 (Continued)

T#01 MAXIMUM F:
9.996449351E-01

EVALS: 243

XH:

4.909274194E+00

XL:

4.929435484E+00

MINIMUM F:

-9.994130731E-01

XMIN: 4.875000000E+00

XMAX: 4.937500000E+00

Figure 4.5 (Continued)

infinite number of roots, and the pattern of the roots.

The function $F=(X-5)**2/1E12$ is not shown here but was solved using both the NAPSS automatic system and the interactive system. When 10 roots were requested, the NAPSS routine returned six roots clustered around $X=5$, some designated as double roots and some as single roots, after 3170 function evaluations. The root at $X=5$ was found accurate to the number of places displayed (ten) with the interactive system using the SEC and MIN elements. Extensive investigation with the NHOOD element showed no clustered roots. The function was evaluated 708 times.

The following function was also examined with both systems: $F=ABS(X-1E-8)*(COS(1/X)+2)$. The NAPSS system took much longer than the interactive system when asked to find 10 roots, and much shorter when asked to find just one root (see Table 4.1). The interactive system was more accurate, finding the root by both the secant and minimization methods.

In the next function example, with $F=(X-1)*(X-2)*(X-3)*(X-4)*(X-5)$, the NAPSS routine was far superior when the actual number of roots (five) was requested, although much less efficient when the actual number of roots was not known ahead of time (see Table 4.1).

Figure 4.6 shows the definition step of two functions; the first involving scalar parameters, vectors of parameters, and summations, and the second a generalized step, or partitioned domain, function.

T#01 INPUT FUNCTION:

$F = \text{SUM}(I=1 \text{ TO } N, \text{SUM}(J=I \text{ TO } N \text{ BY } 2, A(I) * \text{VELOCITY} ** J / C(J)))$

T#01 PLEASE IDENTIFY THE INDEPENDENT VARIABLE:

VELOCITY

Figure 4.6. Definitions of two functions

T#01 PLEASE DEFINE THE VALUE OF SCALAR N

5

T#01 PLEASE DEFINE VALUES (SEPARATED BY 1 OR MORE BLANKS) FOR VECTOR A
(PRECEED VALUES BY SUBSCRIPT OF FIRST VECTOR ELEMENT
ENCLOSED IN PARANS IF IT IS OTHER THAN 1)

2.5 -.00613 15.1 0 3.14159265358979

Figure 4.6 (Continued)

T#01 PLEASE DEFINE VALUES (SEPARATED BY 1 OR MORE BLANKS) FOR VECTOR C
(PRECEED VALUES BY SUBSCRIPT OF FIRST VECTOR ELEMENT
ENCLOSED IN PARANS IF IT IS OTHER THAN 1)

31.57 , , 14.001 , , 91E13

Figure 4.6 (Continued)

T#01 INPUT FUNCTION:

F=STEP(SQRT(-X) IF $X < 0$, X^{**2} IF $0 \leq X \leq 10$, X^{**3} OTHERWISE)

Figure 4.6 (Continued)

Figure 4.7 shows an element called SHOWVALUE which can be used for evaluating the function at a sequence of points.

Table 4.1 contains a summary of all results and comparisons with the NAPSS system.

T#01 **** ELEMENT ACCEPTED ****

SHOWVALUES X INCREMENT;
SHOW X+INCREMENT*I F(X+INCREMENT*I),I=0,1,...;

SHOWVALUES 1 0.1

T#01
X+INCREMENT*I 1.000000E+00
F(X+INCREMENT*I) 0.000000E+00

Figure 4.7. Element for evaluating a function at a sequence of points

T#01
X+INCREMENT*I 1.100000E+00
F(X+INCREMENT*I) 1.934010E+00

T#01
X+INCREMENT*I 1.200000E+00
F(X+INCREMENT*I) 3.064320E+00

Figure 4.7 (Continued)

T#01
X+INCREMENT*I 1.300000E+00
F(X+INCREMENT*I) 3.566430E+00

Figure 4.7 (Continued)

Table 4.1. Results of solving some example functions with the interactive and NAPSS systems

Function		No. of function evaluations	No. of roots requested	No. of roots found
$(X-12E3) \cdot (X+21E6) \cdot 1E-10$				
	Interactive:	900 ^c		2
	NAPSS:	1690	10	2
	NAPSS:	1620	2	2
$1/X$				
	Interactive:	200		0
	NAPSS:	2290	10	1
$1/\text{MAX}(X , 1E-6)$				
	Interactive:	200		0
	NAPSS:	1310	10	0
$ X-17 ^2 \cdot X-17.1 ^{1.8} \cdot (X-20)$				
	Interactive:	1080		3
	NAPSS:	5020	10	4
	NAPSS:	1090	3	3

^aCorrect to 10 places if no decimal point given.

^bNAPSS cpu time is for batch; for on-line cpu time, multiply by six, since would be in low speed main storage.

^cRounded to nearest tens.

^dLabeled a possible discontinuity.

Roots ^a	cpu ^b time (sec.)	Real time (min.)	Remarks
12E3,-21E6	7.0	22	
12E3,-21E6	0.9		
12E3,-21E6	0.9		
	1.5	13	Left of origin not checked thoroughly
-.147E-7 ^d	1.5		Reported many "proba- ble discontinuities"
	1.5	15	
	0.8		"Search terminated-- excessive number of asymptotic conditions"
17,17.10000000,20	14.1	25	
20.000000, 17.099992, 17.000073, 17.000003	10.0		
20.000000, 17.099992, 17.000073	1.2		

Table 4.1 (Continued)

Function	No. of function evaluations	No. of roots requested	No. of roots found
$ X-361 ^{0.7} X+157.2 ^{1.5} (X-1E-6) / (X-1E-11)$			
Interactive:	1130		3
NAPSS:	4770	10	3
NAPSS:	1250	3	3
$\sin(1/(X-5))$			
Interactive:	1070		2
NAPSS:	8870	10	5
$(X-5)^2 / 1E12$			
Interactive:	1090		1
NAPSS:	3170	10	6
NAPSS:	1350	2	2
$ X-1E-8 \cdot (\cos(1/X) + 2)$			
Interactive:	3400		1
NAPSS:	6700	10	2
NAPSS:	390	1	1
$(X-1) \cdot (X-2) \cdot (X-3) \cdot (X-4) \cdot (X-5)$			
Interactive:	1560		5
NAPSS:	5070	10	5
NAPSS:	215	5	5

Roots ^a	cpu ^b time (sec.)	Real time (min.)	Remarks
361,-157.20000, 1E-6	11.6	26	
-157.19998, 361.200000, 1.0000034E-6	10.7		
Same as above	2.1		
4.6816901, 4.8408451	10.2	35	Any number of roots could have easily been found
5.3183099, 5.1591549			
4.6810901, 5.0636620 4.8938967	9.2		
5	7.1	17	
5.0000006, 4.9999983, 5.0000004, 5.00000002, 5.0000877, 4.9999458	1.8		
5.0000006, 4.9999983	0.1		
1E-8	20.1	22	
1.1605E-8, 1.7125E-4	6.2		
1.1605E-8	0.2		
1,2,3,4,5	11.0	10	
1,2.0000003,3,4,5	6.1		
Same as above	0.2		

CHAPTER 5

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

Conclusions

In this research the man-computer communication gap has been examined, and various approaches to this problem presented. One such approach--high-level interactive problem-solving systems--is characterized in Chapter 2. The components of such a system are identified and discussed. This should greatly ease the task of designing and building an interactive system for a specific application.

Design and implementation of a graphic interactive root-finding system is described in Chapter 3 as an example of the type of system defined in Chapter 2. The usefulness of interaction is demonstrated in the examples in Chapter 4. Other uses for this system not demonstrated in Chapter 4 include finding function minima and maxima using the MIN element or by finding the zeroes of the derivative of a function, and graphic investigation of various functions.

The root-finding system was designed to be convenient and natural for users whose mathematical and programming backgrounds vary widely. To help attain this goal, people in the following positions have used the system during its development for test and/or practical purposes:

- (1) undergraduate students with certain scientific backgrounds and little or no

knowledge of computers;

- (2) graduate students majoring in computer science and taking a course in interactive numerical problem-solving systems;
- (3) a professor of mathematics with specialties in real and functional analysis, with no knowledge of computers;
- (4) several scientists from the Ames Laboratory (U.S. Atomic Energy Commission, Ames, Iowa) with computing knowledge ranging from slight to extensive.

The root-finding system should be easily extendible because of the modularity through the "element approach" used in the system, and emphasis on the system documentation and readability. In fact the elements SAVE and CONT were added long after the rest of the system was built. These elements were added to allow printing of the displays for inclusion in this dissertation, although they will be of use to other users as well. Total time taken by the author to design and add these two elements was about four man-hours.

Another root-finding algorithm, Muller's method, was added to the system by a programmer who was familiar with PL/I but unfamiliar with the root-finding system. This addition of a basic element was accomplished, with no help from the author, in approximately five man-hours. The results are given in Appendix B.

A copy of the entire root-finding system program is available at the Computer Science library, Iowa State University, Ames, Iowa.

Future Work

There are several areas in which the root-finding system could be augmented or improved. Facilities allowing the user to save his function and his user-defined elements for use at a later date are highly desirable. Such facilities should also include the ability to display previously defined elements.

The ability to accept user input during execution of a user-defined element would be useful. This would involve a change in the element extension language. Such a capability would allow the user to defer certain decisions he now must make at either element definition time or element invocation time. An example of this is the decision of checking the neighborhood of a root. A PLOTNSOLVE element might then be defined as

```
PLOTNSOLVE XMIN XMAX;
PLOT XMIN XMAX;
IF F(XL(PLOT))*F(XH(PLOT))>0 THEN PLOTNSOLVE
    XMAX XMAX+(XMAX-XMIN);
HI XMIN XMAX 0 0;
INPUT FLAG;
IF FLAG=0 THEN PLOTNSOLVE XMAX XMAX+(XMAX-XMIN);
```

INPUT DELTA;

NHOOD X(HI) DELTA;

where NHOOD is defined as in Chapter 4.

A technique which was not incorporated in the root-finding system and yet may be useful in future systems is the use of a special communication routine to handle all communication between the system and user (see Pyle (1965)). Compiler-compilers, mentioned in Chapters 2 and 3, would also be useful in future systems, especially when means of specifying the semantics of a given language become better understood (see Schorre (1964) and O'Neil (1968)).

There are many numerical problem areas in which interactive systems might be beneficial. Root-finding in the complex plane has been investigated in this manner by Larkin (1964) and by Fried (1966). Smith (1969) briefly mentions the areas of function minimization, ordinary and partial differential equations, and solution of systems of linear equations. Numerical integration is another area which should benefit considerably from an interactive system.

BIBLIOGRAPHY

- Ball, W. E. and Berns, R. I. 1966. AUTOMAST - automatic mathematical analysis and symbolic translation. Communications of the Association for Computing Machinery 9: 626-633.
- Burgess, E., ed. 1965. Symposium on on-line computing systems. Detroit, Michigan, American Data Processing.
- Burley, H. T. 1967. A programming language for linear algebra. Computer Journal 10: 69-73.
- Cardenas, A. F. 1968. A problem-oriented language and a translator for partial differential equations. Unpublished Ph.D. dissertation. Los Angeles, Ca., Library, University of California. (Xerox copy No. 69-11878, Ann Arbor, Mich., University Microfilm).
- Cardenas, A. F. and Karplus, W. J. 1970. PDEL - a language for partial differential equations. Communications of the Association for Computing Machinery 13: 184-191.
- Champagne, W. P., Jr. 1965. On finding roots of polynomials by hook or by crook. Unpublished M.S. thesis. Austin, Texas, Library, University of Texas.
- Chasen, S. H. and Seitz, R. N. 1967. On-line systems and man-computer graphics. Computers and Automation 16, No. 11: 22-28.
- Coons, S. A. 1955. Graphics and differential equations. Journal of Engineering Education 46: 555-556.
- Culler, G. J. 1968. Mathematical laboratories: a new power for the physical sciences. In Klerer, M. and Reinfelds, J., eds. Interactive systems for experimental applied mathematics. Pp. 355-384. New York, New York, Academic Press, Inc.
- Culler, G. J. and Fried, B. D. 1964. TRW on-line scientific computer - general description. In Sass, M. A. and Wilkinson, W. D., eds. Computer augmentation of human reasoning. Pp. 65-87. Washington, D.C., Spartan Books.
- Culler, G. J. and Huff, R. W. 1962. Solution of nonlinear integral equations using on-line computer control. American Federation of Information Processing Societies Spring Joint Computer Conference Proceedings 20: 129-138.

- Dill, C., Ellis, C. A., Gear, C. W., and Ratliff, K. 1968. The automatic integration package for ordinary differential equations. University of Illinois Computer Science Department Report No. COO-1469-0103.
- Dixon, W. J. 1967. Use of displays with packaged statistical programs. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 31: 481-484.
- Edwin, L. R. and Edwin, A. I. 1968. Nonlinear interactive stepwise regression analysis in a time-sharing environment. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 33, Part 1: 127-139.
- Evans, T. G. 1965. Machine-aided design of context-free grammars. National Conference, Association for Computing Machinery Proceedings 20: 344-353.
- Falkoff, A. D. and Iverson, K. E. 1968. APL/360 User's Manual. Yorktown Heights, New York, IBM Corporation.
- Feldman, J. and Gries, D. 1968. Communications of the Association for Computing Machinery 11: 77-113.
- Fraser, J. B. 1967. On the role of natural language in man-machine communication. In Walker, D. E., ed. Third Congress on Information System Science and Technology. Pp. 21-31. Washington, D.C., Thompson Book Co.
- Fried, B. D. 1966. On-line root-finding in the complex plane. TRW Systems Report No. 9990-7308-R00000.
- Fried, G. D. 1968. Summary. In Klerer, M. and Reinfelds, J., eds. Interactive systems for experimental applied mathematics. Pp. 462-465. New York, New York, Academic Press, Inc.
- Gawlik, H. J. 1963. MIRFAC: a compiler based on standard mathematical notation and plain English. Communications of the Association for Computing Machinery 6: 545-548.
- IBM Corporation. 1969a. Conversational programming system. Hawthorne, New York, IBM Corporation.
- IBM Corporation. 1969b. Multiple terminal monitor task for 2260. Hawthorne, New York, IBM Corporation.

- Iverson, K. E. 1962. A programming language. New York, New York, John Wiley and Sons.
- Kaplow, R., Brackett, J., and Strong, S. 1966. Man-machine communication in on-line mathematical analysis. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 29: 465-477.
- Karplus, W. J., ed. 1967. On-line computing - time-shared man-computer systems. New York, New York, McGraw-Hill Book Co.
- Kulsrud, H. E. 1968. A general purpose graphic language. Communications of the Association for Computing Machinery 22: 247-254.
- Larkin, H. E. 1964. A combined graphical and iterative approach to the problem of finding zeros of functions in the complex plane. Computer Journal 7: 212-219.
- Licklider, J. C. R. 1965. Man-computer partnership. International Science and Technology 41: 18-26.
- Licklider, J. C. R. and Clark, W. E. 1962. On-line man-computer communication. American Federation of Information Processing societies Spring Joint Computer Conference Proceedings 20: 113-128.
- Machover, C. 1967. Graphic CRT terminals - characteristics of commercially available equipment. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 30: 149-159.
- Marchuk, G. I. and Yershov, A. P. 1966. Man-machine interaction in solving a certain class of differential equations. International Federation of Information Processing Societies Congress Proceedings 2: 550-551.
- Miller, R. B. 1969. Archetypes in man-computer problem-solving. Institute of Electrical and Electronic Engineers Transactions on Man-machine Systems MMS-10, No. 4: 219-241.
- Mills, R. G. 1967. Man-machine communication and problem-solving. In Cuadra, C. A., ed. Annual review of information science and technology. Vol. 2. Pp. 223-254. New York, New York, John Wiley and Sons.
- Newell, A. and Simon, H. A. 1964. Problem-solving machines. International Science and Technology 48: 48-62.

- Nickerson, R. S., Elkind, J. I., and Carbonell, J. R. 1968. Human factors and the design of time-sharing computer systems. *Human Factors* 10: 127-134.
- O'Neil, J. R. 1968. META-PI: An on-line interactive compiler-compiler. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 33, Part 1: 201-218.
- Pyle, I. C. 1965. Data input by question and answer. *Communications of the Association for Computing Machinery* 8: 223-226.
- Reinfelds, J., Eskelson, N., Kopetz, H., and Kratky, G. 1970. AMTRAN - an interactive system. American Federation of Information Processing Societies Spring Joint Computer Conference Proceedings 36: 537-541.
- Rice, J. R. 1967. On the construction of polyalgorithms for automatic numerical analysis. Purdue University Computer Science Department Report No. CSD TR 10.
- Rice, J. R. 1969a. A polyalgorithm for the automatic solution of nonlinear equations. National Conference, Association for Computing Machinery Proceedings 24: 179-183.
- Rice, J. R. 1969b. A polyalgorithm for the automatic solution of nonlinear equations. Purdue University Computer Science Department Report No. CSD TR 32.
- Rice, J. R. 1969c. A set of 74 test functions for nonlinear equation solvers. Purdue University Computer Science Department Report No. CSD TR 34.
- Rice, J. R. and Rosen, S. 1966. NAPSS - a numerical analysis problem-solving system. National Conference, Association for Computing Machinery Proceedings 21: 51-56.
- Sackman, H. 1968. Time-sharing versus batch processing: the experimental evidence. American Federation of Information Processing Societies Spring Joint Computer Conference Proceedings 32: 1-10.
- Schlesinger, S. and Sashkin, L. 1967. POSE: a language for posing problems to a computer. *Communications of the Association for Computing Machinery* 10: 279-285.
- Schorre, D. V. 1964. META-2: a syntax-oriented compiler writing language. National Conference, Association for Computing Machinery Proceedings 19: PD1.3-1.

- Seitz, R. N., Wood, L. H., and Ely, C. A. 1968. AMTRAN: automatic mathematical translation. In Klerer, M., and Reinfelds, J., eds. Interactive systems for experimental applied mathematics. Pp. 44-66. New York, New York, Academic Press, Inc.
- Shaw, J. C. 1964. JOSS: a designer's view of an experimental on-line computing system. American Federation of Information Processing Societies Fall Joint Computer Conference Proceedings 25: 455-464.
- Smith, L. B. 1969. The use of man-machine interaction in data-fitting problems. Stanford Linear Accelerator Center Report No. 96.
- Symes, L. R. 1969. Evaluation of NAPSS expressions involving polyalgorithms, functions, recursion, and untyped variables. Purdue University Computer Science Department Report No. 33.
- Theys, M. E. 1970. An overview of the META compiler systems. Unpublished M.S. paper. Ames, Iowa, Computer Science Library, Iowa State University.
- Wegner, P. 1968. Programming languages, information structures, and machine organization. New York, New York, McGraw-Hill Book Co.
- Whitbeck, R. F. 1964. Graphical analysis of nonlinear systems. Unpublished Ph.D. thesis. New York, N. Y., Library, Cornell University.
- Zaguskin, V. L. 1961. Handbook of numerical methods for the solution of algebraic and transcendental equations. New York, New York, Pergamon Press.

ACKNOWLEDGMENTS

The author wishes to sincerely thank Professors R. J. Lambert and R. F. Keller for their guidance in this research and help in completion of this dissertation.

The author also wishes to thank the scientists and students who used the root-finding system in its various stages of development. In particular, thanks are due to Professor J. C. Kegley for his help in making the system intelligible to non-programmers, and to Mr. A. Lucido for his work in adding an element to the system.

The able typing of this dissertation by Mrs. Jody Jacobson, Louise, and by Mrs. Mary Brown is gratefully acknowledged.

APPENDIX A: TUTORIAL INFORMATION FOR
THE ROOT-FINDING SYSTEM

* * WELCOME * * THIS IS AN EXPERIMENTAL INTERACTIVE ROOT-FINDING SYSTEM. YOU ARE ABLE TO GUIDE THE COMPUTER THROUGH THE BASIC STEPS OF FINDING ROOTS OF AN EQUATION. MORE PRECISELY, THE ROUTINE IS TO BE USED TO FIND THE ZEROS OF A SINGLE REAL-VALUED FUNCTION OF A REAL VARIABLE, AS SUPPLIED BY YOU, THE USER.

EACH TIME YOU HAVE READ A SCREEN-FULL OF INFORMATION, OR PAGE, PUSH THE BUTTON MARKED 'ENTER' TO GO TO THE NEXT PAGE. (IF YOU ARE ALREADY FAMILIAR WITH THIS SYSTEM, YOU MAY BEGIN YOUR INPUT AT ANY TIME).

NOTE: PLEASE START ANY INPUT IN THE FIRST POSITION OF A LINE.

* * * T U T O R: I N T R O * * *

A NOTE ABOUT STOPPING AND OVERRIDING THE SYSTEM: TO STOP NORMALLY, ENTER THE COMMAND 'STOP' (WITHOUT THE QUOTES, AS ALWAYS); TO STOP ABNORMALLY, IF THE SYSTEM IS IN A LOOP AND WILL NOT ACCEPT INPUT, PRESS THE START BUTTON (UPPER CASE), THEN INPUT THE COMMAND '%CANCEL 1'. YOU MAY AT ANY TIME INTERRUPT EXECUTION OF AN ELEMENT BY INPUTTING A NEW COMMAND ON THE BOTTOM LINE.

THIS SYSTEM CONSISTS OF A NUMBER OF BASIC STEPS, CALLED (BASIC) ELEMENTS. THESE ELEMENTS ARE INVOKED BY YOU AT ANY TIME OR IN ANY ORDER BY MERELY TYPING IN THE ELEMENT NAME AND ANY PARAMETERS REQUIRED BY THE ELEMENT.

THE BASIC ELEMENTS IN THIS SYSTEM ARE LISTED BELOW. FOLLOWING THIS LIST, SOME OF THE MORE INVOLVED ELEMENTS ARE DESCRIBED IN DETAIL.

* * * T U T O R: I N T R O

* * *

ELEMENT	**DESCRIPTION**
F	ALLOWS USER TO INPUT HIS FUNCTION
PLOT	PLOTS FUNCTION F
HI	HALF INTERVAL METHOD OF FINDING A ROOT
SEC	SECANT METHOD OF FINDING A ROOT
MIN	MINIMIZATION METHOD OF FINDING A ROOT
SETTIMER	SETS A CPU-TIME (ACTUAL COMPUTER TIME) TIMER
CHECKTIMER	CHECKS THE TIMER SET BY THE SETTIMER ELEMENT
SHOW	DISPLAYS INPUT OR OUTPUT VALUES OF ANY ELEMENT, ANY VALUE OF F, AND ANY ELEMENT ARITHMETIC EXPRESSIONS

* * * T U T O R: I N T R O

* * *

ELEMENT	**DESCRIPTION**
TUTOR	DISPLAYS THE TUTOR YOU ARE NOW READING
ELEMENT	ALLOWS USER TO COMBINE THE EXISTING ELEMENTS INTO ANOTHER (USER- DEFINED) ELEMENT
SAVE	STORES THE CURRENT DISPLAY IN AUXILIARY STORAGE; THESE DISPLAYS MAY THEN BE PRINTED OR PUNCHED IF DESIRED
CONT	CAUSES CONTINUATION OF EXECUTION OF LAST-EXECUTED USER ELEMENT
STOP	SHUTS DOWN THE SYSTEM

ALL BUT THE TUTOR, SAVE, CONT, AND STOP ELEMENTS WILL NOW BE DESCRIBED
IN DETAIL.

* * * T U T O R : F * * *
 F THIS ELEMENT ALLOWS YOU TO DEFINE THE FUNCTION YOU WISH TO FIND
 THE ZEROS OF. JUST TYPE THE LETTER F AND PUSH THE ENTER BUTTON,
 AND THE COMPUTER WILL RETURN A BLANK SCREEN TO ALLOW YOU TO INPUT YOUR FUNCTION.

THE FUNCTION MUST ALWAYS BE NAMED F. IT IS DEFINED IN TERMS OF AN
 ARITHMETIC EXPRESSION, WHICH IS ANY ARITHMETICAL COMBINATION (+ - * / ** WHERE
 ** MEANS 'RAISED TO THE POWER OF') OF THE FOLLOWING, EACH PRECEDED BY EXAMPLES:

3, -4.2, 1.5E10	CONSTANTS
X, W, SAM	THE INDEPENDENT VARIABLE
PRESSURE, A, C, TEMP	PARAMETERS

* * * T U T O R : F * * *
 D(I), VAL(3), T(X/3+1) VECTORS OF PARAMETERS
 SIN, LOG, MAX STANDARD MATH FUNCTIONS
 Y=X+3; OTHER (USER-DEFINED) FUNCTIONS
 STEP(0 IF X<0, 3*X IF X>=0) (GENERALIZED) STEP FUNCTIONS
 SUM(I=1 TO N, VAL(I)*X) SUMMATIONS
 PROD(J=2 TO 50 BY 2, J*X) PRODUCTS

*** TUTOR: F
SOME EXAMPLE FUNCTION DEFINITIONS ARE:

F=X+3.7;
F=W*COS(2*W)**2+C3;
F=STEP(-X IF X<=0,EXP(X) IF 0<X<1,3 OTHERWISE);
Z=COS(X)+PI*(X+7); F=PROD(C=-3 TO 3,C*Z*D(C));

ALL OPERATIONS IN AN ARITHMETIC EXPRESSION ARE DONE IN A SPECIFIED ORDER,
CONFORMING GENERALLY TO THE ORDER IMPLIED IN STANDARD MATH NOTATION;

*** TUTOR: F ***
THE OPERATIONS ARE DONE FROM LEFT TO RIGHT IN THE FOLLOWING ORDER:
PARENTHETICAL EXPRESSIONS
MATHEMATICAL FUNCTIONS, SUCH AS LOG, COS, STEP
**
* AND /
+ AND -

THUS THE EXPRESSION A*B*C**D**E+F/COS(G)*H
IS EVALUATED AS INDICATED IN THE FOLLOWING PARENTHETICAL VERSION:

((A*B)*((C**D)**E)) + ((F/(COS(G)))*H)

* * * T U T O R: F * * *

A CONSTANT IS ANY SIGNED (+ OR -) OR UNSIGNED SEQUENCE OF SIXTEEN OR FEWER DIGITS. IT MAY OPTIONALLY INCLUDE A DECIMAL POINT. A CONSTANT MAY ALSO OPTIONALLY INCLUDE A BASE 10 INTEGER EXPONENT (BETWEEN -78 AND 75, INCLUSIVE) TO ALLOW FOR SCIENTIFIC NOTATION:

FOR EXAMPLE, -3.5×10^{15} WOULD BE WRITTEN AS $-3.5E15$ OR $-3.5E+15$
AND $0.036 \times 10^{(-5)}$ WOULD BE WRITTEN AS $0.036E-5$

ALL NAMES (VARIABLES, FUNCTIONS, PARAMETERS) CONSIST OF TEN OR FEWER LETTERS AND/OR DIGITS, AND MUST BEGIN WITH A LETTER. PARAMETER VECTORS MAY HAVE NEGATIVE, ZERO, AND/OR POSITIVE SUBSCRIPTS.

* * * T U T O R: F * * *

THE STANDARD MATH FUNCTIONS AVAILABLE ARE (FOR ARGUMENTS X,Y,...):

FUNCTION

ABS(X)

CEIL(X)

FLOOR(X)

MAX(X,Y,...)

MIN(X,Y,...)

MOD(X,Y)

SIGN(X)

TRUNC(X)

DEFINITION

ABSOLUTE VALUE OF X

SMALLEST INTEGER $\geq X$

LARGEST INTEGER $\leq X$

MAXIMUM OF X,Y,...

MINIMUM OF X,Y,...

X MODULO Y (REMAINDER OF X/Y)

HAS A VALUE OF -1 IF $X < 0$, 0 IF $X = 0$, 1 IF $X > 0$

X WITHOUT ITS FRACTIONAL PART

```

      * * *   T U T O R :   F   * * *
**FUNCTION**      **DEFINITION**
ATAN(X)           ARCTANGENT OF X IN RADIANS
ATAN(X,Y)         ARCTANGENT OF X/Y IN RADIANS
ATAND(X)          ARCTANGENT OF X IN DEGREES
ATAND(X,Y)        ARCTANGENT OF X/Y IN DEGREES
ATANH(X)          HYPERBOLIC ARCTANGENT OF X
COS(X),SIN(X),TAN(X) COSINE, SINE, TANGENT OF X, X IN RADIANS
COSD(X),SIND(X),TAND(X) COSINE, SINE, TANGENT OF X, X IN DEGREES
COSH(X),SINH(X),TANH(X) HYPERBOLIC COSINE, SINE, TANGENT OF X
ERF(X)            ERROR FUNCTION OF X
ERFC(X)           COMPLEMENT OF ERF(X) (1-ERF(X))

```

```

      * * *   T U T O R :   F   * * *
**FUNCTION**      **DEFINITION**
EXP(X)            EXPONENTIAL OF X (E RAISED TO X POWER)
LOG(X),LOG2(X),LOG10(X) LOGARITHM (BASE E, 2, OR 10) OF X
SQRT(X)           SQUARE ROOT OF X

```

* * * T U T O R: F

* * *

GENERALIZED STEP, OR PARTITIONED DOMAIN, EXPRESSIONS ARE WRITTEN AS A SEQUENCE OF ONE OR MORE DOMAIN SPECIFICATIONS SEPARATED BY COMMAS. A DOMAIN SPECIFICATION MAY BE OF THE FORM:

<AE> IF <AE><OP><AE>

OR, <AE> IF <AE><OP><AE><OP><AE>

OR, <AE> OTHERWISE

WHERE <AE> MEANS ARITHMETIC EXPRESSION AND <OP> MEANS ONE OF THE BOOLEAN OPERATORS:

< LESS THAN

<= LESS THAN OR EQUAL TO

= EQUAL TO

> GREATER THAN

>= GREATER THAN OR EQUAL TO

≠ NOT EQUAL TO

EXAMPLES:

$X \cdot \cos(3 \cdot X)$ IF $X \leq 0.2$

$\log(\text{ABS}(X))$ IF $X \neq 0$

10.375 IF $0.2 < X \leq \sqrt{Y} + 1$

$X^{2.5}/W$ OTHERWISE

* * * T U T O R: F

* * *

SOME EXAMPLES OF STEP EXPRESSIONS ARE:

STEP(0 IF $X < 0$, X OTHERWISE)

STEP($X \cdot \cos(3 \cdot X)$ IF $X \leq 0.2$, 10.375 IF $0.2 < X \leq \sqrt{Y}$, $X^{2.5}/W$ OTHERWISE)

STEP($\log(Y)$ IF $Y > 0$)

THE LAST EXAMPLE IS NOT DEFINED FOR $Y \leq 0$, AND THE SYSTEM WOULD REPORT THIS TO THE USER SHOULD AN ATTEMPT BE MADE TO EVALUATE IT FOR $Y \leq 0$. HOWEVER THE FOLLOWING EXPRESSION, THOUGH NOT WELL-DEFINED MATHEMATICALLY, IS LEGAL AND HAS A VALUE OF $2 \cdot 8 = 16$ FOR $X = 8$: STEP($2 \cdot X$ IF $X < 10$, $\sqrt{X}$ IF $5 \leq X$)

*** TUTOR: F

SUMMATIONS AND PRODUCTS (CAPITAL SIGMA AND PI IN MATH NOTATION) ARE WRITTEN SIMILARLY, HENCE WE WILL ONLY LOOK AT SUM IN DETAIL. SUM IS WRITTEN

SUM(<VAR>=<AE> TO <AE>,<AE>)

OR, SUM(<VAR>=<AE> TO <AE> BY <AE>,<AE>)

WHERE <VAR> IS ANY VARIABLE NAME, AND <AE> IS AN ARITHMETIC EXPRESSION. IF THE 'BY <AE>' IS OMITTED, 'BY 1' IS ASSUMED; THE VARIABLE IS INCREMENTED BY THE AMOUNT SPECIFIED IN THIS 'BY CLAUSE', WHICH MUST BE A POSITIVE QUANTITY.

SOME EXAMPLES ARE:

REMARKS:

SUM(I=1 TO 3, I**2)

HAS VALUE OF 1**2+2**2+3**2=14

PROD(Y=X TO X+1 BY 0.1*TEMP, COS(Y))

SUM(I=1 TO N, SUM(J=1 TO M, A(I)*C(J))) HERE A AND C ARE PARAM. VECTORS

*** TUTOR: F

FUNCTION STATEMENTS MUST CONTAIN NO BLANKS, EXCEPT WHERE REQUIRED - ONE BLANK ON EACH SIDE OF THE CONTROL WORDS 'IF', 'BY', 'TO', AND ONE BLANK PRECEDING THE CONTROL WORD 'OTHERWISE'. EACH FUNCTION STATEMENT MUST BE FOLLOWED BY A SEMICOLON (;), AND FUNCTION STATEMENTS MAY BE SEPARATED BY ONE OR MORE BLANKS.

F MUST BE THE LAST FUNCTION SPECIFIED. FOR EXAMPLE, F=Y**2, WHERE Y=X+3, MUST BE WRITTEN AS Y=X+3; F=Y**2;.

* * * T U T O R : F * * *

ONE LAST ITEM - SHOULD YOU EVER WANT TO RECALL YOUR DEFINITION OF F, SIMPLY INPUT 'F ?' (WITHOUT THE QUOTES, AS ALWAYS).

AFTER INPUTTING YOUR FUNCTION DEFINITION, PUSH ENTER. THE COMPUTER WILL CHECK YOUR DEFINITION FOR CORRECTNESS, AND ASK YOU TO IDENTIFY THE INDEPENDENT VARIABLE (IT CAN'T TELL IT APART FROM THE PARAMETERS). THEN IT WILL REQUEST VALUES FOR EACH OF THE PARAMETERS.

THIS CONCLUDES THE DESCRIPTION OF THE FUNCTION DEFINITION ELEMENT, F.

* * * T U T O R : P L O T * * *

PLOT XMIN XMAX THE PLOT ELEMENT IS USED TO PLOT $F(X)$ (IF, SAY, X IS THE INDEPENDENT VARIABLE) FROM $X=XMIN$ TO $X=XMAX$. THE INPUTS, XMIN AND XMAX, MUST BE CONSTANTS, AND SEPARATED BY ONE OR MORE BLANKS. THE OUTPUTS ARE THE PLOT, AND

XL: VALUE OF X WHERE F TOOK ON THE SMALLEST VALUE

XH: VALUE OF X WHERE F TOOK ON THE LARGEST VALUE

EVALS: TOTAL NUMBER OF FUNCTION EVALUATIONS TO DATE

NOTE: AVOID PLOTTING SUCH SMALL INTERVALS THAT XMIN AND XMAX AGREE TO MORE THAN FOURTEEN DECIMAL DIGITS.

* * * T U T O R: H I * * *
 HI XMIN XMAX EPS MAXITERS THE HI ELEMENT IS THE HALF-INTERVAL METHOD OF
 FINDING A ROOT. THE INPUTS INDICATE AN
 INTERVAL IN WHICH THE SUSPECTED ROOT LIES, AND TWO CONVERGENCE CRITERIA:
 XMIN: LEFT ENDPOINT OF INTERVAL
 XMAX: RIGHT ENDPOINT OF INTERVAL
 EPS: CONVERGENCE WILL BE SIGNALLED WHEN $ABS(F(X)) < EPS$
 MAXITERS: METHOD WILL ITERATE AT MOST MAXITERS TIMES

IF EPS AND/OR MAXITERS ARE INPUT AS ZERO IN ANY OF THE ELEMENTS HI, SEC, OR MIN,
 IF EPS AND/OR MAXITERS ARE INPUT AS ZERO IN ANY OF THE ELEMENTS HI, SEC, OR MIN,
 DEFAULT VALUES OF $EPS=1E-20$ AND $MAXITERS=100$ WILL BE SUPPLIED BY THE SYSTEM.

* * * T U T O R: H I * * *
 THE HALF-INTERVAL METHOD REQUIRES THAT YOU SUPPLY XMIN AND XMAX SUCH THAT
 $F(XMIN)$ AND $F(XMAX)$ DIFFER IN SIGN. THEN THE FUNCTION IS EVALUATED AT
 THE MIDPOINT OF THE INTERVAL. THE NEW INTERVAL (NEXT ITERATION) IS GIVEN EITHER
 BY XMIN AND THE MIDPOINT, IF $F(XMIN)$ AND $F(MIDPOINT)$ DIFFER IN SIGN, OR BY THE
 MIDPOINT AND XMAX, IF $F(MIDPOINT)$ AND $F(XMAX)$ DIFFER IN SIGN. THE PROCESS IS
 THEN REPEATED WITH THIS NEW (HALVED) INTERVAL. CONVERGENCE OCCURS WHEN AN
 INTERVAL IS FOUND SUCH THAT ITS MIDPOINT SATISFIES $ABS(F(MIDPOINT)) < EPS$.

THIS METHOD WILL CONVERGE TO EITHER A ROOT OF F OR A VALUE OF X AT WHICH
 THERE IS A JUMP DISCONTINUITY ACROSS THE X -AXIS. BECAUSE OF SUCH CONSIDERATIONS

* * * T U T O R: H I

* * *

AS WHETHER THE ROOT IS REPRESENTABLE EXACTLY IN THE COMPUTER AND ACCURACY OF FUNCTION F EVALUATION IN THE COMPUTER, THE USER MAY NEVER KNOW WHETHER HE HAS FOUND A ROOT OR A DISCONTINUITY. FOR EXAMPLE, AFTER THE INTERVAL BECOMES SO SMALL THAT THE ENDPOINTS AGREE TO SIXTEEN DECIMAL PLACES, THE METHOD CAN GO NO FURTHER. AT THIS POINT, A MESSAGE AND THE LAST ITERATES (X,F(X)) ARE DISPLAYED. X WILL BE THE ROOT OR DISCONTINUITY CORRECT TO ABOUT SIXTEEN DECIMAL PLACES.

THIS PROBLEM IS NOT UNIQUE TO INTERACTIVE ROUTINES - ANY ROOT-FINDING PROGRAM WOULD HAVE THE SAME PROBLEM.

* * * T U T O R: H I

* * *

OUTPUTS OF THE HALF INTERVAL ELEMENT ARE:

X: FINAL VALUE OF THE ROOT CANDIDATE
EVALS: TOTAL NUMBER OF FUNCTION EVALUATIONS TO DATE
ITERS: NUMBER OF ACTUAL ITERATIONS
FLAG: =0 IF METHOD CONVERGED
 =1 IF THE ENDPOINTS AGREE TO SIXTEEN PLACES
 =2 IF F DOES NOT DIFFER IN SIGN AT ENDS OF INITIAL INTERVAL

* * * T U T O R: SEC * * *
 SEC XSTART XSTOP EPS MAXITERS THE SEC ELEMENT IS THE SECANT METHOD OF
 FINDING A ROOT. THIS METHOD IS CONVENIENT
 WHEN THE HALF-INTERVAL METHOD CANNOT BE USED BECAUSE OF NO SIGN CHANGE NEAR THE
 SUSPECTED ROOT. GIVEN TWO POINTS, SAY A AND B, THE SECANT METHOD PASSES A
 STRAIGHT LINE THROUGH THE POINTS (A,F(A)) AND (B,F(B)). THE INTERSECTION OF
 THIS LINE WITH THE X-AXIS, SAY AT POINT C, IS THE NEW ESTIMATED ROOT.
 IF $ABS(F(C)) \geq EPS$, THEN THE PROCEDURE IS REPEATED ON B AND C.

* * * T U T O R: SEC * * *
 THE METHOD REQUIRES TWO POINTS A AND B TO GET STARTED. XSTART IS USED FOR
 A, AND A POINT 1/5 OF THE WAY FROM XSTART TO XSTOP IS USED FOR B. THE METHOD
 TERMINATES WITH NO CONVERGENCE IF ANY ITERATE SHOULD EVER GET OUTSIDE THE
 INTERVAL DETERMINED BY XSTART AND XSTOP. THIS INDICATES THE PRESENCE OF AT
 LEAST ONE 'WIGGLE' (NON-MONOTONIC SLOPE) IN THE GIVEN INTERVAL. THE AREA SHOULD
 THEN BE GRAPHICALLY EXAMINED MORE CLOSELY FOR POSSIBLE BETTER STARTING VALUES.

INPUTS FOR THE SEC ELEMENT ARE:

XSTART: EXPLAINED ABOVE
 XSTOP: EXPLAINED ABOVE
 EPS: CONVERGENCE WHEN $ABS(F(X)) < EPS$
 MAXITERS: MAXIMUM NUMBER OF ITERATIONS ALLOWED

```

      * * * T U T O R:  SEC      * * *
OUTPUTS FOR THE SEC ELEMENT ARE:
      X:  FINAL VALUE OF ROOT CANDIDATE
      EVALS:  TOTAL NUMBER OF FUNCTION EVALUATIONS TO DATE
      ITERS:  NUMBER OF ACTUAL ITERATIONS
      FLAG:  =0 IF METHOD CONVERGED
              =1 IF FUNCTION VALUES BECAME TOO CLOSE TOGETHER TO ALLOW
                DIVISION BY THEIR DIFFERENCE (REQUIRED BY THE METHOD)
              =2 IF X PASSED XSTART
              =3 IF X PASSED XSTOP

```

```

      * * * T U T O R:  MIN      * * *
MIN  XMIN  XMAX  EPS  MAXITERS  THE MIN ELEMENT ATTEMPTS TO MINIMIZE
                                ABS(F(X)) IN THE INTERVAL DETERMINED BY XMIN
AND XMAX.  THIS IS DONE BY STEPPING THROUGH THE INTERVAL IN ONE DIRECTION,
SAMPLING ABS(F) AT THESE STEP POINTS UNTIL AN INCREASE IN ABS(F) IS FOUND.  AT
THIS POINT, THE DIRECTION OF SAMPLING IS REVERSED AND THE STEP SIZE IS REDUCED
AND THE PROCEDURE REPEATED UNTIL ABS(F(X)) < EPS OR THE SAMPLING INTERVAL
BECOMES TOO SMALL.

```

THIS METHOD, OF COURSE, MAY FIND ONLY A LOCAL MINIMUM AND NOT THE TRUE MINIMUM IN THE GIVEN INTERVAL. HENCE CAREFUL EXAMINATION OF THE INTERVAL SHOULD FOLLOW REPORTED CONVERGENCE.

*** T U T O R: M I N ***
INPUTS AND OUTPUTS FOR THE MIN ELEMENT ARE THE SAME AS FOR SEC EXCEPT:
FLAG: =0 IF METHOD CONVERGES
 =1 IF SAMPLING INTERVAL LENGTH BECOMES SO SMALL THAT ENDPOINTS
 AGREE TO SIXTEEN PLACES.

*** T U T O R: S E T T I M E R ***
SETTIMER TOTALTIME THE SETTIMER ELEMENT SETS A CPU-TIME TIMER TO THE NUMBER
 OF SECONDS INDICATED BY INPUT VALUE OF TOTALTIME. THIS
TIMER WILL COUNT DOWN ONLY WHEN THE CPU (CENTRAL PROCESSING UNIT OF THE
COMPUTER) IS WORKING ON YOUR JOB. THIS TIME, OF COURSE, IS CONSIDERABLY LESS
THAN THE TOTAL AMOUNT OF (WALL-CLOCK) TIME FOR WHICH YOU ARE SIGNED ON, YET IS
THE MOST EXPENSIVE (ABOUT 10 CENTS/SECOND COMPARED TO ABOUT \$7/HOUR FOR
WALL-CLOCK TIME).

INDIRECT OUTPUT IS A FLAG CALLED EXPIRE WHICH IS SET TO ZERO (AND A MESSAGE
IS GIVEN) WHEN THE TIMER AMOUNT HAS EXPIRED.

* * * T U T O R: CHECKTIMER * * *
 CHECKTIMER THIS ELEMENT CHECKS THE CPU-TIME TIMER AS LAST SET BY THE SETTIMER
 ELEMENT. THERE ARE NO INPUTS.
 OUTPUTS ARE:
 TOTALTIME: TOTAL TIME LAST SET USING SETTIMER ELEMENT
 TIMEUSED: CPU TIME USED SINCE TIMER WAS LAST SET BY SETTIMER ELEMENT

* * * T U T O R: ELEMENT * * *
 ELEMENT THE ELEMENT NAMED ELEMENT ALLOWS YOU TO DEFINE A NEW ELEMENT IN
 TERMS OF ANY BASIC OR USER-DEFINED ELEMENTS. THIS ALLOWS YOU TO
 INCREASE THE DEGREE OF AUTOMATION OF THE SYSTEM AT ANY TIME OR IN ANY PHASE OF
 THE ROOT-FINDING PROCESS. THIS FACILITY IS THE MOST POWERFUL FEATURE IN THIS
 SYSTEM. HENCE IT IS WELL WORTH LEARNING HOW TO DEFINE YOUR OWN ELEMENTS.

FOR EXAMPLE, SUPPOSE YOU WOULD LIKE TO SEARCH FOR ROOTS GRAPHICALLY IN THE
 INTERVALS (0,1), (1,2), (2,3), ... TO DO THIS YOU WOULD HAVE TO SEQUENTIALLY
 INPUT THE COMMANDS PLOT 0 1, PLOT 1 2, PLOT 2 3, ...
 BUT BY DEFINING A NEW 'USER-DEFINED' ELEMENT, CALL IT 'SEARCH', ALL THIS CAN BE
 DONE WITH ONE COMMAND.

* * * T U T O R: E L E M E N T * * *
 ONE WAY OF DEFINING SUCH A 'SEARCH' ELEMENT IS:
 SEARCH XMIN XMAX;
 PLOT XMIN+I XMAX+I, I=0,1,...;
 THEN THE ABOVE PLOTS WILL BE GENERATED BY THE FOLLOWING SINGLE COMMAND:
 SEARCH 0 1
 EACH SUCCEEDING PLOT WILL BE GENERATED WITH ONE PUSH OF THE ENTER BUTTON.

AS A SECOND EXAMPLE, SUPPOSE YOU WOULD LIKE THE HALF INTERVAL METHOD
 AUTOMATICALLY INVOKED EACH TIME THE FUNCTION TAKES ON POSITIVE AND NEGATIVE
 VALUES IN A PLOT.

* * * T U T O R: E L E M E N T * * *
 THE FOLLOWING ELEMENT WOULD DO THIS JOB:
 PLOTNSOLVE X Y;
 PLOT X Y;
 IF F(XL(PLOT))*F(XH(PLOT))<=0 THEN HI XL(PLOT) XH(PLOT) 1E-30 0;
 THEN THE COMMAND
 PLOTNSOLVE 1 2
 WILL GENERATE A PLOT FROM X=1 TO X=2; IF THE FUNCTION DIFFERED IN SIGN AT
 XL AND XH, THE LOW AND HIGH POINTS OF THE PLOT, THE PRODUCT OF THESE TWO
 VALUES WILL BE <= 0 AND THE HI ELEMENT WILL BE INVOKED WITH XL(PLOT) AND
 XH(PLOT) AS STARTING VALUES, EPS=1E-30, AND MAXITERS=100 (DEFAULT VALUE).

* * * T U T O R: E L E M E N T * * *

AS THE ABOVE EXAMPLES INDICATE, AN ELEMENT DEFINITION CONSISTS OF A 'SAMPLE REFERENCE' STATEMENT, FOLLOWED BY ANY NUMBER OF ELEMENT REFERENCE STATEMENTS AND 'IF' STATEMENTS. THE SAMPLE REFERENCE STATEMENT DEFINES THE NAME OF THE NEW ELEMENT AND THE NAMES OF ANY INPUT PARAMETERS.

AN ELEMENT REFERENCE IS OF THE FORM:

<ELEMENT NAME> <PARAMETER LIST>;

OR, <ELEMENT NAME> <PARAMETER LIST>,<REPETITION>;

WHERE <PARAMETER LIST> IS A SEQUENCE OF ELEMENT ARITHMETIC EXPRESSIONS, ONE EXPRESSION FOR EACH INPUT PARAMETER REQUIRED BY THE ELEMENT BEING REFERENCED.

* * * T U T O R: E L E M E N T * * *

AN ELEMENT ARITHMETIC EXPRESSION IS ANY ARITHMETIC COMBINATION OF ELEMENT INPUTS, ELEMENT OUTPUTS, VALUES OF F, AND CONSTANTS. AN ELEMENT MAY CONTAIN NO MATH FUNCTIONS, SUCH AS LOG. ELEMENT INPUTS AND OUTPUTS ARE ELEMENT INPUT OR OUTPUT NAMES, FOLLOWED BY THE CORRESPONDING ELEMENT NAME IN PARENTHESES.

<REPETITION> ALLOWS THE ELEMENT REFERENCE TO BE REPEATED. ITS FORM IS:

<VARIABLE>=<CONSTANT>,<CONSTANT>,...;

OR, <VARIABLE>=<CONSTANT>,<CONSTANT>,...,<CONSTANT>;

```

      * * * T U T O R:  E L E M E N T      * * *
SOME EXAMPLES OF ELEMENT REFERENCES ARE:
    HI XL(PLOT) XH(PLOT) 1E-30 0;
    PLOT XMIN+I XMAX+I, I=0,1,...;
    PLOT X-DEL X+DEL, DEL=5,10,...,25;
    SHOW X+INCR F(X+INCR), INCR=0,-1,...;

```

(NOTE THAT IF A NAME REFERS TO AN INPUT OF THE ELEMENT CURRENTLY BEING DEFINED, IT NEED NOT BE FOLLOWED BY THE ELEMENT NAME IN PARENTHESES).

159

```

      * * * T U T O R:  E L E M E N T      * * *
AN 'IF' STATEMENT IS OF THE FORM:
    IF <AE><OP><AE> THEN <ELEMENT REFERENCE>;
    OR, IF <AE><OP><AE> THEN <ELEMENT REFERENCE>; ELSE <ELEMENT REFERENCE>;
WHERE <AE> IS AN ELEMENT ARITHMETIC EXPRESSION AND <OP> IS A BOOLEAN
OPERATOR (< <= > >= = !=).
SOME 'IF' STATEMENTS ARE:
    IF F(XL(PLOT))*F(XH(PLOT))<=0 THEN HI XL(PLOT) XH(PLOT) 0 200;
    IF EXPIRE(SETTIMER)=0 THEN STOP;
    IF FLAG(HI)>0 THEN SEC XMIN XMAX 0 0; ELSE PLOT XMIN+10 XMAX+10;

```

* * * T U T O R : E L E M E N T * * *

NOTES: ELEMENT DEFINITION STATEMENTS MAY CONTAIN NO BLANKS EXCEPT ONE BLANK
BETWEEN ANY TWO NAMES;
THE STATEMENTS MAY BE SEPARATED BY ONE OR MORE BLANKS, BUT MUST BE
TERMINATED WITH A SEMICOLON;
EXECUTION OF AN ELEMENT MAY BE TERMINATED AT ANY TIME BY INPUTTING
A NEW COMMAND.

THIS CONCLUDES THE DESCRIPTION OF THE 'ELEMENT' ELEMENT.

* * * T U T O R : S H O W * * *

SHOW LIST; THE SHOW ELEMENT DISPLAYS ALL ELEMENT ARITHMETIC EXPRESSIONS
LISTED. NOTE THE LIST MUST BE FOLLOWED BY A SEMICOLON.

THE FOLLOWING SHOW REFERENCE:
SHOW X(HI) F(X(HI)) FLAG(HI);

SHOW X+INCR F(X+INCR), INCR=0,1,...
SHOW F((X+Y)/2);

GIVES A DISPLAY OF:
CERTAIN RESULTS (OUTPUTS) FROM THE
HI ELEMENT
X,F(X); X+1,F(X+1); X+2,F(X+2); ...
F AT MIDPOINT OF THE INTERVAL (X,Y)

*** T U T O R: INPUT ***
YOU MAY NOW PROCEED WITH YOUR INPUT. REMEMBER YOU CAN ALWAYS RETURN
TO THIS TUTOR BY INPUTTING THE WORD 'TUTOR'. *** GOOD LUCK ***

APPENDIX B: ADDITION OF A NEW ELEMENT
TO THE SYSTEM

A new basic element, MULLER, was added to the root-finding system for the purpose of demonstrating the readability and extendibility of the system.

This element was added to the system by a programmer who was familiar with the PL/I language but entirely unfamiliar with the root-finding system program and the algorithm (Muller's method) used in the MULLER element. The addition was completed, with no help from the author, in approximately five man-hours:

Reading and modifying the root-finding system program (2.5 hours)

Converting the batch version of Muller's method to use in the interactive system (0.5 hours)

Writing the required tutorial information (0.3 hours)

Keypunching (1.2 hours)

Running the root-finding system and tutor-generator programs (0.5 hours).

The new (augmented) system was successfully generated in the second computer run of the program--the first run uncovered one syntax error and one keypunching error.

Both the batch and interactive versions of the Muller algorithm are given in the following pages. Also shown is the code added to the element interpreter in the control routine. Not shown are changes required in various variable declarations and initialization values.


```

IF X2=X1 THEN
DO; PUT EDIT('      IN MULLER'S METHOD, ITERATES BECAME '||
'IDENTICAL TO FIFTEEN PLACES')(COL(1),A); FLAGMU=1;
GO TO DUMP_MULLER;
END;

T1=(X2-X1)/(X1-X0); T2=T1+1; T3=F0*T1*T1-F1*T2*T2+F2*(T2+T1);
T4=SQRT(ABS(T3*T3-4*X2*T1*T2*(F0*T1-F1*T2+F2)));
IF T3<0 THEN T4=-T4; T1=-2*F2*T2/(T3+T4);
X0=X1; X1=X2; X2=X1+(X1-X0)*T1;
F0=F1; F1=F2; F2=F(X2);
GO TO AGAIN_MUL;

DUMP_MULLER: PUT EDIT
('      X      F(X)      ITERATIONS',
X2,F2,ITERSMU) (COL(1),A,COL(1),2 E(21,9),F(10));
EVALSMU=IFN; XMU=X2;
RETURN;

END MULLER;

```

MULLER'S METHOD AS USED IN INTERACTIVE ON-LINE MODE
(CONVERTED FROM BATCH VERSION FOR USE IN THE
INTERACTIVE ROOT-FINDING SYSTEM)

MULLER: ENTRY;

MULLER'S METHOD REQUIRES THREE STARTING POINTS, SAY X_0 , X_1 , AND X_2 ; THE NEXT ITERATE IS CHOSEN AS ONE OF THE TWO ROOTS OF THE SECOND DEGREE POLYNOMIAL PASSING THROUGH THE POINTS (X_0, F_0) , (X_1, F_1) , AND (X_2, F_2) . IF THE FUNCTION IS $\geq$ EPSMU IN ABSOLUTE VALUE AT THE NEW ITERATE, THE PROCESS IS REPEATED USING X_1 , X_2 , AND THE NEW ITERATE AS THE THREE STARTING POINTS.

```

XMINMU:  ONE END OF STARTING INTERVAL
XMAXMU:  OTHER END OF STARTING INTERVAL
EPSMU:   CONVERGENCE WILL BE SIGNALLED WHEN  $ABS(F(X)) < EPSMU$ 
MAXITERSMU:  METHOD WILL ITERATE AT MOST MAXITERSMU TIMES

```

IF EPSMU AND/OR MAXITERSMU ARE INPUT AS ZERO, DEFAULT VALUES OF
1E-20 AND 100 WILL BE SUPPLIED IN THE CONTROL ROUTINE, RESP.

OUTPUTS

XMU: FINAL VALUE OF THE ROOT CANDIDATE
EVALSMU: TOTAL NUMBER OF FUNCTION EVALUATIONS TO DATE
ITERSMU: NUMBER OF ACTUAL ITERATIONS
FLAGMU: =0 IF METHOD CONVERGED
 =1 IF TWO SUCCESSIVE ITERATES BECOME TOO CLOSE TOGETHER
 =2 IF NO CONVERGENCE AFTER MAXITERSMU ITERATIONS

INTERNAL VARIABLES

T1,T2,T3,T4: TEMPORARIES USED TO CALCULATE THE NEW ITERATE, X2
X0,X1,X2: LAST THREE ITERATES
F0,F1,F2: VALUE OF THE FUNCTION AT THE LAST THREE ITERATES

166

*/

ITERSMU=0; X0=XMINMU; X1=XMAXMU; X2=(X1+X0)/2;
RETURN=MULLERS_1; X=X0; GO TO FN; MULLERS_1: F0=F;
RETURN=MULLERS_2; X=X1; GO TO FN; MULLERS_2: F1=F;
RETURN=MULLERS_3; X=X2; GO TO FN; MULLERS_3: F2=F;

AGAIN_MUL:

IF ITERSMU>MAXITERSMU THEN

DO; BRC(1)= ' IN MULLER'S METHOD, NO CONVERGENCE AFTER' ||
' MAX. NO. OF ITERATIONS SPECIFIED:';
FLAGMU=1; GO TO DUMP_MULLER;
END;

ITERSMU=ITERSMU+1;

IF ABS(F2)<EPSMU THEN

DO; BRC(1)= ' MULLER'S METHOD CONVERGED:';
FLAGMU=0; GO TO DUMP_MULLER;
END;

IF X2=X1 THEN

DO; BRC(1)= ' IN MULLER'S METHOD, ITERATES BECAME ' ||
' IDENTICAL TO FIFTEEN PLACES'; FLAGMU=1;
GO TO DUMP_MULLER;
END;

T1=(X2-X1)/(X1-X0); T2=T1+1; T3=F0*T1*T1-F1*T2*T2+F2*(T2+T1);

T4=SQRT(ABS(T3*T3-4*X2*T1*T2*(F0*T1-F1*T2+F2)));

IF T3<0 THEN T4=-T4; T1=-2*F2*T2/(T3+T4);

X0=X1; X1=X2; X2=X1+(X1-X0)*T1;

F0=F1; F1=F2; RETURN=MULLERS_4; X=X2; GO TO FN; MULLERS_4: F2=F;

GO TO AGAIN_MUL;

DUMP_MULLER: BRC(2)=

' X F(X) ITERATIONS';

PUT STRING(BRC(3)) EDIT(X2,F2,ITERSMU)(COL(1),2 (E(21,9),F(10)));

EVALSMU=IFN; XMU=X2;

RETURN; /* END MULLER */