NAVORD REPORT 4209

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

8 FEBRUARY 1956

U. S. NAVAL ORDNANCE LABORATORY

WHITE OAK, MARYLAND

NAVORD Report 4209

Aeroballistic Research Report 326

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

Prepared by:

E. K. Blum

ABSTRACT: Automatic Digital Encoding System II (ADES II) is a system for the automatic translation of mathematical formulas into programs of coded instructions for an electronic digital computer. ADES represents a new approach to the problem of automatic programming in that it is applicable to almost all modern computers, and it is not limited to a special class of mathematical problems. The system consists of a formulation language, an Encoder, and a digital computer. The formulation language closely resembles ordinary mathematical language, and is based on the theory of recursive functions. The Encoder is an automatic device which receives a mathematical formulation as input and produces the necessary computer instructions as output. The logical design of an Encoder is given. It consists of a computer with certain routines loaded into its storage. Certain desirable features of the computer in such a system are suggested.

U. S. NAVAL ORDNANCE LABORATORY
WHITE OAK, MARYLAND

NAVORD Report 4209 8 February 1956

This report contains the results of research directed toward expediting the process of preparing, or programming, mathematical problems for computation on high-speed electronic digital computers. The advent of the automatic computer which performs computations at electronic speed, has revolutionized the art of computation, but it has raised problems of its own. For many mathematical problems, the speed of computation is so great and the process of programming so involved, that the time spent on the computer in useful computation is often just a fraction of the total time elapsed between the formulation of the problem and the obtainment of the numerical results.

This report presents a system designed to correct this situation. The system relieves the mathematician of a major portion of the programming task. In effect, it replaces the human by an automatic device which carries out the programming duties. The design for such a device is given. An experimental model is in process of construction.

This work was performed under NOL Task No. FR-30-1-56.

W. W. WILBOURNE
Captain, USN
Commander

H. H. KURZWEG
By direction

PREFACE

At the September 1955 Annual Meeting of the Association for Computing Machinery, the author presented a paper which described Automatic Digital Encoding System I, (ADES I), a system of automatic translation of mathematical formulas into computer programs. The present paper describes an extension of that system, ADES II, which significantly enlarges the class of problems which can be automatically programmed. It is conjectured that ADES II is sufficiently general to cope with most of the two-dimensional mathematical problems that are submitted to existing digital computers.

ADES I excluded problems which involved double recursion. This is now provided for in ADES II, which is also capable of programming a special kind of triple recursion. However, ADES II harbors certain obvious limitations which can and should be removed. Further, there is room for much improvement in procedure, for example, in input-output procedure. Finally, there remains the deeper problem of efficiency,both in time and storage utilization. The minimalization or efficiency problem has thus far been treated as secondary in urgency. These considerations motivate plans for another system, ADES III, now in the research stages.

The ADES approach to automatic programming is believed to be entirely new. Mathematically, it has its foundations in the bedrock of the theory of recursive functions. The proposal to apply this theory to automatic programming was first made by C. C. Elgot, a former colleague of the author's. While at the Naval Ordnance Laboratory, Elgot did some research on a language for automatic programming. Some of his ideas were adapted to ADES, and we wish to acknowledge this fundamental contribution.

It is our belief that the theory of recursive functions, as propounded in references [1] , [2] , and [3] say, and the closely related mathematical logic provide the natural, possibly the only, framework for constructing a general automatic coding system for digital computation. However, it appears that these subjects are not yet within the purview of the majority of those mathematicians presently concerned with preparing problems for an electronic digital computer. Since this report is directed primarily at them, ADES II is presented here as a practical system for the translation of mathematical problems into computer programs, rather than as an application derived from the theory of metamathematics. The mathematical logician will find no theorems to justify the methods adopted, and the sporadic references to recursive function theory are intentionally in an intuitive vein. The style is consciously expository and heuristic rather than telegraphic. However we do assume that the reader has a good mathematical background. ADES is designed primarily for the mathematician.

This report is in two parts, under separate covers. The first part presents definitions and rules for formulating problems in a stylized mathematical language. The second part describes the logical design of the 'Encoder', a machine which will receive the mathematical formulation as input and yield a complete computer program as output. The logical mechanism of the Encoder becomes rather intricate in places. Therefore, minute details are often omitted, since it is felt that the mathematician versed in machine techniques will be able to supply the missing details, in one form or another, once he has grasped the overall pattern. In fact, he can modify the design in many ways, for there usually is more than one means to an end. Many features in the present design were suggested by the author's colleague, Leroy Krider. Mr. Krider and the author are working on the construction of an actual Encoder for ADES II, which will utilize a 650 Magnetic Drum Calculator. A joint report describing this Encoder and experiences with its operation will be published in the near future.

Many thanks are due to Larry Schmid for his critical reading of the manuscript and helpful suggestions. We also thank Mrs. M. Zawatzky for her assistance in preparing the manuscript for publication.

CONTENTS

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

Introduction. In the last few years, we have witnessed the advent of automatic computing machines which perform digital computation at electronic speed. In fact, the speed of computation is so great that for many mathematical problems, the time spent on the machine is but a small fraction of the total time elapsed between the formulation of the problem and the obtainment of the numerical results.

Much of the time and effort required to solve any moderately complex mathematical problem by means of an automatic digital computer is consumed in the process commonly referred to as 'programming'.

Programming can be described as that process which starts with the problem as a set of statements in mathematical language (i.e. equations), coupled with a set of metamathematical statements in ordinary language (e.g. English), and transforms these two sets into a single set of instructions coded in a language intelligible to the machine. This set of coded instructions, known as the 'program', is a list of the successive computer operations required to obtain the numerical results.

To elaborate, programming can be considered to take place in two stages. The first stage involves the writing of explicit mathematical formulas which define the quantities to be computed, indicating the order in which these quantities are to be computed, (i.e. the flow chart), and finally, specifying the numerical values to be assigned to the independent variables in these formulas. The writing of the formulas may be a matter of transcription, or it may entail some numerical analysis and algebra, for example, converting implicit equations into explicit ones, setting up a numerical integration, or an interpolation, and so on.

In the second stage of programming, the mathematical formulas obtained in the first stage, together with the metamathematical statements which specify the flow chart and the procedure for substituting numbers for the variables, are translated into the language of the machine. This translation is essentially a clerical process in which three main tasks are performed. (1) The computation is broken down into an ordered sequence of machine operations. These include arithmetic operations and logical operations, each of which is translated into a numerical code. (2) The numbers to be substituted for the independent variables, (i.e. the input data) are assigned storage locations, and their addresses are then combined with the appropriate operation codes from step (1) to form machine instructions. (3) Storage locations are allocated to the computed results, and machine instructions for the output of these results are given.

These three tasks of encoding are recognized to be quasi-mechanical, and, to a great extent, they can be assigned to a machine, preferably to the computer itself. Toward this end, various schemes such as compilers, library routines, interpretive routines, and pseudo-code systems have been devised. As yet, these schemes have fallen short of the immediate objective, which is to accept a mathematical formulation of a problem as obtained in the first stage of programming and produce a complete program for machine computation.

In the present paper, a system is presented, which, for the most part, realizes this objective for a wide class of problems. This system will be called 'Automatic Digital Encoding System, II' , (ADES,II). The Roman numeral, II, indicates both the incompleteness of the system and its extensibility. The incompleteness is of two kinds. First, certain components of the system are still in an unrefined though workable state, resulting in a loss of efficiency. Second, a class of mathematical problems cannot be encoded by system II, namely, those problems which necessarily involve arbitrary triple recursions or, more generally, recursions on k indices, where $k \geq 3$. However, the system does provide for a special kind of triple recursion which arises frequently in practice. Furthermore, there is no inherent reason for excluding the general triple recursion.

PART I

1. The Components

As conceived in this paper, an automatic encoding system consists of three primary components: a formulation language, an Encoder (i.e. a machine which does the encoding), and an electronic digital computer. The three components influence each other to such an extent that it is difficult to construct any one of them independently of the others. Seemingly, this is a disadvantage, since in most instances we are given the computer and required to design the other components around it. It would appear that each computer requires a system peculiar to itself. Actually, the computer does not play a critical role.

Fortunately, there are certain general principles, drawn from the theory of metamathematics, which are applicable to all digital computation. Thus, although ADES II has been set up for computers with certain characteristics, the system can be modified very easily to fit most digital computers. In particular, the formulation language and the logical design of the Encoder can be used universally with relatively minor changes, while the internal machinery of the Encoder can readily be adapted to different computers.

2. The Computer

To fix our ideas, we assume that the computer in ADES II is a single-address, sequenced-program, floating-point calculator which can perform the usual arithmetic and logical operations. It can also carry out a new operation which facilitates the metamathematical processes of the computer, i.e. those processes which compute and modify storage addresses. This operation is called 'Modify'.

The coded instruction, 'Modify n', causes the next instruction in sequence to be executed as if the number in storage n had been added to it. For example, if the number 230 is in storage n, then the pair of instructions, Modify n, Multiply 1300, is equivalent in effect to the single instruction, Multiply 1530, i.e. multiply by the number in 1530. Since n is arbitrary, this makes it possible to use any storage in the computer as a 'B-box', or 'index register'. Furthermore, the instruction, 'Modify n', converts the floating-point number in storage n into an integer suitably positioned for address modification. This eliminates the need for two kinds of arithmetic in the computer, and makes the mathematical formulation of a problem conceptually simpler. It also simplifies the machinery of the Encoder.

The above features are not absolutely essential in system II. However, they are convenient. In any case, they can be achieved by means of interpretive routines if the actual computer hardware is lacking.

3. Requirements of the Language

The formulation language is the critical component in any automatic coding system. In devising such a language, one is beset by conflicting demands. The language should resemble 'ordinary mathematical notation' as much as possible. It should not result in inefficient programs. It should not be so flexible that the Encoder becomes excessively complicated. Its alphabet and syntax should be rich enough to accomodate any effectively calculable function.

The first three properties depend largely on personal judgment. In the designing of system II, there was a tendency to choose a language which simplified the Encoder at the expense of efficiency in the computer program. The resulting inelegancies can be removed, and this will be done in system III. Most important is the fourth requirement, and to satisfy it, recourse was had to the theory of recursive functions (see [1], [2], [3]). This theory is pertinent, since the arithmetic and logical operations performed by a digital computer are, in effect, operations on integers, and what's more, they are 'primitive recursive' functions.

In the domain of positive integers, one distinguishes two kinds of functions, primitive recursive and general recursive. Intuitively speaking, a primitive recursive function is one which is obtained by composition of a finite sequence of arithmetic functions and recursions on one index. A general recursive function is obtained by adducing multiple recursions on k indices, where k can be as large as we please. Theoretically, multiple recursions, as such, can be eliminated if a new operator, μ, called the minimization operator, is included in the language, [1]. Thus,

$$\phi(x_1, \ldots, x_n) = \mu y \{ f(x_1, \ldots, x_n, y) = 0 \},$$

means that $\phi(x_1, \ldots, x_n)$ is the minimum value of y such that $f(x_1, \ldots, x_n, y) = 0$.

The minimization operator appears to be of limited practical value in eliminating multiple recursions. Hence, the language for system II provides for explicit double recursions, and a special type of triple recursion, as explained later.

In the next few sections, the language will be described semi-formally by presenting its alphabet and syntax in the form of definitions and rules, and illustrating them with numerous examples.

4. The Alphabet and Elementary Expressions

The alphabet is constructed from the nine generic symbols, q,r,a,b,c,f,e,p and d, the positive integers and zero. The usage of the generic symbols is indicated by their names:

q.....independent index
r.....dependent index
a.....independent variable (or data symbol)
b.....dependent variable (or defined variable)
c.....free variable
f.....function symbol
e.....equal sign, (=)
p.....punctuation symbol
d.....output symbol

The letters of the alphabet consist of the generic symbols with numerical subscripts attached. Thus, the alphabet consists of the independent indices $q_0,q_1,q_2,\ldots$, the dependent indices $r_0,r_1,r_2,\ldots$, the independent variables $a_0,a_1,a_2,\ldots$, and so on for the other symbols.

In what follows, we shall abbreviate, using 'variable' to denote either one of the independent variables $a_0,a_1,a_2,\ldots$, or one of the dependent variables $b_0,b_1,b_2,\ldots$, unless otherwise stated. Likewise, 'index' will mean either an independent or dependent index.

Def. 4.1. An indexed variable of degree one is an expression of the form, xi, where x denotes a variable and i denotes an index.

e.g. : a_1q_1, b_4q_5, $a_{21}r_{14}$, b_2r_7 .

An indexed variable of degree two is an expression of the form, xij, where x denotes a variable, and i and j denote indices.

e.g. : $a_3q_1q_2$, $b_2r_4q_9$, $b_1r_2r_3$, $a_5q_3r_2$.

An expression consisting simply of a variable is said to be an indexed variable of degree zero.

e.g. : a_1, b_2, a_4;

A numerical constant is a floating-point number, written in some standard form consisting of an exponent, a modulus, and algebraic sign. (Note: The precise form cannot be specified, since it depends on the computer. However, in this discussion, the exact form is irrelevant, and we shall write numerical constants in the form that is easiest to present.)

Definition 4.1 can be extended to indexed variables of arbitrary degree. However, in ADES, II, the maximum degree of a variable is two.

In the formulation of a problem, the independent variables are used to denote the various kinds of input data; e.g. a_0 might represent pressure, a_1 temperature, a_3 time, a_4 the elements of a given matrix, etc. If a_1 is to assume n numerical values as the computation proceeds, the n values being supplied as a row of data, then in the formulation this data must be represented as an indexed independent variable of degree one; e.g. $a_1 q_1$, where $q_1 = 0,1,2, \ldots, n-1$. (Thus, in that formulation, a_1 must always occur with one index.) For data supplied in the form of a two-dimensional matrix, we must use an independent variable of degree two; e.g. $a_4 q_2 q_3$, where q_2 and q_3 assume successive positive integral values between specified lower and upper bounds. A data constant must be denoted either by an indexed variable of degree zero, say a_6, or by its numerical value.

A computed quantity (i.e. an intermediate or final result) which is defined by a designated formula is represented by a dependent variable. If several values of a dependent variable, b_1, are to be computed, they can be distinguished, if necessary*, by using one or two indices; e.g. $b_1 q_1$, $b_1 q_1 q_2$.

It should be apparent from these remarks that the independent indices in a problem are to assume successive integral values as the computation proceeds. The lower and upper bounds of each independent index used must be specified explicitly. This specification is called 'quantification' of the index, and is an important part of the formulation. A quantification consists of a phrase 'for all integral values of q_1, such that $L_1 \leq q_1 \leq u_1$'. In this report, we shall use the quantification symbol, ' $\forall$ 'to represent 'for all', and a quantification will be written briefly in a form to suggest ' $\forall q_1, L_1 \leq q_1 \leq u_1$ '. Here, L_1 and u_1 denote independent variables or indices which specify the bounds of q_1. Rigorous rules for quantification will be given in section 8.

It is often necessary to have indices which do not take on successive values, or it may be convenient to have an index which is defined in terms of a quantified index. In such cases, dependent indices are used. For example, if a problem calls for an index which takes on only even integral values, we define $r_1 = 2 q_1$, where $q_1 = 0,1,2,\ldots,n$, and write $a_1 r_1$ instead of $a_1(2q_1)$. Other uses of the dependent indices will appear later.

5. Notation

In ADES, the formulas which define the dependent variables and dependent indices must be written in what is known as parenthesis-free, or prefixed-operator notation. In this concise notation, the formula

*The necessity depends on other factors to be explained later.

$(a_1 + a_2)$ is written as $+a_1a_2$, and $a_1 \cdot a_2$ is written as $\cdot a_1a_2$. In other words, each operator (i.e. function) symbol is written to the left of its operands, and all parentheses are eliminated since they are redundant. Note that juxtaposition, a_1a_2, does not connote multiplication. The centered dot is used for the multiplication operator.

For a unary operator, such as cosine, the prefixed-operator notation coincides with ordinary mathematical notation, provided that all parentheses are eliminated, e.g. cos x. A unary operator is said to be of degree 1, a binary operator (e.g. +, $\cdot$) is of degree 2, and in general, a function of n operands is of degree n. In the actual ADES II language, each function is denoted by the symbol f with identifying subscript. For example, f_2 denotes +, f_3 denotes − , f_4 denotes $\cdot$, and so on. However, to simplify the exposition, in this report we shall continue to use the conventional mathematical designations +, $\cdot$, cos etc., with a few exceptions. One of these is the identity function, which is usually not written explicitly in conventional mathematics. In ADES, it will be denoted by f_1, and must be written wherever called for by the syntactical rules. Thus, as we shall see, one must write $b_1 = f_1a_1$ instead of $b_1 = a_1$. Note that for expository reasons again, we shall write ' = ' for an ordinary equal sign instead of 'e$_{oo}$', as required in ADES. Likewise, in this paper, all common punctuation in formulas will be written in conventional form instead of using the symbol P with a subscript.

Now, to further illustrate the use of prefixed-operator notation, we list several examples written in both conventional and prefixed-operator notation.

Conventional	Prefixed-Operator
$(a_1 + a_2)\, a_3 \cdot$	$\cdot + a_1a_2a_3$
$a_3(a_1 + a_2)$	$\cdot a_3 + a_1a_2$
$\sqrt{[a_1(q_1) - a_2(r_1, r_2)]}$	$\sqrt{} - a_1q_1a_2r_1r_2$
$(q_2 + q_3)(r_1 - r_2)$	$\cdot + q_2q_3 - r_1r_2$
$\sin(b_1 + a_1(q_1))$	$\sin + b_1a_1q_1$
$2.(a_1 + 3.)$	$\cdot 2. + a_13.$

6. Well-formed formulas and terms.

An expression in parenthesis-free notation consisting of a juxtaposition of operators and operands is called a 'string' [4] . Not all strings are meaningful; e.g. $+a_2$, $\sin a_1b_1$, $+a_1a_2a_3$ are meaningless. A meaningful string is called a 'well-formed formula', (w.f.f.) . In essence, a string is a well-formed formula if for each operator in the string there is the correct number of operands. A rigorous definition is as follows.

If x is an operand, then x is a well-formed formula. If $y_1, y_2, \ldots, y_n$ are well-formed formulas and f_k is a function of degree n, then $f_k y_1 y_2 \ldots y_n$ is a well-formed formula.

In ADES, we use 'terms' rather than well-formed formulas.

Def. 6.1. An index term is a well-formed formula in which the operands are independent indices, $q_0, q_1, q_2, \ldots$, dependent indices, $r_0, r_1, r_2, \ldots$, indexed independent variables, or numerical constants, and in which there is at least one function symbol.

Note: The w.f.f., a_1, is not an index term, nor is the w.f.f., q_1. Instead, one must write $f_1 a_1$ and $f_1 q_1$, where f_1 is the identity function.

Def. 6.2. An index equation (or r-equation), is an expression of the form

$$j = \phi,$$

where j denotes one of the dependent indices $r_0, r_1, r_2, \ldots$, and ϕ denotes an index term. (Note the comma after ϕ.)

We illustrate this definition by examples written both in conventional and in ADES language.

Conventional	ADES
$r_1 = a_1(q_2),$	$r_1 = f_1 a_1 q_2,$
$r_0 = q_1 - a_1,$	$r_0 = -q_1 a_1,$
$r_4 = q_1(q_1 + a_1) + q_2,$	$r_4 = +\cdot q_1 + q_1 a_1 q_2,$
$r_5 = [a_1(q_1) + r_2] / q_7,$	$r_5 = /+ a_1 q_1 r_2 q_7,$
$r_6 = q_1(q_1 + a_1) / 2.,$	$r_6 = /\cdot q_1 + q_1 a_1 2.,$

Def. 6.3. A b-term is a well-formed formula in which the operands are indexed variables or numerical constants, and in which there is at least one function symbol.

Def. 6.4. A b-equation is an expression of the form,

$$y = \psi , ,$$

where y denotes one of the dependent variables $b_0, b_1, b_2, \ldots$, and ψ denotes a b-term.

e.g.

Conventional	ADES
$b_2 = a_2(q_1)$	$b_2 = f_1 a_2 q_1,$
$b_1 = [a_1(q_1) + a_2] a_3$	$b_1 = \cdot + a_1 q_1 a_2 a_3,$
$b_{17} = b_5(r_1, r_2) - a_1(q_1, q_2)$	$b_{17} = -b_5 r_1 r_2 a_1 q_1 q_2,$
$b_3 = (a_4 + 2.)$	$b_3 = + a_4 2.,$

It may be necessary to have an independent index, q_i say, as an operand in a b-equation. This is done by introducing an independent variable, a_i, and defining $a_i(q_j) = q_j$. The identification of a_i with q_j is done in the 'Computer Table', to be explained in section 13.

Def. 6.5. An integer-valued independent variable is an independent variable which assumes the values of a designated independent index. Integer-valued variables are permissible operands in b-equations.

The b-equations and the r-equations constitute the major part of the formulation of a problem, since they define what is to be computed. However, there is need for another kind of equation. Suppose a formula occurs several times in the same problem. It would sometimes be convenient to be able to write this formula once with 'free' or unspecified variables. Then each time the formula is used, it would be necessary to indicate only which data are to be substituted for the free variables. For this purpose, we adduce the 'free terms' and 'auxiliary equations' defined as follows.

Def. 6.6. A free term is a well-formed formula in which the operands are free variables or constants, and in which there is at least one function symbol.

Def. 6.7. An auxiliary equation is an expression of the form,

$$g = \phi, ,$$

where g denotes a function and ϕ a free term. The degree of g is equal to the number, n, of free variables in ϕ. If $x_1, \ldots, x_n$ denote permissible operands for an index term, then $gx_1, \ldots, x_n$ is the index term obtained when x_i is substituted for the free variable c_i in ϕ ; and similarly for b-terms. The function, g, on the left side of an auxiliary equation is called an 'auxiliary' function. In this report, we shall use subscripts of 50 or greater to designate auxiliary functions.

For example, suppose the index equations,

$$r_1 = q_1(q_4 + 1), \qquad r_2 = q_2(q_5 + 1.), \qquad r_3 = q_3(q_6 + 1.),$$

occur in the same problem. If we define an auxiliary function, f_{50} say, by the auxiliary equation,

$$f_{50} = \cdot c_1 + c_2 1.,$$

then we can write the index equations as

$$r_1 = f_{50} q_1 q_4,$$
$$r_2 = f_{50} q_2 q_5,$$
$$r_3 = f_{50} q_3 q_6, \cdot$$

Note the following important point, which we emphasize as a Rule.

Rule 6.1. Each equation in an ADES formulation must be punctuated by a comma at the right end of the equation.

The functions used in an ADES formulation are of two types; the auxiliary functions defined above, and 'library' functions. To explain what is meant by 'library function', we must refer to the computer again.

7. Functions and the Computer.

Associated with a digital computer is a library of subroutines for computing various mathematical functions. In ADES, it is assumed that the computer library contains at least the floating-point arithmetic operations, $+$, $-$, $\cdot$, $/$. In some machines, these subroutines are part of the hardware. In others, interpretive subroutines are used. In either case, we regard them as part of the library of functions, that is, as mathematical functions which can be called for without special formulation.

Each library function is designated by an f with a characterising subscript. (In ADES II, we have reserved subscripts 0 to 49 for library functions. As mentioned earlier, f_1 is the identity function, f_2 is $+$, f_3 is $-$, f_4 is $\cdot$, f_5 is $/$, and so on.) In the translation process, the Encoder recognizes the subscript and inserts the pertinent calling sequence for the library subroutine into the program. At the same time, it compiles the subroutine itself, that is, it arranges the transfer of the subroutine from external storage (e.g. tape) to a suitable location in high speed storage. (See description of Encoder.)

The main point here is that in formulating a problem, a library function can be called for simply by writing an f with the correct subscript. No special equations are needed to define the function.*1 On the other

*1. See Appendix I for exceptions.

hand, each auxiliary function must be defined by an auxiliary equation. The assignment of a subscript to an auxiliary function is arbitrary, provided that it does not conflict with the subscript of a library function or another auxiliary function in the same problem.

We assume that the ADES II computer library contains, besides the four arithmetic operations, the following elementary functions: absolute value, square root, direct and inverse circular and hyperbolic functions, exponential, logarithm, power function.

In the terminology of recursive function theory, the library functions are the given functions of our formal system. The formulas for b's, r's and auxiliary f's are derived from the library functions by applying the syntactical rules. These rules are such that any derived function is primitive recursive relative to the set of library functions. Conversely, the library and rules of syntax should be such that any primitive recursive function can be derived. This implies that the syntax should include definition by recursion. Recursion will be introduced in section 11. In ADES II, however, recursive definitions are restricted to dependent variables, that is, a dependent index cannot be defined by recursion. Thus, not every primitive recursive function can be formulated. However, in practice, only a narrow class of functions is excluded thereby.

8. Quantification, Phasing;

The b-equations constitute the main part of an ADES formulation. They define the quantities to be computed, and closely resemble the conventional equations which a mathematician writes to describe a computational problem. But, in automatic digital computation, mathematical formulas are not sufficient. One must also specify the order in which quantities are to be computed, that is, a flow chart must be drawn up. This is frequently more difficult than writing the formulas.

In ADES, a large part of the flow chart is determined implicitly and in a natural way by the very structure of the b-equations. However, certain explicit directions must be written; e.g. for the quantification of independent indices, for branch equations, and for recursions. This section presents some of the rules for writing quantifications.

Def. 8.1. A quantifier is an expression of the form '$\forall$ Liu', where the symbol '$\forall$'* is the universal quantifier 'for all', i denotes an independent index, L denotes the lower bound of i and may be either an indexed independent variable or an index, and u denotes the upper bound of i and may be an indexed independent variable or an index.

Thus, '$\forall$ Liu' is to be read as 'for all integral values of i such that $L \leqslant i \leqslant u$.'

* In the actual ADES alphabet, $\forall$ is denoted by a library function symbol.

Rule 8.1. A quantifier is written to the left of a b-equation. Unless specifically forbidden by other rules, any b-equation may be preceded by one or more quantifiers. (The b-equation and the dependent variable which it defines are then also said to be quantified.) If more than one quantifier is written to the left of a b-equation, the order of quantification is considered to take place from left-to-right, i.e. if we have

$$\forall\, L_1 i_1 u_1 \quad \forall\, L_2 i_2 u_2 \quad y = \phi,$$

this implies that for all i_1, $L_1 \leq i_1 \leq u_1$, y is to be computed for all $i_2, L_2 \leq i_2 \leq u_2$. The quantification '$\forall\, L_1 i_1 u_1$' applies to the quantification '$\forall\, L_2 i_2 u_2$'. Hence, L_2 and u_2 can depend on i_1, whereas L_1 and u_1 must be independent of i_2.

It is sometimes necessary to quantify several b-equations by the same quantifier, or it may be necessary to specify the order in which certain independent quantifications are to take place. In many problems, it is necessary that certain b's be computed before others. These situations are provided for by the 'phase equation'.

Def. 8.2. A phase equation is a b-equation of the form,

$$y = f_o y_1 y_2 \cdots y_n,$$

where the y's denote dependent variables, n is an arbitrary positive integer, and f_o is a special library function.

The function, f_o, is to be read as, 'compute the following quantities'. Strictly speaking, f_o is not a function in the mathematical sense. However, in ADES, it is to be regarded as a function of arbitrary degree which operates on those b's between itself and the first comma to its right. (Therefore, the explicit and correct placement of this comma is essential, just as it is in all b-equations.)

The dependent variable denoted by y on the left side of a phase equation does not represent a computed mathematical quantity; but, this introduction of a dependent variable allows us to compound phase equations, for y can be used as an operand in another phase equation, or even in an ordinary b-equation. This makes it possible to specify the order of computation by the simple expedient of writing the dependent variables in the desired order within a term. This is amplified in Appendix II, and in examples.

Rule 8.2. To quantify several dependent variables $b_{j1}, \ldots, b_{jn}$, with the same quantifier, a phase equation is written as follows:

$$\forall\, Liu \quad b_{jo} = f_o b_{j1} b_{j2} \cdots b_{jn},$$

(This can be read as 'for all i, $L \leq i \leq u$, compute $b_{j1}, \ldots, b_{jn}$.')

Def. 8.3. A phase equation in which the left member is the special dependent variable, b_o, is called a master phase equation; i.e.

$$b_o = f_o b_{j1} b_{j2} \cdots b_{jn} , \quad .$$

Rule 8.3. There is precisely one master phase equation in each problem formulation. The dependent variable, b_o, is used only on the left side of the master phase equation and in no other equation.

The master phase equation is the first b-equation to be programmed. The left-to-right order of the $b_{j1}, \ldots, b_{jn}$ in the master phase equation determines the order in which the various phases of the computation will proceed. We shall postpone a precise detailed description of the ordering procedure to Appendix II, but many of the main ideas are suggested in the following examples, and a rule of thumb is given in example 8.4 below.

e.g. 8.1. It is required to compute $a_1(q_1) + a_2(q_1)$, and $a_1(q_1) - a_2(q_1)$ for all $q_1, 0 \leq q_1 \leq 25$. A formulation, neglecting input-output symbols, is as follows:

$$\begin{aligned} \forall 0 q_1 25 \quad & b_o = f_o b_2 b_1 , \\ & b_1 = + a_1 q_1 a_2 q_1 , \\ & b_2 = - a_1 q_1 a_2 q_1 , \quad . \end{aligned}$$

The master phase equation specifies that b_1 and b_2 are to be computed for all $q_1, 0 \leq q_1 \leq 25$. The order in which the b's on the right side of a b-equation are written, reading from left to right, determines the order in which they are to be computed. In all problems, the computation starts with the master phase equation. In example 8.1, the quantification information will be obtained first. Then the programs for b_2 and b_1 will be set up in that order. Note that, in this case, the order in which the b-equations is written is irrelevant. This is true for all b-equations except recursion equations and vector equations (see sections 10,11).

e.g. 8.2. Suppose it is required to compute both $a_1(q_1) + a_2(q_2)$ and $a_3(q_1,q_2) + a_4(q_1,q_2)$, for all $q_1, 0 \leq q_1 \leq 9$, and for all $q_2, 0 \leq q_2 \leq 8$. A formulation, neglecting input-output symbols, is as follows:

$$\begin{aligned} & b_1 = + a_1 q_1 a_2 q_2 , \\ & b_2 = + a_3 q_1 q_2 a_4 q_1 q_2 , \\ \forall 0 q_1 9 \; \forall 0 q_2 8 \quad & b_o = f_o b_1 b_2 , \quad . \end{aligned}$$

Starting with the master phase equation, the program for the quantification of q_1 will be set up first. The quantifier for q_2 will be set up next, and then the programs for b_1 and b_2 in that order. Again, the b-equations can be written in any order.

e.g. 8.3. It is required to compute $a_1(q_1) + a_2(q_1)$ for all $q_1, 0 \leq q_1 \leq 5$, and $a_3(q_1,q_2) + a_4(q_1,q_2)$ for all $q_1, 0 \leq q_1 \leq 5$, and all $q_2, 0 \leq q_2 \leq 9$. A formulation, neglecting input-output symbols, is as follows:

$$\begin{aligned} \forall\, 0q_1 5 \quad b_o &= f_o b_1 b_2, \\ b_1 &= +\, a_1 q_1 a_2 q_1, \\ \forall\, 0q_2 9 \quad b_2 &= +\, a_3 q_1 q_2 a_4 q_1 q_2, \ . \end{aligned}$$

The quantification of q_1 in the master phase equation will be set up first. Then b_1 will be programmed, since it precedes b_2 in the phase equation. Before the program for b_2 is composed, the quantification of q_2 is set up.

Remark: Each quantification corresponds to a 'loop' in the flow chart. In example 8.3, the loop for q_1 is traversed six times. Within the q_1 loop is the loop for q_2, which is traversed ten times for each traverse of the q_1 loop. Six values of b_1 are computed in the q_1 loop, whereas sixty values of b_2 are computed in the q_2 loop.

e.g. 8.4. It is required to compute $a_1(q_1) + a_2(q_1)$ and $\sin(a_1(q_1) + a_2(q_1))$ for all $q_1, 0 \leq q_1 \leq 5$. A formulation, neglecting input-output symbols, is as follows:

$$\begin{aligned} b_o &= f_o b_1, \\ \forall\, 0q_1 5 b_1 &= \sin b_2, \\ b_2 &= +a_1 q_1 a_2 q_1, \ . \end{aligned}$$

Here, the master phase equation serves only as a starting point. Since the b_1 equation is quantified, all dependent variables in that equation are also quantified by that same quantifier. Thus, the b_2-equation is implicitly quantified.

As a rule of thumb for specifying the order of computation, the formulator can use the following brief outline of the operation of the Encoder. The Encoder will start at the master phase equation, scan right for the first dependent variable, b_1 in example 8.4. It finds the b_1-equation, sets up the quantifier for q_1, then scans the right side for b's. If one is present, the Encoder finds its defining equation. It repeats this process until an equation is found which contains no b's on the right side. This equation is programmed first. Then the path of search is retraced, ignoring b's which have already been programmed. In this example, after

the quantification of q_1 is set up, b_2 will be programmed and then b_1. (Note that the quantification could have been written in the master phase equation.)

Rule 8.4. No quantifier can be written more than once in a formulation.

Thus, not only is it unnecessary to quantify b_2 in the preceding example, but it is an error, since this would require writing '$\forall 0 q_1 5$' twice.

9. Branching

We now introduce a logical operation into the language. This will permit the definition of a quantity that is to be computed by one of two or more alternative formulas, depending on one or more conditions. This situation corresponds to a branch point in a flow chart of a calculation.

Def. 9.1. A branch equation is an equation of the form,

$$y = P_4 \phi x_1 x_2, \ \psi, \ \theta, \ ,$$

where y denotes the dependent variable being defined, P_4 is a special punctuation symbol called a 'branch symbol'; ϕ denotes one of three library functions for the conditions $\leq$, $<$, and $=$; x_1 and x_2 are variables; ψ denotes the b-term which defines y if the condition $\phi x_1 x_2$ holds, and θ denotes the b-term which defines y if the condition does not hold.

Def. 9.2. The definition 6.4 of b-equation is hereby extended to include branch equations.

Def. 9.3. A branch r-equation is an equation of the form,

$$i = P_4 \phi x_1 x_2, \ \psi, \ \theta, \ ,$$

where i denotes the dependent index being defined; P_4 is the branch symbol; ϕ is as in def. 9.1; x_1 and x_2 are indices or independent variables; ψ denotes the index term which defines i when $\phi x_1 x_2$ holds, and θ denotes the index term which defines i otherwise.

Def. 9.4. The definition 6.2 of r-equation is hereby extended to include branch r-equations.

e.g. 9.1. The branch equation,

$$b_1 = P_4 f_{10} a_5 7., + a_1 a_2, \ -a_1 a_2,$$

defines b_1 as equal to $(a_1 + a_2)$ if $a_5 < 7$, and $(a_1 - a_2)$ if $a_5 \geq 7$; i.e. f_{10} denotes '$<$' and is a library function.

e.g. 9.2. $r_1 = P_4 f_{11} q_1 a_2, + q_2 1, -q_2 3,$

This branch equation defines r_1 to be equal to $(q_2 + 1)$ if $q_1 \leq a_2$, and equal to $(q_2 - 3)$ if $q_1 > a_2$; i.e. f_{11} is the library function for '$\leq$'.

Observe that multiple branches can be set up by using a succession of branch equations in which one of the two alternative terms is $f_1 x$, where x is a dependent variable (or index, as the case may be) which is defined by another branch equation.

e.g.9.3. $b_1 = P_4 f_{10} a_5 7, + a_1 a_2, \quad f_1 b_2,$

$b_2 = P_4 f_{10} a_5 9, \quad -a_1 a_2, \cdot a_1 a_2,$

These equations define $b_1 = (a_1 + a_2)$ for $a_5 < 7$, $b_1 = (a_1 - a_2)$ for $7 \leq a_5 < 9$, and $b_1 = a_1 \cdot a_2$ for $a_5 \geq 9$.

Remark. We could dispense with the branch equation as such. Suppose $\phi(x)$ is a function which is equal to 1 if $x \leq a$ and equal to 0 if $x > a$. The equation, $y = \phi f + \phi g$, is equivalent to a branch equation for y, with f and g as the two alternative terms. The function ϕ is a primitive recursive function, and could be incorporated into the library as one of the given functions of the system. Nevertheless, the branch equation is included in the language for practical reasons. Many mathematicians are accustomed to that terminology, and it leads to more efficient programs.

10. Vector Equations.

A vector quantity is an n-tuple of scalar quantities; i.e. the components of the vector. In ADES II, we shall not provide any special letter to designate a vector, since all arithmetic will be carried out with the components of vectors. In fact, if the components are denoted by different b's, no new syntax is required at all. However, it may be convenient to represent the components of a vector by an indexed b, such as $b_1 q_1$, where $q_1 = 0,1,\ldots,n$. In that case, several situations can arise, each requiring different phasing, quantification and output specifications.

If the components, $b_1 q_1$, are defined by a single formula, then an output specification may be required in the equation for b_1. This is determined by rules given in section 13.

If the components, $b_1 q_1$, are defined by different formulas involving different arguments, and if it is necessary to compute all components before the computation can proceed, then a 'vector equation' should be used.

Def. 10.1. A vector equation is a set of consecutive b-equations of the form,

$$\begin{aligned}(y &=_{09} \phi_0,\\ y &=_{09} \phi_1,\\ &\vdots \quad\ \vdots\\ y &=_{09} \phi_n,\ )\end{aligned}$$

where y denotes the dependent variable which represents the vector and ϕ_i denotes the b-term which defines the components y(i), $i = 0,1,\ldots,n$. Each component must then be called for by writing y with a suitable index.

e.g.10.1. Given the acceleration vector (cos t, sin t, t+1), compute the force vector, b_1, for a mass a_1 at time $t = a_0$. An ADES formulation follows.

$$\begin{aligned}&b_o = f_o b_2 b_1,\\ \forall 0 q_1 2\ &b_1 = \cdot a_1 b_2 q_1,\\ &(b_2 =_{09} \cos a_o,\\ &\ b_2 =_{09} \sin a_o,\\ &\ b_2 =_{09} +1\ a_o,\ ).\end{aligned}$$

Now, it frequently happens that (k + 1) scalar quantities, y(i), $0 \leq i \leq k$, are defined by k + 1 different functions, g_i, of the same arguments, $x_1,\ldots,x_n$. In such cases, it is customary for the mathematician to write,

$$y(i) = g_i(x_1,\ldots,x_n), \quad 0 \leq i \leq k,$$

and then define the functions g_i in separate equations. This is especially convenient when another vector, z(i), is defined by the same functions with different arguments; i.e.

$$z(i) = g_i(u_1,\ldots,u_n), \quad 0 \leq i \leq k.$$

This type of formulation requires the introduction of an indexed function symbol, and a means of defining it.

Def. 10.2. An indexed function is an expression of the form, gi, where g denotes an auxiliary function and i denotes an independent index.

e.g. $f_{51}q_1$; $f_{60}q_2$;

Def. 10.3. A vector auxiliary equation is a set of consecutive auxiliary equations of the form,

$$
\begin{array}{l}
(g =_{09} \phi_o, \\
\quad g =_{09} \phi_1, \\
\quad \vdots \qquad \vdots \\
\quad g =_{09} \phi_k,) \ ,
\end{array}
$$

where g denotes an auxiliary function, and the ϕ_i denotes free terms which define the functions denoted by the indexed function, gi, $0 \leq i \leq k$.

Rule 10.1. An indexed function, gi, can be used only in a b-equation. g must be the only function in that equation. A vector auxiliary equation must be written to define gi. The index, i, must be quantified somewhere in the formulation. If y denotes the dependent variable being defined by the indexed function, we write the b-equation,

$$y = gi\ x_1 \cdots x_n \ ,$$

where the x's denote the arguments of gi. If it is necessary to refer to several values of y in the formulation, one writes y with a suitable index. In that case, the equation for y must contain a storage symbol as explained in section 13.

As a simple example, consider the problem,

$$
\begin{aligned}
y_i &= g_i(x_1(i), x_2(i)), \quad 0 \leq i \leq 1 \\
z_j &= g_j(x_3(j), x_4(j)), \quad 0 \leq j \leq 1, \\
g_o &= \sin(c_1 + c_2), \\
g_1 &= \cos(c_1 + c_2), \ .
\end{aligned}
$$

Letting $b_1 = y$, $b_2 = z$, $q_1 = i$, $q_2 = j$, $f_{60} = g$, $a_1q_1 = x_1(i)$, $a_2q_1 = x_2(i)$, $a_3q_2 = x_3(j)$, $a_4q_2 = x_4(j)$, we obtain the ADES formulation,

$$
\begin{array}{l}
(f_{60} =_{09} \sin + c_1c_2, \\
\quad f_{60} =_{09} \cos + c_1c_2,) \\
\quad b_o = f_o b_1 b_2, \\
\forall 0 q_1 1 b_1 = f_{60} q_1 a_1 q_1 a_2 q_1, \\
\forall 0 q_2 1 b_2 = f_{60} q_2 a_3 q_2 a_4 q_2,
\end{array}
$$

11. Recursion

If the ADES language is to allow for the formulation of primitive recursive functions, the syntax must include rules for definition by recursion. This is probably the most difficult part of the syntax.

In ADES II, we provide for the definition of dependent variables by several different kinds of recursion equations. The various types of recursion equations are distinguished by means of identifying subscripts attached to the equal signs. With the exception of a vector equation, the equal sign in any non-recursive equation has the subscript 00, which we have agreed to omit throughout this report. In recursion equations, a two-digit subscript, $n_1 n_2$, is written explicitly with each equal sign. The following table lists the value of n_1 and n_2, and the names of the corresponding recursions.

Subscript		Type of Recursion
n_1	n_2	
1	1	Simple scalar preceding-values
2	1	Simple scalar course-of-values
3	1	Simple vector preceding-values
4	1	Simple vector course-of-values
5	1	Simple vector one-row
1	2	Double preceding-row
2	2	Double two-preceding-rows
3	2	Double one-row
4	2	Double course-of-values
0	3	Triple special

The names are intended to describe both the structure of the mathematical formulation and the kind of store instructions in the program produced by the Encoder.

The second digit, n_2, indicates the number of independent indices involved in the recursion.

e.g. 11.1. For example, the Newton iteration algorithm for the square root of a_1 is a recursion on one index. It defines a scalar function, and the new value in an iteration depends only on the preceding value. Hence, it is a simple scalar preceding-values recursion; its subscript is 11. In ordinary mathematical notation, it might be written as,

$$x(i+1) = (1/2)\left[\, x(i) + a_1 / x(i) \right] ,$$
$$x(0) = 3,$$

where the recursion index, i, goes from zero to an upper bound, u, which depends on the accuracy desired. We shall rewrite this later as an ADES formulation.

e.g. 11.2. The summation, $b_1 = \Sigma_{q_1=0}^{n-1} a_1(q_1)$, also can be written as a recursion with subscript 11, namely,

$$b_1(q_1 + 1) = a_1(q_1) + b_1(q_1), \quad 0 \leq q_1 \leq n - 1,$$
$$b_1(0) = 0,$$

e.g. 11.3. As an example of a double recursion with subscript 12, we take the recursion formula for the binomial coefficients. Denoting the coefficients by $b_1(i,j)$, we write in conventional notation,

$$b_1(i + 1, j + 1) = b_1(i, j + 1) + b_1(i,j),$$
$$b_1(i + 1, 0) = 1, \text{ for all } i \leq -1,$$
$$b_1(0, j + 1) = 0, \text{ for all } j \geq 0,$$

Def. 11.1. A simple scalar recursion equation is an equation of the form,

$$\forall \text{ Liu } x =_{n_1} \phi, \quad \psi_0, \ldots, \psi_m, \quad ,$$

where x denotes the dependent variable being defined, i denotes the recursion (independent) index, ϕ denotes the b-term which defines $x(i + 1)$ as a function of $x(i)$, $x(i - 1)$, ..., $x(L)$, and other variables, and $\psi_0, \ldots, \psi_m$, are the b-terms which define the initial values $x(L)$, $x(L + 1), \ldots, x(m)$, respectively. $\forall$ Liu is the quantifier for i. It is the only quantifier permitted in the equation.

It is understood that the x on the left side of the equal sign denotes $x(i + 1)$. If for each value of i, the term, ϕ, involves only the $m + 1$ values of x preceding $x(i + 1)$, i.e. only $x(i)$, $x(i - 1), \ldots, x(i - m)$, then $n_1 = 1$, and we call this a preceding-values recursion. If ϕ involves any other set of values of x(e.g. all the values prior to $x(i + 1)$), we call this a course-of-values recursion and write $n_1 = 2$.

Remark: In a course-of-values recursion, the Encoder will reserve a total of N(i) computer storages for the values of x, where N(i) is the maximum number of values which the index i assumes, whereas in a preceding-values recursion, it allots only $m + 2$ storages. Thus, although $n_1 = 2$ will always yield a correct program, $n_1 = 1$ should be used whenever possible since it conserves computer storage.

e.g. 11.4. The Newton iteration algorithm of e.g. 11.1 for $b_1 = \sqrt{a_1}$ can be formulated in ADES as follows (neglecting Input-Output).

$$b_o = f_o b_1,$$
$$\forall 0 q_1 6\ b_1 = {}_{11}/ + b_1 q_1 / a_1 b_1 q_1 2., f_1 3, .$$

Here, six iterations are specified. Later, we shall introduce a device to permit the number of iterations to depend on the accuracy desired.

Def. 11.1a. The class of b-equations is hereby extended to include simple scalar recursion equations.

A system of simultaneous recursion equations on one index will be called a 'simple vector recursion'.

e.g. 11.5. Suppose the quantities x and y are defined by the system of equations,

$$\left.\begin{aligned} x(i+1) &= x(i) + y(i), \\ y(i+1) &= x(i) / y(i), \end{aligned}\right\} \quad \text{all } 0 \leq i \leq 6,$$
$$x(0) = a_1,$$
$$y(0) = a_2, .$$

The vector (x,y) is thereby defined by a 'simple vector recursion'. The formal definition is as follows.

Def. 11.2. A simple vector recursion is a system of equations of the form,

$$\forall \text{ Liu } (y_0 = {}_{n_1 1}\ \phi_0, \psi_{00}, \psi_{01}, \ldots, \psi_{0m},$$
$$y_1 = {}_{n_1 1}\ \phi_1, \psi_{10}, \psi_{11}, \ldots, \psi_{1m},$$
$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$
$$y_n = {}_{n_1 1}\ \phi_n, \psi_{n0}, \psi_{n1}, \ldots, \psi_{nm},) ,$$

where $y_0, y_1, \ldots, y_n$ denote dependent variables representing the components of the vector. There are two possible cases. Either each y denotes a distinct dependent variable, or all the y's denote the same dependent variable. In the former case, the ith value of the jth component is denoted by $y_j i$. In the latter case, it is denoted by yij, where $0 \leq j \leq n$, and is defined by the jth equation.

It is understood that $y_0, \ldots, y_n$ on the left side of the equations is an abbreviation for the values at the new point, i + 1. Thus, ϕ_j defines

$y_j(i+1)$ in terms of the preceding values $y_0(i)$, ..., $y_n(i)$, $y_0(i-1)$, ..., $y_n(i-1)$, ..., $y_0(L)$, ..., $y_n(L)$, and the new values $y_k(i+1)$, where $k < j$. ψ_{j0}, ..., ψ_{jm} denote b-terms which define the initial values $y_j(L)$, ..., $y_j(L+m)$, respectively. As in a simple scalar recursion, '$\forall$ Liu' is the quantifier for the recursion index, i.

If each ϕ_j involves only the $m+1$ preceding values of y_0, ..., y_n, and the new values of y_k, $k < j$, the subscript $n_1 = 3$ in all equations. As a special case, if each ϕ_j involves only $y_k(i+1)$ for $k < j$, and $y_k(i)$ for $k \geq j$, the subscript $n_1 = 5$. This is the vector 'one-row' recursion. In all other cases, $n_1 = 4$; i.e. they are treated as course-of-values vector recursions.

No quantifiers are permitted within the parentheses.

In calling for a vector recursion, one refers to the desired component. Reference to any component causes the entire recursion to be computed. The components are computed in the order in which they are written.

e.g. 11.6. We rewrite example 11.5 as an ADES formulation.

$$\begin{aligned} & b_o = f_o b_1 , \\ \forall 0 q_1 6 (& b_1 = 31 + b_1 q_1 b_2 q_1 , \quad f_1 a_1 , \\ & b_2 = 31 \,/\, b_1 q_1 b_2 q_1 , \quad f_1 a_2 ,) . \end{aligned}$$

In the above formulation, the vector is denoted by (b_1, b_2).

The problem can also be formulated with the notation $(b_1(0), b_1(1))$ for the vector. Thus,

$$\begin{aligned} & r_1 = f_1 0 ., \quad r_2 = f_1 1 ., \\ & b_o = f_o b_1 , \\ \forall 0 q_1 6 (& b_1 = 31 + b_1 q_1 r_1 \quad b_1 q_1 r_2 , \quad f_1 a_1 , \\ & b_1 = 31 \,/\, b_1 q_1 r_1 \quad b_1 q_1 r_2 , \quad f_1 a_2 ,) . \end{aligned}$$

In this second case, the components involve two indices in the recursion formulas. The final results, however, when referred to in another part of the problem, are denoted by $b_1 r_1$ and $b_1 r_2$.

If the components of the vector in a recursion are denoted by $b_1(j)$, $0 \leq j \leq n$, and if the same formula with different operands defines all components, then the simple vector recursion can be regarded as a double recursion. Thus, using i as the recursion index, $b_1(i+1, j+1)$ would be defined in terms of the preceding values of b_1. The final components of

the vector would then appear as the final line of the double recursion, and would have to be referred to as such. This will be further explained in the ensuing paragraphs on double recursion.

As shown in example 11.3, a double recursion involves two indices, i and j. We shall speak of the (i,j) 'grid' and refer to 'points' in this grid in an obvious way. A double recursion defines a quantity, b_1 say, at the grid point $(i+1, j+1)$ in terms of its values at grid points which 'precede' $(i+1, j+1)$. Following Péter [3], we say that the grid point (α_1, β_1) precedes (α_2, β_2) if either $\alpha_1 < \alpha_2$, or $\alpha_1 = \alpha_2$ and $\beta_1 < \beta_2$. Geometrically speaking, grid point P_1 precedes P_2 if either P_1 is in a lower row than P_2 or if P_1 is in the same row and in a column to the left of P_2. i and j will be called the row and column index, respectively. The order of computation in a double recursion will be along a row from column to column to the last point in the row, then up to the first point in the next row and so on.

Def. 11.3. A double recursion is a system of b-equations of the form,

$$
\begin{aligned}
\forall\, L_1 i_1 u_1 (\, & y_1 = {}_{n_1}2\ \phi_1, \\
& y_2 = {}_{n_1}2\ \phi_2, \\
& \vdots \qquad\quad \vdots \\
& y_{k-1} = {}_{n_1}2\ \phi_{k-1}, \\
\forall\, L_2 i_2 u_2 [\, & y_k r_{k_1} = {}_{n_1}2\ \phi_k, \\
& \vdots \qquad\quad \vdots \\
& y_s r_{k_s} = {}_{n_1}2\ \phi_s,] \\
& y_{s+1} = {}_{n_1}2\ \phi_{s+1}, \\
& \vdots \qquad\quad \vdots \\
& y_m = {}_{n_1}2\ \phi_m, \ ) \ .
\end{aligned}
$$

i_1 and i_2 denote the two recursion (independent) indices. i_1 is the row index. i_2 is the column index. (The column index, i_2, will run from its lower bound L_2 to its upper bound u_2 for each value of the row index, i_1.) $y_k, y_{k+1}, \ldots, y_s$ denote dependent variables which are to be computed for all i_1 and for all i_2. The equations for these variables are enclosed in

the square brackets. $y_1, \ldots, y_{k-1}$ and $y_{s+1}, \ldots, y_m$ denote b's which are to be computed only for all i_1. The equations for these b's are within the parentheses, but outside the brackets.

For $k \leq j \leq s$, in the equation, $y_j = {}_{n_1 2}\phi_j$, ϕ_j denotes the complete recursion formula for computing y_j at the grid point $(i_1 + 1, i_2 + 1)$. Unlike simple recursion, the equation for y_j will usually be a branch equation in which the alternative terms define the initial values $y_j(0, i_2 + 1)$ and $y_j(i_1 + 1, 0)$, and the general value $y_j(i_1 + 1, i_2 + 1)$, . Because of the choice of the indices $i_1 + 1$ and $i_2 + 1$ for the 'new' grid point, the lower and upper bounds in the quantifiers of i_1 and i_2 must be one less than the actual bounds of i_1 and i_2; e.g. if $0 \leq i_1 \leq a_1 + 1$, we must write '$\forall -1 i_1 a_1$,' to insure that $y_j(i_1 + 1, 0)$ starts with $y_j(0,0)$.

Of course, for $j < k$ or $j > s$, there is only one pertinent index, namely i_1. Hence, ϕ_j defines the initial value $y_j(0)$ and the value $y_j(i_1 + 1)$, again by using a branch equation.

Note: No quantifiers other than those for i_1 and i_2 are permitted within the parentheses.

Def. 11.3 is continued in the following rules, 11.1 to 11.3.

Rule 11.1. The equations in a double recursion will be computed in the order in which they are written; i.e. y_j is computed after y_{j-1} and and before y_{j+1}. This means that ϕ_j can contain operands of the following types:

(1) $y_m(\alpha,\beta)$, where $m < j$, and $k \leq m \leq s$; $\alpha \leq i_1 + 1, \beta \leq i_2 + 1$.

(2) $y_m(\alpha,\beta)$, where $m \geq j$ and $k \leq m \leq s$; and (α,β) precedes $(i_1 + 1, i_2 + 1)$, $i_1 \geq 0$, $i_2 \geq 0$.

(3) $y_m(\alpha)$, where $m < j$, and $m < k$ or $m > s$; $\alpha \leq i_1 + 1$

(4) $y_m(\alpha)$, where $m \geq j$ and $m < k$ or $m > s$, $\alpha \leq i_1$, $(i_1 \geq 0)$.

Rule 11.2. In a double recursion, the digit, n_1, in the subscript of the equal sign of the equation for y_j is determined by the row indices that go with the occurrences of y_j in the ϕ's. Each equation may have a different n_1 subscript.

(1) If for all $m < j$, ϕ_m involves only $y_j(i_1,\beta)$, where β is subject to rule 11.1, and if for all $m > j$, ϕ_m involves only $y_j(i_1 + 1,\beta)$, where β is subject to rule 11.1 and if ϕ_j involves only $y_j(i_1 + 1,\beta)$, where $\beta < i_2 + 1$, or $y_j(i_1,\beta)$, where $\beta \geq i_2 + 1$, then $n_1 = 3$ in the equation for y_j.

(2) If only i_1 and $i_1 + 1$ appear as row indices of y_j, then we write $n_1 = 1$ if we cannot write $n_1 = 3$.

(3) If only $i_1 - 1$, i_1 and $i_1 + 1$ appear as row indices of y_j, then we write $n_1 = 2$.

(4) In all the remaining cases, we write $n_1 = 4$.

The above rule for determining n_1 will assure an economy of storage in the computer. However, if economy is not an important factor, the formulator can simply write $n_1 = 4$ in all equations. Each equation will then be treated as a course-of-values recursion.

Rule 11.3. If $n_1 = 4$ in an equation within the brackets, the formulator must write a dependent index immediately to the right of the dependent variable on the left side of the equation. The precise nature of this index will be explained in Rule 13.6 in section 13, on Input-Output. Its purpose is to provide store instructions.

Note: Rule 11.1 regarding the order of the b-equations within the parentheses of a double recursion must be strictly observed. This rule prescribes necessary conditions on the order, but it does not determine the order uniquely in all cases. Hence, some of the ordering is at the formulator's option.

Rule 11.4. If one of the equations within the parentheses involves as an operand a dependent variable, b_j, which is not part of the simultaneous double recursion, then the equation for b_j should be written outside the parentheses.

As remarked earlier, each quantification corresponds to a loop in the flow chart. Thus, in a double recursion, the quantification of i_1 sets up a recursive loop which is repeated for $-1 \leq i_1 \leq u_1$. Within the i_1-loop is the recursive loop for i_2. Recursive loops differ from ordinary loops in that the quantities computed are stored in a recursive manner. Thus, recursive loops allow several interdependent variables to be computed simultaneously; i.e. if b_1 is a function of b_2 and b_2 is a function of b_1, this implies that a recursion is taking place, since ADES does not permit implicit functions.

To provide greater flexibility in the ordering of recursive loops, the double recursion definition is extended as follows.

Def. 11.4. A general double recursion is a double recursion in which there is more than one column index. Each column index is quantified separately, and the b-equations to which it applies are enclosed in square brackets.

For example, a general double recursion in which there are two column indices would have the following format:

$$
\begin{aligned}
\forall\text{-}li_1u_1\ (\ & y_1 = {}_{n_1}2\ \phi_1, \\
& \vdots \qquad\qquad \vdots \\
& y_{k-1} = {}_{n_1}2\ \phi_{k-1}, \\
\forall\text{-}li_2u_2\ [\ & y_k r_{k_1} = {}_{n_1}2\ \phi_k, \\
& \vdots \qquad\qquad \vdots \\
& y_s r_{k_s} = {}_{n_1}2\ \phi_s,\ ] \\
& y_{s+1} = {}_{n_1}2\ \phi_{s+1}, \\
& \vdots \qquad\qquad \vdots \\
& y_m = {}_{n_1}2\ \phi_m, \\
\forall\text{-}li_3u_3\ [\ & y_{m+1} r_{j_1} = {}_{n_1}2\ \phi_{m+1}, \\
& \vdots \qquad\qquad \vdots \\
& y_h r_{j_h} = {}_{n_1}2\ \phi_h,\ ] \\
& y_{h+1} = {}_{n_1}2\ \phi_{h+1}, \\
& \vdots \qquad\qquad \vdots \\
& y_t = {}_{n_1}2\ \phi_t,\ ).
\end{aligned}
$$

Rules 11.1, 11.2, and 11.3 on order and indices apply, with obvious modifications to account for i_3.

e.g. 11.7. We reformulate example 11.3 in ADES (omitting input-output specifications). The binomial coefficients are denoted by b_1, the row index by q_1, and the column index by q_2.

$$\forall\text{-}1q_1a_3(\forall\text{-}1q_2a_3\,[\,r_1 = +q_2 1,\quad b_1 = {}_{12}P_4 f_{10}a_2 0,\ f_1 1.,\ f_1 b_2,\quad b_2 = P_4 f_{10}a_1 0,\ f_1 0,\ +\ b_1 q_1 r_1 b_1 q_1 q_2,\quad b_o = f_o b_1,\,])\ .$$

Remarks: There is only one recursion equation, namely, that which defines b_1. Since there are two initial conditions, a triple branch is required. The first branch condition, $f_{10}a_2 0$, is '$a_2 < 0$', where $a_2 = q_2$. If $a_2 < 0$, i.e. if $q_2 = -1$, the equation defines b_1 to be equal to 1. If $a_2 \geq 0$, $b_1 = b_2$, where b_2 is defined by a branch equation outside the parentheses. The branch condition for b_2 is '$a_1 < 0$', where $a_1 = q_1$. If $a_1 < 0$, i.e. $q_1 = -1$, then $b_1 = b_2 = 0$. If $a_1 \geq 0$, we have the main recursion formula. Part 2 of Rule 11.2 applies, and we write 12 as the subscript of the recursion equal sign. Therefore, this will be treated by the Encoder as a preceding-row recursion. $(a_3 + 2)$ storages will be reserved for the 'preceding' row corresponding to the index q_1, and $(a_3 + 2)$ storages for the 'current' row corresponding to $q_1 + 1$. Note that in applying Rule 11.2, we must check through all b-equations referred to by the recursion equation for b_1.

e.g. 11.8. In the Choleski method for solving systems of linear algebraic equations, a square symmetric matrix $a_3(i,j)$, of order $(n+1)$ is given, and a triangular matrix $b_1(i,j)$ is computed by the formulas,

$$i > j,\ b_1(i,j) = \left[\, a_3(i,j) - \Sigma_{k=0}^{j-1} b_1(i,k)b_1(j,k)\right] \Big/ b_1(j,j),$$

$$i = j \quad b_1(i,i) = \sqrt{\left[\, a_3(i,i) - \Sigma_{k=0}^{i-1} b_1(i,k)b_1(i,k)\right]}\ ,\ .$$

Denoting i by q_1, j by q_2, (n-1) by a_4, k by q_3, the expression in brackets by b_2, and the summation by b_3, we can formulate this in ADES as follows (again without input-output):

$$\forall\text{-}1q_1a_4(\forall\text{-}1q_2q_1\,[\,r_1 = +q_1 1,\quad r_2 = +q_2 1,\quad b_1 r_3 = {}_{42}P_4 f_{10}a_2 a_1, / b_2 b_1 r_2 r_2, \sqrt{}\, b_2,\quad b_2 = -a_3 r_1 r_2 b_3,\quad \forall\, 0q_3 q_2 b_3 = {}_{11}+\cdot b_1 r_2 q_3 b_1 r_1 q_3 b_3 q_3,\ f_1 0.,\quad b_o = f_o b_1,\,])$$

Remarks: In the input specifications, a_1 and a_2 are identified as q_1 and q_2 respectively, i.e. they are integer-valued variables (Def.6.5).

This artifice must be used since a b-equation cannot contain indices as operands. Since all values of b_1 are involved in the recursion, it is considered to be a course-of-values recursion. Hence, the subscript is 42. The symbol r_3 is a storage (i.e. output) specification. It will be discussed in section 13, Rule 13.6.

In the preceding section on vectors, it was said that a simple vector recursion in which the components of the vector are defined by the same function can be formulated as a double recursion. The next example illustrates this. It is the well-known Gauss-Seidel method for solving a system of linear equations.

e.g. 11.9. The system of equations,

$\Sigma_{j=0}^{n} a_1(i,j)\, b_1(j) = a_2(i)$, $i = 0,1,\ldots,n$, is to be solved for the vector, $b_1(j)$, given the matrix $a_1(i,j)$ and the vector $a_2(i)$. Denoting the iteration index by k, the Gauss-Seidel formula, in conventional notation, can be written as,

$$b_1(k+1,\, j+1) = \frac{1}{a_1(j+1,\, j+1)} \left[a_2(j+1) - \Sigma_{q=j+2}^{n} a_1(j+1,q) b_1(k,q) - \Sigma_{q=0}^{j} a_1(j+1,q) b_1(k+1,q) \right].$$

This is a vector recursion on the index k. It is of the type in which all components of the vector are defined by a single formula. As mentioned in section 10, such a vector recursion can be treated as a double recursion. The Gauss-Seidel formula illustrates this. In ADES language, if we write q_1 for k, q_2 for j, q_3 for q in the first sum, which we denote by b_2, write q_4 for q in the second sum, which we denote by b_3, and a_3 for n, we get the formulation (using $a_4 + 2$ iterations):

$$b_0 = f_0 b_1,$$

$$r_1 = +1.q_2, \quad r_2 = +1q_1, \quad r_4 = +2.q_2, \quad r_3 = -a_3 1,$$

$$\forall\, -1 q_1 a_4\ (\forall\, -1\ q_2 r_3\ [b_1 = {}_{12} / -a_2 r_1 + b_2 b_3 a_1 r_1 r_1\ ,])$$

$$\forall\, r_4 q_3 a_3 b_2 = {}_{11} + b_2 q_3 \cdot a_1 r_1 q_3 b_1 q_1 q_3,\ f_1 0\ ,$$

$$\forall\, 0 q_4 q_2\ b_3 = {}_{11} + b_3 q_4 \cdot a_1 r_1 q_4 b_1 r_2 q_4,\ f_1 0,\ .$$

Next, we consider the following problem. Suppose that the elements of a two-dimensional matrix, $b_1(q_2,q_3)$, are to be computed by a simple recursion on q_1. In some cases, one could set up the quantification to take place in the order q_1, q_2, q_3, and a simple scalar recursion formulation would suffice. Thus, for each element, $b_1(q_2,q_3)$, a simple recursion

would be performed. Even if this procedure were possible mathematically, it might not yield an efficient program since some quantities might be recomputed many times. In cases where the recursion is simultaneous, that is, $b_1(q_1+1,q_2,q_3)$ depends on some $b_1(q_1,\alpha,\beta)$ where $(\alpha,\beta) \neq (q_2,q_3)$, the simple recursion procedure is no longer valid. The quantification must now take place in the order q_2,q_3,q_1 or q_3,q_2,q_1, that is, for each q_1, we compute b_1 for all q_3 and q_2. This is, in effect, a triple recursion.

In ADES II, a special type of triple recursion is provided. It will cope with the above matrix recursion, and with any triple recursion which satisfies the special requirements explained below.

Def. 11.5. Let the symbol 'GDR' denote a set of b-equations written in the format of a general double recursion, except that all the equal signs are written with subscript 03. A special triple recursion is a set of b-equations of the form,

$$
\begin{aligned}
\forall\, L_1 i_1 u_1 \{\; & y_1 =_{03} \phi_1, \\
& \vdots \qquad\quad \vdots \\
& y_k =_{03} \phi_k, \\
& (\text{ GDR })_1 \\
& y_{k+1} =_{03} \phi_{k+1}, \\
& \vdots \qquad\quad \vdots \\
& y_s =_{03} \phi_s, \\
& (\text{ GDR })_2 \\
& \vdots \\
& y_m =_{03} \phi_m, \;\}\ .
\end{aligned}
$$

Thus, a triple recursion consists of b-equations interspersed with GDR's, all quantified by the i_1 quantifier. In each GDR there is one row quantification and any number of column quantifications (possibly none).

We shall not attempt to give formal rules for the writing of indices in a triple recursion. Instead, we shall explain briefly the order of computation and the manner of storage in the computer. This will indicate how to index the dependent variables.

Let i_2 denote a row index which precedes the parenthesis in a GDR. Let i_3 be a column index which precedes a bracket in that GDR. If y_j is defined by an equation within the i_3 brackets, the recursion for y_j proceeds along the grid points of the (i_1+1)-plane, running along the (i_2+1)-row,

as the column index, i_3, runs from its lower bound to its upper bound. The storage of y_j values is limited to at most a two-dimensional array. All storage specification is accomplished by writing a suitable dependent index to the right of y_j in the y_j -equation. This is explained in section 13. If y_j is stored as a two-dimensional array, this implies that when the index i_1 increases and a new value $y_j(i_1 + 1, \alpha,\beta)$ is computed, it immediately replaces $y_j(i_1,\alpha,\beta)$ in storage. With this convention, only the row and column indices need be specified when y_j is used as an operand. y_j can also be stored as a one-dimensional array if the problem permits. In that case, only one row of y_j is stored in the computer at any juncture. When i_2 increases, the new value $y_j(i_2 + 1,\beta)$ replaces $y_j(i_2,\beta)$ immediately. Hence, if y_j is used as an operand, only one index need be written in this instance.

If the y_j equation occurs within the i_2 parentheses of a GDR but not within any of the brackets, the recursion for y_j will run from row to row in the $(i_1 + 1)$ -plane as i_2 varies. Here, we have the possibility of storing y_j as two-dimensional (i_1,i_2) array, or as a one-dimensional i_2 array. In the latter case, when $y_j(i_1 + 1,\alpha)$ is computed, it immediately replaces $y_j(i_1,\alpha)$ in storage. Thus, only the array $y_j(i_2)$ is in storage at any time. Part of this array may belong to the $(i_1 + 1)$ -plane and the rest to the previous plane. When y_j is used as an operand, only one index is required in this case.

Finally, if the y_j -equation occurs outside all GDR's, it is computed only for all i_1. Here, the new value of y_j does not immediately replace the previous value. This replacement is done after all y's have been computed in the i_1 loop. Thus, after y_j is evaluated in the $(i_1 + 1)$ -plane, there are two values of y_j in storage, namely, $y_j(i_1 + 1)$ and $y_j(i_1)$. To use either as an operand in a formula one simply writes the proper index, keeping in mind that the y's are computed in the order in which their equations are written. If all values, $y_j(i_1)$, are to be stored, an index must be written as explained in section 13.

No quantifications are written within the braces other than those in the GDR's. However, if it necessary to introduce other quantifiers, one can write a phase equation, $y_j = {}_{03}f_1 y_i$, at the appropriate place within the braces, and then quantify the y_i -equation, which is written outside the braces. Thus, for example, y_i might itself be defined by a triple recursion. The storage for y_i is specified in the y_i -equation.

12. Minimization.

In section 3, brief mention was made of the minimization operator which is introduced in the theory of recursive functions to eliminate multiple recursions. We incorporate a form of this operator into ADES II as a library function. However, we do not propose to eliminate multiple recursions. Rather, minimization will be used frequently within recursion formulations to define the upper bound of a recursion index. It will also be used in 'table look-up' operations.

Def. 12.1. A minimization equation is a special equation of the type

$$j = f_{\mu} ixy, \quad ,$$

where j is a dependent index, i is an independent index, x is an indexed independent variable or numerical constant, y is a dependent variable, and f_{μ} denotes a special library function known as the 'minimization operator'. The minimization equation can be read as 'j is the minimum value of $i, i \leq x$, such that $y \leq 0$' .

Def. 12.2. The class of r-equations is hereby extended to include minimization equations.

e.g. 12.1. In example 11.4, the Newton iteration algorithm for the square root was formulated in ADES language for six iterations. We now impose the requirement that the error in the square root be less than 10^{-6}, if possible, but that at most six iterations be used. The formulation, with a minimization equation, becomes

$$\begin{aligned} r_1 &= f_{\mu} q_1 6.b_2, \\ r_2 &= + q_1 1, \\ b_o &= f_o b_1, \\ \forall\, 0 q_1 r_1 b_1 &= {}_{11}/ + b_1 q_1 / a_1 b_1 q_1 2., \quad f_1 3, \\ b_2 &= -f_{abs.} -b_1 r_2 b_1 q_1 10^{-6}, \quad , \end{aligned}$$

where we have written $f_{abs.}$ to denote the library function for the absolute value.

Remark: The above use of the minimization operator in scalar recursions is so common that some special abbreviation would be feasible. We might simple write,

$$r_1 = f_{\mu} 6. \times 10^{0} \quad 1. \times 10^{-6},$$

it being understood that the first number, 6×10^{0}, is the upper bound of the recursion index, while the second is the tolerance for the absolute difference between successive values of the quantity being computed. In vector recursions, the tolerance might be applied to the sum of squares of differences of successive components.

The 'table look-up' aspect of minimization is illustrated by the next example.

e.g. 12.2. Let fifty values, $a_3(q_1)$, of a function be tabulated against the corresponding values of its independent variable, $a_1(q_1)$. It is required to evaluate the function at some point, a_2, by linear interpolation. In conventional language, this value is given by,

$$b_1 = a_3(r_1) + \frac{a_3(r_1 - 1) - a_3(r_1)}{a_1(r_1 - 1) - a_1(r_1)} \left[a_2 - a_1(r_1) \right],$$

where r_1 is the minimum value of q_1 such that $a_1(q_1) - a_2 \leq 0$. In ADES, the formulation (except for Input-Output specifications), is as follows:

$$r_1 = f_\mu q_1 50.b_2,$$
$$r_2 = -1 r_1,$$
$$b_o = f_o b_1,$$
$$b_1 = +a_3 r_1 \cdot -a_2 a_1 r_1 / -a_3 r_2 a_3 r_1 - a_1 r_2 a_1 r_1,$$
$$b_2 = -a_1 q_1 a_2,$$

Other examples will be found in Appendix III.

13. Input and Output Formulation.

The mathematical and logical part of the formulation of a problem in ADES II, as described in the preceding sections, consists of r-equations, b-equations, and auxiliary equations. To complete the description of the language, syntactical rules will now be given for the specification of the input of data, and the output of results.

Let $a_0, a_1, a_2, \ldots, a_n$ be the independent variables which occur in the r-equations and b-equations of a problem. The a_k represents the different types of input data. Let N_k be the maximum number of numerical quantities to be supplied in the computer as data for a_k. In problems which are to be repeated for j different cases, the actual number, $\overline{N}_{kj}$, of data for a_k may vary from case to case. Thus, $N_k = \max_j \left\{ \overline{N}_{kj} \right\}$.

Def. 13.1. In what follows, we shall say that a variable, a_k (or b_k), 'is of degree two in a formulation' if it occurs at least once as part of an indexed variable of degree two; e.g. $a_k q_1 q_2$ (or $b_k q_1 q_2$). If a variable a_k (or b_k) is not of degree two, and if it occurs at least once as part of an indexed variable of degree one, then we say that a_k (or b_k) 'is of degree one'. Otherwise, a_k (or b_k) 'is of degree zero'.

Remark: If an independent variable is of degree zero, one, or two, then it always occurs with zero, one, or two indices, respectively. The phrase 'at least once' really applies to dependent variables, since they can occur with a different number of indices at different places in the same problem.

For an a_k of degree zero, $N_k = 1$; i.e. one numerical datum will be loaded into the computer for a_k. (Exception: If a_k is an integer-valued variable (Def.6.5), no data is supplied.) If a_k is of degree one, then $N_k > 1$; a 'column' of numerical data, to be substituted for $a_k(j)$, $0 \leq j \leq u$, will be loaded into the computer.

If a_k is of degree two, a matrix of data, to be substituted for $a_k(i,j)$, will be loaded into the computer.

The loading of data into the computer's internal storage in ADES II is under control of a 'Read' program composed by the Encoder. The Read program is executed by the computer just after the entire program has been loaded into internal storage. The Read program transfers data from external storage (e.g. tape, cards) to predetermined internal storages. In ADES II, the Encoder composes the Read program on the assumption that the data in external storage is arranged according to the following conventions.

Computer Convention 1. All the data for each a_k is in a block of consecutive external storages. The beginning of the block must be identified as belonging to the variable a_k. If tape is used, the end of the block should also be marked. If punched cards are used, all data on a card should be for the same a_k, and should be in the order dictated by conventions 2, 3, and 4 below. Further, all the data for an a_k should be on consecutive cards, in order, and suitably identified. Just what constitutes suitable identification will depend on the particular computer.

Remark: It is not difficult to see how these conventions for the computer can be relaxed to allow for variations in the arrangement of input data. The scheme suggested here somewhat simplifies the Encoder and the Read program. For those problems in which the amount of data is comparatively small, the conventions impose little or no inconvenience.

Computer Convention 2. If a_k is of degree zero, only one datum is supplied, i.e. this variable requires a block of external storage of length 1. The block should be suitably identified at beginning and end. The blocks for all a_k of degree zero should themselves be consecutive.

Computer Convention 3. Following the data for all the variables of degree zero is the block of data for the first variable of degree one. All the data for a variable a_k of degree one should be loaded in a block of consecutive storages in an order corresponding to $a_k(0)$, $a_k(1)$,..., $a_k(\overline{N}_k - 1)$. Following the block of data for the first variable of degree one is the block of data for the second variable of degree one, etc.

Computer Convention 4. Following the block of data for the last variable of degree one is the block of data for the first variable of degree two, a_k. This is actually a matrix of data to be substituted for $a_k(i,j)$, $0 \leq i \leq u_1$, $L_2 \leq j \leq u_2$. The linear ordering of the data in this matrix into a block of consecutive storages must correspond to $a_k(0,L_2)$, $a_k(0,L_2+1),\ldots, a_k(0,u_2)$, $a_k(1,L_2)$, $a_k(1,L_2+1),\ldots, a_k(1,u_2)$, $\ldots, a_k(u_1,L_2)$, $\ldots, a_k(u_1,u_2)$. In short, the matrix should be loaded into external storage by consecutive rows. This ordering can be described as a mapping of the two-dimensional (i,j) grid onto a one-dimensional r-grid extending from some point α to the point $\alpha + \bar{N}_k - 1$. For example, if $0 \leq i \leq 5$ and $0 \leq j \leq 5$, the square 6 x 6 grid is mapped into the linear grid by the mapping,

$$r = \alpha + (j + 6i), \quad .$$

Again, if $0 \leq i \leq 5$, $i \leq j \leq 5$, the triangular grid is mapped into the linear one by,

$$r = \alpha + (j + 21 - \frac{(6-i)(7-i)}{2}), \quad .$$

If the matrix structure is more complicated, the mapping may be defined by using a tabular function. The data for the other variables of degree two is loaded similarly, and in consecutive blocks of storage, each suitably marked.

The structure of the matrix of data for a variable, a_k, of degree two determines a mapping according to Convention 4. The formulator must write this mapping in the formulation as an index equation defining a dependent index. Such a dependent index will be called a 'storage index' of type A, since it describes how the data for a_k is to be arranged in storage. However, in writing the 'storage index equation', the formulator need make no reference to any storage address; or equivalently, the initial storage address, α, is always taken to be zero. The storage index is defined as a purely mathematical function of two independent indices: q_{98}, representing the row index, and q_{99}, the column index of a matrix. Thus, for the square matrix with row index $0 \leq q_{98} \leq 5$ and column index $0 \leq q_{99} \leq 5$, the storage index equation is,

$$r_1 = (q_{99} + 6q_{98}), \qquad \text{(common notation)}$$

$$r_1 = + q_{99} \bullet 6.q_{98}, \quad . \qquad \text{(ADES notation)}$$

For the triangular matrix with $0 \leq q_{98} \leq 5$, $q_{98} \leq q_{99} \leq 5$, the storage index is,

$$r_1 = q_{99} + (21 - \frac{(6 - q_{98})(7 - q_{98})}{2}), \text{(common)}$$

$$r_1 = +q_{99} - 21. /\bullet -6.q_{98} - 7.q_{98}2, \quad . \text{(ADES)}$$

For the triangular matrix, $0 \leq q_{98} \leq a_1$, $0 \leq q_{99} \leq q_{98}$, the storage index equation is,

$$r_1 = q_{99} + q_{98}(q_{98} + 1)/2, \quad \text{(common)}$$

$$r_1 = +q_{99}/\bullet q_{98} + q_{98}\,1.2, \quad \text{(ADES)}.$$

The formulator must also list the type A storage indices in a table, called the 'Computer Table', which is part of the formulation. The Computer Table will be defined precisely in Def. 13.4.

Since the data for most of the independent variables of degree two in a given problem will usually have the same matrix structure, the same storage index will apply to most of the a's. Rectangular and Triangular structures are the most common. Several rectangular arrays of different dimensions may occur in the same problem. In that case, it is more efficient to have a single function, f_r say, to denote the formula for the rectangular mapping. If f_r is a library function, the formulator need only write an equation of the type,

$$r_1 = f_r q_{98} q_{99} a_1, \quad ,$$

where q_{98} and q_{99} are the row and column indices, respectively, and $0 \leq q_{99} \leq a_1$. If f_r is not a library function, the auxiliary equation,

$$f_r = +c_2 \bullet c_1 c_3, \quad ,$$

must also be written, but this auxiliary equation will serve for all other rectangular storage index equations.

It should be evident that no particular row and column indices can be specified in a storage index equation for an independent variable of degree two. It is only necessary to distinguish the row index from the column index. With all the above considerations in mind, we adopt the following definition and rule for ADES II.

Def. 13.2. A storage index equation of type A is an r-equation of the form,

$$j = \phi q_{98} q_{99} x_3 \cdots x_n, \quad ,$$

where j is a dependent index (called a type A storage index), ϕ is a function symbol, q_{98} always denotes the row index, q_{99} the column index, and the x's are indexed independent variables. The function ϕ defines a mapping of the grid points (q_{98}, q_{99}) onto a linear grid. If ϕ is not a library function it must be defined by an auxiliary equation in which the free variables $c_1, c_2, \ldots, c_n$ are to be identified with $q_{98}, q_{99}, x_2, \ldots, x_n$, respectively. q_{98} and q_{99} are never quantified and can be used only in storage index equations of type A.

Rule 13.1. To each independent variable, a_k, of degree two in a formulation, there is assigned a storage index of type A. A storage index equation of type A must be written to define this storage index. The function ϕ in this equation depends on the structure of the matrix of data for a_k. ϕ must map the 'grid points' of the matrix into a linear sequence according to computer convention 4. The same storage index can be used for different variables if the matrix structures are identical. The type A storage indices must be listed in the 'Computer Table' (see Def. 13.4 below).

Storage index equations are also written to specify the storage structure of computed results. The rules for these equations are somewhat lengthier.

Def. 13.3. A storage index equation of type B is an r-equation of the form

$$j = f_1 i_1, \quad ,$$

or of the form

$$j = \phi i_1 i_2 x_3 \ldots x_n, \quad ,$$

where j is a dependent index (a storage index of type B), ϕ is a function, i_1 and i_2 are indices, and x_3, ..., x_n are indexed independent variables. If ϕ is an auxiliary function, the free variables $c_1, c_2, \ldots, c_n$ in the auxiliary equation are to be identified with $i_1, i_2, \ldots, x_n$, respectively. A branch symbol is not permitted in a storage index equation.

After writing the b-equations, the formulator ascertains the degree of each b-symbol (Def. 13.1) by inspecting the righthand sides of all the equations.

Rule 13.2. If b_k is of degree zero, no storage index equation is required. b_k will automatically be stored in the same storage each time it is computed.

Rule 13.3. If b_k is of degree one or two, and if b_k is not defined by a recursion or a vector equation, then a type B storage index, r_j say, is assigned to b_k. The symbol r_j is written to the left of b_k on the left side of the equation for b_k. A storage index equation of type B is added to the formulation to define r_j. If b_k is of degree one, the first form of the type B storage equation is used. The choice of the index i_1 will cause $b_k i_1$, to be stored in separate storages for all i_1. If b_k is of degree two, the second form of the type B storage index equation is used. The choice of the indices i_1 and i_2 will cause $b_k i_1 i_2$ to be stored

in separate storages for all i_1 and i_2. The function ϕ is selected by the formulator after he determines the structure of the (i_1,i_2) grid defined by the quantifiers of i_1 and i_2. The discussion on type A storage equations is applicable if q_{98} and q_{99} are replaced by i_1 and i_2, respectively.

Note that the same storage index can be applied to different b's if they have identical storage requirements.

Careful consideration of Rules 13.2 and 13.3 gives rise to several suggestions for writing b-equations.

In the interests of economy in the use of computer storage, the formulator should avoid introducing unnecessary b-equations. In the computation of the well-formed formulas which occur as parts of a b-term, results are stored only when necessary. (This storing is automatically programmed by the Encoder. It does not concern the formulator.) However, the final result of a b-term is automatically stored whether or not this is necessary for the progress of the computation. For example, in the equation

$$b_1 = \cdot + a_1a_2 + a_3a_4,$$

the intermediate result, $+ a_3a_4$, must and will be stored before the program proceeds. No other quantities will be stored, except the final result, b_1. Now, if we reformulate this as follows,

$$\begin{aligned} b_2 &= + a_1a_2, \\ b_3 &= + a_3a_4, \\ b_1 &= \cdot b_2b_3, \quad , \end{aligned}$$

then b_1, b_2 and b_3 will be stored, b_2 unnecessarily.

The indexing of a dependent variable, b_k, should be avoided if possible, since this will require the introduction of a storage index. This will cause the Encoder to produce a program in which the different values of b_k are stored in separate storages. A complete discussion of this point leads to rather involved and deep problems. We shall content ourselves, in the present report, with the formal rules and examples to illustrate what is at issue.

e.g. 13.1. Suppose it is required to compute $b_1 = \Sigma_{q_1=0}^{a_3} b_2(q_1)$, where $b_2(q_1) = a_1(q_1) \cdot a_2(q_1)$. This should be formulated as follows.

$$\begin{aligned} & b_o = f_o b_1, \\ \forall_{0 q_1 a_3} \quad & b_1 = {}_{11} + b_2 b_1 q_1, \; f_1 0, \\ & b_2 = \cdot a_1 q_1 a_2 q_1, \quad . \end{aligned}$$

Note that b_2 is written without an index, since only one value of b_2 at a time is needed, and the proper indexing is already contained in the right side of the b_2 -equation.

e.g. 13.2. Suppose $b_3 = \Sigma_{q_1=0}^{a_3-1} b_2(q_1)b_2(q_1+1)$, where again $b_2(q_1) = a_1(q_1) \cdot a_2(q_1)$. We shall formulate the problem in three ways, and mention a fourth.

Formulation 1.

$$r_1 = -a_3 1,\ r_2 = +1 q_1,$$
$$b_0 = f_0 b_3,$$
$$\forall 0 q_1 r_1\ b_3 = {}_{11} + b_3 q_1 \cdot a_1 q_1 \cdot a_2 q_1 \cdot a_1 r_2 a_2 r_2,\ f_1 0,$$

Formulation 2.

$$r_1 = -a_3 1,\ r_2 = +1 q_1,\ r_3 = f_1 q_2,$$
$$b_0 = f_0 b_2 b_3,$$
$$\forall 0 q_2 a_3 r_3 b_2 = \cdot a_1 q_2 a_2 q_2,$$
$$\forall 0 q_1 r_1 b_3 = {}_{11} + b_3 q_1 \cdot b_2 q_1 b_2 r_2,\ f_1 0,$$

Formulation 3.

$$r_1 = -a_3 1,\ r_2 = +1 q_1,$$
$$b_0 = f_0 b_3,$$
$$\forall 0 q_1 r_1\ (\ b_1 = {}_{51} f_1 b_2 q_1,\ f_1 0,$$
$$b_2 = {}_{51} \cdot a_1 r_2 a_2 r_2,\ \cdot a_1 0 a_2 0,$$
$$b_3 = {}_{51} + b_3 q_1 \cdot b_1 q_1 b_2 q_1,\ f_1 0,\ )$$

Another possible formulation would introduce a double recursion to compute b_2. We shall not go into this.

The preceding example is very informative and merits study. In formulation 1, the symbol b_2 is dispensed with entirely. b_3 is computed by a simple scalar recursion which involves a rather long b-term. When this b-term is computed, no intermediate results will be stored. Thus, this formulation would be fairly economical storagewise. Timewise, however, it is obviously inefficient since each product of a_1 by a_2 is performed twice.

In formulation 2, the duplicate computation of $a_1 \cdot a_2$ is avoided by computing all the products first and storing them for all q_2. Then b_3 is computed by simply calling for the successive pairs of b_2 values. Since b_2 is of degree one in this formulation, a B type storage index, r_3, is written to the left of the b_2 -equation, and a B type equation, $r_3 = f_1 q_2$,

indicates that b_2q_2 is to be stored in separate storages for all q_2. This formulation produces a faster program, but requires $(a_3 + 1)$ intermediate storages. If a_3 represents a large number, this formulation is uneconomical storagewise.

In formulation 3, b_3 is treated as the third component in a simple vector recursion. The second component is the product of a_1 by a_2. The first component is then defined merely as the previous value of the second component in the recursion, thus avoiding a duplicate calculation of the product $a_1 \cdot a_2$. The subscript 51 makes this a 'one-row' recursion, which means that a minimum of storage is used; i.e. the new value of a component replaces the previous value in storage. Therefore, this formulation is economical both timewise and storagewise. In fact, the Encoder will translate it into a program which is about as efficient as the program produced by a skilled human programmer, who will, in effect, program it as a vector recursion (although he may not think of it as such.)

In formulation 2 above, the use of a B type storage index equation is illustrated. The equation is of the simple form, $j = \phi i_1,$. The next example illustrates the B-type storage equation with two indices.

e.g. 13.3. Let it be required to compute three nxn matrices,

$$b_1(q_1,q_2) = a_1(q_1) + a_2(q_2),$$
$$b_2(q_1,q_2) = a_1(q_1) / a_2(q_2), \text{ and their product}$$
$$b_3(q_3,q_4) = \Sigma_{q_5=0}^{n-1} b_1(q_3,q_5)\, b_2(q_5,q_4).$$ The ADES

formulation is as follows (letting $a_3 = n-1$).

$$f_{51} = +c_2 \cdot c_1 + 1c_3,$$
$$r_1 = f_{51}q_1q_2a_3,$$
$$b_0 = f_0b_4b_5,$$
$$\forall 0q_1a_3 \forall 0q_2a_3b_4 = f_0b_1b_2,$$
$$r_1b_1 = +a_1q_1a_2q_2,$$
$$r_1b_2 = /a_1q_1a_2q_2,$$
$$\forall 0q_3a_3 \forall 0q_4a_3b_5 = f_0b_3,$$
$$\forall 0q_5a_3b_3 = {}_{11}+b_3q_5 \cdot b_1q_3q_5b_2q_5q_4,\ f_10,$$

Since b_1 and b_2 are of degree two in this formulation, the storage index, r_1, must be introduced and defined as shown. It causes the Encoder to compose a program which will store each value of b_1 and b_2 in a separate storage according to Computer Convention 4. The program for b_3 will later call these values out as they are needed. Note the use of phase equations

and quantifiers to prescribe the order of computation. The phase equation for b_5 must be introduced because the recursion equation for b_3 cannot be quantified more than once (Def. 11.1).

Rules 13.2 and 13.3 apply to the storing of non-recursive dependent variables. We now state rules for recursively-defined variables.

Rule 13.4. Let b_k be defined by a simple scalar recursion. b_k must be of degree one at least. If it is of degree one and occurs only with the recursion index, or with some index which depends on the recursion index, and with no other index, then no storage index need be assigned. The final value of b_k in the recursion will automatically be stored like any dependent variable of degree zero.

However, if b_k occurs with an index which is independent of the recursion index, then Rule 13.3 applies. The storage index pertains to the final value of b_k in the recursion. Since the recursion is repeated, there will be more than one final value. Each final value will be stored separately as prescribed by the storage index. The storage index is written immediately to the left of b_k, but to the right of the quantifier.

If the formulator wishes to store one or more intermediate values of b_k computed during the iterations, he must write the recursion as a course-of-values recursion. This automatically causes all values of b_k to be stored. No storage index is required.

Rule 13.5. If b_k is one component in a simple vector recursion, and if the other components are not also denoted by b_k, then Rule 13.4 applies to b_k. If the components are denoted by $b_k(i)$, no storage index can be assigned. If it is necessary to store $b_k(i)$ for all j, this must be formulated by relabeling $b_k(i)$, using the identity function.

Rule 13.6. If b_k is defined in a double recursion and its equation is outside the brackets, then Rule 13.4 applies. If the b_k-equation occurs inside the brackets of a double recursion, then b_k is necessarily of degree two in the formulation.

If the recursion is a course-of-values double recursion, then a B type storage index, r_n, is required as indicated in Rule 11.3. The index, r_n, is written immediately to the right of b_k on the left side of the equation. r_n is a function of the recursion indices i_1, i_2. The formulator must define r_n by the mapping of the (i_1, i_2) grid onto a linear grid, according to Convention 4. The structure of the (i_1, i_2) grid is determined by the quantifiers of i_1 and i_2. Thus, r_n specifies the storing of all values of $b_k(i_1, i_2)$.

If b_k occurs with one or two indices which are independent of the recursion indices, then Rule 13.4 applies for the storing of the final value of b_k.

If b_k is defined by an equation within the braces of a triple recursion, the degree of b_k in the triple recursion should be determined. A suitable B-type storage index should then be written to the right of b_k in the b_k -equation.

Rule 13.7. If y denotes a variable of degree one which is defined by a vector equation (Def.10.1), then no storage index need be written. If y is a variable of degree one which is defined by an indexed function (Defs. 10.2, 10.3 and Rule 10.1), the y-equation should contain a B-type storage index.

(Remark: The rules for storage indices, as given above, are to be regarded as temporary. It is planned to eliminate most or all storage indices in ADES III; i.e. the Encoder of ADES III will take over this task.)

Having written all necessary storage indices, the formulator can write the part of the formulation known as the 'Computer Table'.

Def. 13.4. Let $x_1^0, x_2^0, \ldots, x_k^0$ denote the independent variables of degree zero in a formulation, written in an order corresponding to the order of the data in external storage. (See computer conventions.) Let $x_1^1, x_2^1, \ldots, x_m^1$, denote the independent variables of degree one. Let $x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)}$ be the independent variables of degree two, and $j_1^{(2)}, \ldots, j_n^{(2)}$ the corresponding A-type storage indices. (The variables of degree one and two are in an order corresponding to the order of the blocks of data in external storage.) Let $x_{i_1}^1, x_{i_2}^1, \ldots, x_{i_s}^1$ denote the integer-valued variables, and $i_1, i_2, \ldots, i_s$ the respective independent indices with which they are to be identified (Def. 6.5.). Let $j_1, \ldots, j_p$ denote those independent indices which are used in recursions, and $N(j_1), \ldots, N(j_p)$ the respective numbers which specify the maximum number of values each index assumes. Finally, let $K_1, \ldots, K_H$ denote the B-type storage indices and $N(K_1), \ldots, N(K_H)$ the respective numbers which give the maximum number of values of each index. The Computer Table is a table arranged as follows:

x_1^0	1	Independent Variables of Degree Zero
x_2^0	1	
$\vdots$	$\vdots$	
x_k^0	1	
x_1^1	N_1^1	Independent Variables of Degree One
x_2^1	N_2^1	
$\vdots$	$\vdots$	
x_m^1	N_m^1	
$x_1^{(2)}$	$N_1^{(2)}$ $j_1^{(2)}$	Independent Variables of Degree Two
$x_2^{(2)}$	$N_2^{(2)}$ $j_2^{(2)}$	
$\vdots$	$\vdots$ $\vdots$	
$x_n^{(2)}$	$N_n^{(2)}$ $j_n^{(2)}$	
	;	Semi-colon
x_1^i	i_1	Integer-valued variables
x_2^i	i_2	
$\vdots$	$\vdots$	
x_s^i	i_s	
j_1	$N(j_1)$	Recursion Indices
$\vdots$	$\vdots$	
j_p	$N(j_p)$	
K_1	$N(K_1)$	B-type Storage Indices
$\vdots$	$\vdots$	
K_H	$N(K_H)$,	

where the various N's denote actual numbers which specify the maximum number of pieces of data supplied for the adjacent variables.

With the writing of the Computer Table, the formulation is virtually complete. It remains only to indicate the output; i.e. which results are to be punched out on cards, transferred to tape, printed, etc. The details of output will vary with the computer.

In ADES, output is indicated by writing an output symbol, d, with suitable subscript, on the left side of the appropriate b-equation. The details of output are concealed in this subscript, and will not concern us here. It is sufficient to assume that a list of available modes of output and their corresponding subscripts is in the possession of the formulator; e.g. d_{25} might mean 'punch this result in the second field of a five-field card;' d_{13} might mean 'print this result in the third field on the page and skip one line', d_{62} might mean 'write and copy onto tape 2', etc.

Rule 13.8. If it is desired to have all the values of the dependent $\left(\begin{matrix}\text{index}\\ \text{variable}\end{matrix}\right), \left(b_k^{r_k}\right)$, as output, an appropriately subscripted output symbol, d_n, should be written to the left of $\left(b_k^{r_k}\right)$ on the left side of the $\left(b_k^{r_k}\right)$ equation. If there is a B-type index to the left of b_k, then d_n should be written immediately to the left of this index.

In a recursion, this will cause only the final values of b_k to be processed for output. To effect output of all values in a recursion, d_n should be written to the right of b_k and to the left of the equal sign.

This completes the description of the input and output parts of a formulation.

14. Summary.

The Automatic Digital Encoding System is designed to enable the mathematician to present a mathematical problem to any modern electronic digital computer in a form closely resembling a conventional mathematical statement of the problem. ADES eliminates the programmer, with his specialized knowledge of one computer. All the clerical tasks which the programmer would perform are taken over by a machine, the Encoder. The mathematician, or formulator, must first prepare the problem as a set of equations suitable for digital computation. He then formulates the problem in the ADES language. This frequently requires little more than a transliteration of the original equations. A complete formulation of the problem in ADES consists of the following:

Computer Table,
Auxiliary Equations,
r – equations,
b – equations.

For the most part, the formulation of a problem can be planned and written with no specific computer in mind. It is true that the use of library functions does depend on the computer. However, some standard

assignment of subscripts for the elementary functions and other common library subroutines can be adopted; e.g. f_2 for $+$, f_3 for $-$, etc. Failing that, the formulator can use the conventional symbols, and when a particular computer is selected, the conventional symbols can be replaced by the correct function symbols. The same procedure applies to output symbols, which are somewhat more difficult to standardize.

When the formulation has been completed for a specific computer, it is punched in cards (or tape etc.) and loaded into the Encoder for that computer. The Encoder will translate the formulation into a program in the computer language, and will compile all necessary subroutines. The complete program with subroutines is then punched out on cards (or placed on tape) in a form ready for computation. This program can then be loaded into the computer. It will read the data into high-speed storage, perform the necessary computation, and yield the desired results as output.

REFERENCES

(1) S. C. Kleene, Introduction to Metamathematics, 1952, Van Nostrand.

(2) D. Hilbert and P. Bernays, Grundlagen der Mathematik, 1934, (Edwards Brothers)

(3) R. Péter, Rekursive Funktionen, 1951, (Budapest)

(4) C. C. Elgot, On Single vs. Triple Address Computing Machines, Journal of ACM, July 1954, pp.119–123.

APPENDIX I. Library Functions.

Most of the common library subroutines can be referred to simply by writing the properly subscripted function symbols in the formulation. In system II, the computer library is assumed to comprise mainly these comparatively elementary subroutines, and the preceding report is based on this assumption. This in no way implies that ADES is limited to such simple subroutines. As examples of more complex subroutines, we shall cite two situations which require somewhat special formulation.

First, if the subroutine involves a recursion, and if there are operands containing the recursion index in the formulation, a separate recursion equation must be written. For example, suppose f_{20} denotes a subroutine for summation, Σ. Then if we wish to formulate $b_1 = \Sigma_{q_1 = 0}^{10} a_1(q_1)$, it is clearly not sufficient to specify the quantity to be summed. One must also specify the index of summation (i.e. the recursion index) and its bounds. Further, if $\Sigma_{q_1 = 0}^{n} a_1(q_1)$ occurs as part of another formula, it must be denoted by a b-symbol, which must then be defined by a b-equation. In this case,

$$\forall 0q_1 10 \quad b_1 = f_{20} a_1 q_1,$$

defines the summation. Note that this equation is not considered to be a recursion equation (i.e. subscript of equal sign is 00), since the subroutine will take care of all the recursive features except the quantification.

The second situation which complicates a library subroutine can best be explained by an example. Suppose there is a subroutine for the numerical integration of any real elementary function, $g(x)$, over a finite interval, $[a_3, a_4]$. The numerical formula would be of the type,

$$\Sigma_{i=0}^{n} a_1(i) \cdot g(b_2(i)),$$

where the $a_1(i)$ are constant coefficients independent of the function g, and the limits a_3, a_4, while $b_2(i)$ depends on a_3 and a_4. In fact, $b_2(i) = a_3 + \left[\frac{a_4 - a_3}{n}\right] i$. Therefore, the integration subroutine depends on values of the function g which cannot be computed until $b_2(i)$ has been computed. The formula for g must be written with the argument, $b_2(i)$, and g itself must be identified by a dependent variable. A suggested procedure for referring to such library functions is the following.

Let f_{30} denote the subroutine which computes $b_1 = \int_{a_1}^{a_2} g(x)dx$. To illustrate, let $g(x) = \sin x$. The formulation for b_1 would be,

$$\forall 0q_1 n \quad b_1 = f_{30} a_1 a_2 b_2 b_3,$$
$$b_3 = \sin b_2, \quad .$$

The above format can be standardized, that is, for each library function of the type described, the formulator writes a b-equation, quantifying it if an index is involved. On the right side, he writes the pertinent f-symbol, f_{mn}, followed by indexed independent variables, which represent data to be supplied to the subroutine, and dependent variables. Of the dependent variables, the last one denotes the function to be operated on by f_{mn}. The last b is defined by a b-equation. The other b-symbols denote quantities computed within the subroutine for use in the formula for the last b-symbol. They are not defined by b-equations.

The Encoder must be slightly modified to cope with these special library functions. It must recognize the f-symbol before programming the last b. It must then operate so that the part of the subroutine which computes all b's but the last is compiled into the program first. These b's are replaced by addresses which are relative to the location of the subroutine in the program. Then the last b is programmed. The program for the library function is then completed.

APPENDIX II.

The order in which the b-equations are programmed is determined by (1) the phase equations, (2) the parentheses in vector equations and recursions, and (3) by the order in which dependent variables are written on the right side of an equation.

The Encoder determines the order of computation in a natural manner as follows. Starting with the master phase equation, it scans the right member from left to right for dependent variables. Let the first dependent variable which it finds be denoted by y_1. Assume first that the y_1 -equation is not a branch, vector, or recursion equation. The Encoder scans the right side of the y_1 -equation for dependent variables. If there are none, y_1 is programmed and the Encoder returns to the phase equation to obtain the next dependent variable. If the y_1 -equation contains a dependent variable, y_2 say, then the y_2 -equation is inspected for dependent variables. If no dependent variables are found in the y_2 -equation, it is programmed and the Encoder returns to the y_1 -equation and scans right again for dependent variables. If y_2 is the only dependent variable in the right member of the y_1 -equation, then y_1 is programmed next, and the Encoder returns to the master phase equation. If the y_1 -equation contains another dependent variable, y_3 say, the Encoder will proceed with y_3 as with the y_2 -equation. Eventually, all dependent variables on the right side of the y_1 -equation will have been programmed, and then y_1 itself is programmed.

In this way, the Encoder traces its way through the formulation, programming the variables as they are encountered, whenever possible. This order is interrupted, however, if a variable, y_n, is one component in a vector equation. In that case, the Encoder will arrange to program the components in the order in which they are written, beginning at the left parenthesis and proceeding to the right. Similarly, in double and triple recursions, the equations are programmed in the order in which they are written.

In a branch equation, the leftmost formula is programmed first. Normally, if a variable, y_i, is referred to in several formulas, it is programmed only once, at the time of the first referral. An exception to this must be made in the case of a branch equation in which a dependent variable, y_i, is referred to in both branches. This is inefficient formulation, since y_i will have to be programmed twice. This situation can always be avoided by writing a vector equation in which y_i is the first component and the branch equation defines the second component, for then y_i would be referred to prior to the branch and programmed once. Nevertheless, the Encoder will accept the inefficient formulation, and program y_i once in each branch.

APPENDIX III. Examples.

Each example will be stated first in a more or less conventional mathematical way, and then a complete ADES formulation will be given.

EXAMPLE 1.

Given the data points $a_1(q_1)$, where $0 \leq q_1 \leq a_2 + 1$.

It is required to compute

$$b_1 = \Sigma_{q_1=0}^{a_2+1} \left[a_1(q_1)\right]^{q_2}$$

for all q_2, where $0 \leq q_2 \leq 2a_o$. (The b_1 are the elements of the normal matrix in the least square fitting of a polynomial of degree a_o over $a_2 + 2$ points.) Two ADES formulations are given.

Formulation 1.

Computer Table. $a_o 1\ a_2 1\ a_1 50;\quad a_3 q_2\ r_2 200\ q_1 50\ q_2 40,$

Auxiliary Equations. $f_{50} = +c_2 \cdot c_1 + c_3 1,$

r – Equations.
$$r_o = -r_1 1,$$
$$r_1 = \cdot 2a_o,$$
$$r_2 = f_{50} r_3 r_4 r_1,$$
$$r_3 = +1 q_1,$$
$$r_4 = +1 q_2,$$

b – Equations.
$$b_o = f_o b_2 b_3,$$
$$\forall\, -1 . q_1 a_2 (\forall\, -1 . q_2 r_o [b_2 r_2 = {}_{42}P_4 f_{11} a_3\ -1,\ f_1 1, \cdot a_1 r_3 b_2 r_3 q_2,])$$
$$\forall\, 0 q_4 r_1\ b_3 = f_o b_1,$$
$$\forall\, 0 q_3 a_2 d_{15} b_1 = {}_{11}+ b_1 q_3 b_2 q_3 q_4,\ f_1 0,$$

Remarks: $b_2(q_1, q_2) = \left[a_1(q_1)\right]^{q_2}$ is defined by a double recursion of the course-of-values type (subscript 42). q_1 is the row index and q_2 is the column index. Actually, there is no recursion on q_1. However, since all values of b_2 are to be stored, advantage is taken of this property of the course-of-values double recursion. The storage index, r_2, defines the structure of the $b_2(q_1, q_2)$ array as rectangular. The initial values, $b_1(q_1, 0) = 1$, are defined by means of a branch equation. The branch condition '$f_{11} a_3\ -1$' means '$a_3 \leq -1$,' where a_3 is an integer-valued variable identified

with q_2 in the Computer Table. Since b_2 on the left of the equation is really $b_2(q_1 + 1, q_2 + 1)$, the condition $q_2 \leq -1$ is equivalent to $q_2 + 1 \leq 0$, i.e. $b_2(q_1 + 1, 0)$. This also explains why the lower bound of q_1 is -1, and why the index of a_1 is $r_3 = q_1 + 1$.

The master phase equation causes b_2 to be computed first. It next refers to b_3, which leads to another phase equation. The b_3-equation is introduced to permit the quantifier of q_4 to be applied to b_1. This is necessary since b_1 is defined by a simple recursion equation, and Rule 11.1 prohibits more than one quantifier in a simple recursion equation. The output symbol, d_{15}, will cause the successive values of $b_1(q_2)$ to be punched out five per card, as they are computed.

The above formulation is economical of time but wasteful of storage, since all values of b_2 are stored. A better formulation, given below, causes b_1 and b_2 to be computed in one double recursion simultaneously, and in such a way that only one row of values is stored for b_1 and b_2. However, to obtain output, the final row of b_1 values must be called out in a special b-equation, b_4.

Formulation 2.

Computer Table. $a_0 1.\ a_2 1.\ a_1 50.\ ;\ a_3 q_2\ a_4 q_1\ q_1 50.\ q_2 40.,$

r-equations. $r_0 = -r_1 1.\ ,\quad r_1 = \cdot 2.\ a_0,$

$r_3 = +1.\ q_1\ ,\quad r_4 = +1 q_2,\ r_5 = +1 a_2,$

b-equations. $b_0 = f_0 b_2 b_4\ ,$

$$\forall -1.q_1 r_5 (\forall -1.q_2 r_0 [b_2 =_{32} P_4 f_{11} a_4 -1,\ f_1 0,\ f_1 b_3,$$
$$b_1 =_{32} P_4 f_{11} a_4 -1,\ f_1 0,\ + b_2 r_3 r_4 b_1 q_1 r_4,\ ])$$
$$b_3 = P_4 f_{11} a_3 - 1,\ f_1 1, \cdot a_1 q_1 b_2 r_3 q_2,$$
$$\forall 0 q_3 r_1 d_{15} b_4 = f_1 b_1 q_3,$$

Remarks: To permit the simultaneous computation of b_1 and b_2, the initial values, $b_2(0, q_2 + 1)$, are taken to be zero. This forces us to use a triple branch for b_2, necessitating tha branch equation for b_3. Note that this equation is written outside the parentheses because it will be referred to in the course of programming b_2. In the Computer Table, one sees that there are two integer-valued variables, namely, $a_3 = q_2$ and $a_4 = q_1$. Note also that a maximum of fifty data points, a_1, is permitted. Since a_0 and a_2 are constants, they each have one datum. The maximum number of values of q_1 is 50. The maximum number of values of q_2 is 40.

EXAMPLE 2.

Given the polynomial approximation,

$$\arctan x \doteq p(x) \equiv \Sigma_{q_1=0}^{7} A_{q_1} x^{2q_1+1},$$

valid for $-1 \leq x \leq 1$, compute a table of arctan x for $-9 \leq x \leq 9$, in steps of 0.1.

We introduce ADES notation as follows. Let $a_0 = 0.1$; $b_6 = x$, $a_1 q_1 = A_{q_1}$; $a_2 = \pi/2$; $b_1 = \arctan x$; $b_2 = p(x)$; $b_3 = x$ when $|x| \leq 1$, and $1/x$ when $|x| > 1$; $b_4 = 1$ when $|x| \leq 1$ and -1 when $|x| > 1$; $b_5 = \pi/2$ when $x > 1$, 0 when $|x| \leq 1$, and $-\pi/2$ when $x < -1$; clearly, $\arctan x = b_5 + b_4 \cdot p(b_3)$. Letting f_{10} denote '$<$', f_{11} '$\leq$', $f_6 = -f_1$, f_9 the absolute value, we have

FORMULATION

Computer Table. $a_0 1\ a_2 1\ a_1 8$; $a_3 q_2 q_1 8$,

r-equations. $r_1 = -6q_1$, $r_2 = f_1 7.$,

b-equations. $b_0 = f_0 b_1$,

$\forall 0q_2 180.$ d_{22} $b_1 = +b_5 \cdot b_4 b_2$,

$b_5 = P_4 f_{10} b_6 -1.,\ f_6 a_2,\ f_1 b_7$,

$b_7 = P_4 f_{11} b_6 1.,\ f_1 0.,\ f_1 a_2$,

$d_{12} b_6 = -.a_3 a_0 9.$,

$b_8 = f_9 b_6$,

$b_4 = P_4 f_{11} b_8 1,\ f_1 1,\ f_6 1$,

$b_3 = P_4 f_{11} b_8 1,\ f_1 b_6,\ /1.b_6$,

$b_{10} = .b_3 b_3$,

$b_2 = .b_3 b_9$,

$\forall 0q_1 6 b_9 =_{11} + .b_9 q_1 b_{10} a_1 r_1,\ f_1 a_1 r_2$,

* * *

EXAMPLE 3.

The usual 'differential corrections' problem can be stated as follows. g is a function of the independent variable, a_1, and the m + 1 parameters, $b_{10}(0), \ldots, b_{10}(m)$. A table of values, $a_2(q_1)$ vs. $a_1(q_1)$, $0 \leq q_1 \leq n$, is given; i.e.

$$a_2(q_1) = g\left(a_1(q_1), b_{10}(0), \ldots, b_{10}(m)\right).$$

It is required to determine the 'best' values of the parameters.

The problem is linearized by writing

$$a_2(q_1) - g(a_1(q_1), b_{10}(0,0), \ldots, b_{10}(m,0)) \doteq \Sigma_{i=0}^{m} g_i \cdot \Delta b_{10}(i),$$

where g_i denotes the partial of g with respect to $b_{10}(i)$. The 'corrections', $\Delta b_{10}(i)$, are then determined by the method of least squares, and the process is iterated until the difference in successive sums of squares of residuals is sufficiently small. One possible order of computation is as follows.

1.) Compute values of $b_{10}(i)$, starting with initial guesses, $a_3(i)$.

2.) Compute the augmented 'observational' matrix,

$$0 \leq j \leq m, \quad b_1(i,j) = g_j\left(a_1(i), b_{10}(0), \ldots, b_{10}(m)\right),$$
$$b_1(i,m+1) = a_2(i) - g\left(a_1(i), b_{10}(0), \ldots, b_{10}(m)\right),$$

for $0 \leq i \leq n$.

3.) Compute the augmented 'normal' matrix, b_2, by multiplying the matrix b_1 by its transpose, at the same time computing the augmented triangular, 'Choleski' matrix, b_3.

4.) Solve the linear system of normal equations for $b_8 = \Delta b_{10}$ by the Choleski method.

5.) Test whether the difference in successive sums of the squares of the residuals is less than some constant, a_{10}. Repeat 1 – 5 if necessary.

To write an ADES formulation, we denote the g_i and $a_2 - g$ by the indexed function, ${}_2f_{50}r_5$, $0 \leq r_5 \leq m+1$; m by a_4; n by a_5. Let $g = e^{Ax} + e^{Bx^2}$, where A and B are denoted by $b_{10}(0)$, $b_{10}(1)$, respectively. Thus, $g_0 = \delta g/\delta A = xe^{Ax}$ and $g_1 = x^2 e^{Bx^2}$. We denote the exponential function by f_e.

FORMULATION

Computer Table. $a_4{}^1\ a_5{}^1\ a_{10}{}^1\ a_1{}^{50}\ a_2{}^{50}\ a_3{}^9$; $a_{10}q_1a_6q_2a_7q_1a_8q_5a_9q_6$ $r_{12}{}^9\ r_4{}^{450}\ r_9{}^{55}\ r_{13}{}^9$,

Auxiliary Equations. $(f_{50} = {}_{09} \cdot c_1 f_e \cdot c_1 c_2,$
$f_{50} = {}_{09} \cdot c_1 \cdot c_1 f_e \cdot \cdot c_1 c_1 c_3,$
$f_{50} = {}_{09} -c_4 + f_e \cdot c_1 c_2 f_e \cdot c_3 \cdot c_1 c_1,)$
$f_{51} = +c_2 + 1 \cdot + 1 c_1 c_3,$
$f_{52} = +c_2 + 1 \cdot + c_1 1 + c_1 2,$

r-equations. $r_2 = -1a_4,\ r_3 = +1q_2,$
$r_5 = +1q_4, r_6 = +1q_3,\ r_7 = f_1 0,\ r_8 = f_1 1,$
$r_{10} = +1q_5,\ r_{11} = +1q_6,\ r_{14} = +1a_4,$
$r_{15} = +1q_9,\ r_{16} = -a_4 r_{15},\ r_{17} = -a_4 q_{10};$
$r_{12} = f_1 q_2,\ r_{13} = f_1 q_9,\ r_4 = f_{51} q_3 q_4 r_{14},$
$r_9 = f_{52} q_5 q_6,\ r_1 = f_\mu q_1 10.b_{12},$
$r_{19} = +1q_1,\ r_{18} = f_1 q_1,$

b-equations. $\forall -1q_1 r_1 \{ \forall -1q_2 r_2 (d_{15} b_{10} r_{12} = {}_{03} P_4 f_{10} a_7 0,$
$f_1 a_3 r_3, + b_{10} r_3 b_8 r_3,)$
$\forall -1q_3 r_5 (\forall -1q_4 a_4 [b_1 r_4 = {}_{03} f_{50} r_5 a_1 r_6 b_{10} r_7 b_{10} r_8 a_2 r_6,])$
$\forall -1q_5 a_4 (\forall -1q_6 q_5 [b_2 r_9 = {}_{03} f_1 b_{13},$
$b_3 r_9 = {}_{03} P_4 f_{10} a_9 a_8, \sqrt{b_4} b_3 r_{11} r_{11}, \sqrt{b_4},])$
$\forall -1q_9 r_2 (b_8 r_{13} = {}_{03} / -b_3 r_{14} r_{16} b_9 b_3 r_{16} r_{16},)$
$b_{14} r_{18} = f_1 b_2 r_{14} r_{14}, \}$
$d_{11} b_{12} = P_4 f_{10} a_{10} 0,\ f_1 1,\ -b_{14} r_{19} b_{14} q_1,$
$b_4 = -b_2 r_{10} r_{11} b_5,$
$\forall 0 q_8 q_6\ b_5 = {}_{11} + \cdot b_3 r_{11} q_8 b_3 r_{10} q_8 b_5 q_8,\ f_1 0,$
$\forall 0 q_{10} q_9 b_9 = {}_{11} + b_9 q_{10} \cdot b_3 r_{17} r_{16} b_8 q_{10},\ f_1 0,$
$\forall 0 q_7 a_5\ b_{13} = {}_{11} + b_{13} q_7 \cdot b_1 r_{10} q_7 b_1 r_{11} q_7,\ f_1 0,$

EXAMPLE 4.

Given the system of n ordinary first order differential equations, $0 \leq n \leq m,\ \frac{dy_n}{dx} = g_n (x, y_0, y_1, \ldots, y_m),$

and the boundary values, $y_i(x_o)$, it is required to obtain a solution by the Runge-Kutta fourth-order method.

Using the mesh size, h, the recursion formula can be written as follows.

$$0 \leq n \leq m \quad \Delta y_n(j+1) = h(k_{1n} + 2k_{2n} + 2k_{3n} + k_{4n})/6,$$
$$y_n(j+1) = y_n(j) + \Delta y_n,$$
$$0 \leq i \leq m \quad k_{1i}(j+1) = g_i(x(j), y_o(j), \ldots, y_m(j)),$$
$$k_{2i}(j+1) = g_i(x_j + h/2,\ y_o(j) + hk_{10}/2, \ldots, y_n(j) + hk_{1m}/2)$$
$$k_{3i}(j+1) = g_i(x_j + h/2,\ y_o(j) + hK_{20}/2, \ldots,\ y_n(j) + hk_{2m}/2),$$
$$k_{4i}(j+1) = g_i(x_j + h,\ y_o(j) + hk_{30}, \ldots, y_n(j) + hk_{3m}),$$

To illustrate an ADES formulation, it is sufficient to consider the case $m = 2$, where $g_o(x, y_o, y_1) = y_1 x / \sin y_o$, and $g_1(x, y_o, y_1) = y_o y_1 + x_o$. The computation of y_o and y_1 will be formulated as a triple recursion in which the main recursion index is j. For each value of j, we compute Δy_n, y_n, for all n. Then, by a double recursion, we compute the k's.

The following ADES notation is adopted:

$a_o = m$; $a_1 = h$; $a_2 q_2 = y_{q2}(x_o)$; $a_3 = x_o$; $q_1 = j$; $q_2 = n$;
$b_1 = \Delta y$; $b_2 = y$; $b_4 q_3 q_4 = k_{q3 q4}$, $1 \leq q_3 \leq 4$, $a_4(0) = 0$, $a_4(1) = 1/2$,
$a_4(2) = 1/2$, $a_4(3) = 1$; $f_{50} = g$; $b_3 = x_j$;

FORMULATION

Computer Table. $a_6 1 a_o 1\ a_1 1\ a_3 1\ a_4 4\ a_2 9$; $a_5 q_1\ a_7 q_3\ r_1 9\ r_7 45\ r_9 9$,

Auxiliary Equations. $(f_{50} = {}_{09}/\cdot c_3 c_2 \sin c_1,$
$f_{50} = {}_{09} + \cdot c_1 c_2 c_3,\ )$
$f_{51} = + c_2 \cdot c_1 + c_3 1,$

r-equations. $r_1 = f_1 q_2$, $r_7 = f_{51} q_3 q_4 a_o$, $r_8 = +1 q_1$, $r_9 = f_1 q_5$,

b-equations. $b_o = f_o b_1,$

$\forall -1 q_1 a_6 \{\ \forall\ 0 q_2 a_o\ (b_1 r_1 = {}_{03} P_4 f_{10} a_5 0,\ f_1 0,$
$\cdot a_1 / + b_4 1 q_2 + \cdot 2 b_4 2 q_2 + \cdot 2 b_4 3 q_2 b_4 q_2 6,$
$d_{15} b_2 r_1 = {}_{03} P_4 f_{10} a_5 0,\ f_1 a_2 q_2,\ + b_2 q_2 b_1 q_2,)$
$d_{21}\ b_3 = {}_{03} + a_3 \cdot + a_5 1 a_1,$

$$\forall 0q_3{}^4 \left(\forall 0q_4 a_0 \left[b_4 r_7 = {}_{03} P_4 f_{11} a_7 0,\ f_1 0,\ f_{50} q_4 b_5 0 b_5 1 b_6, \right.\right.$$
$$\forall 0q_5 a_0 \left[b_5 r_9 = {}_{03} + b_2 q_5 \cdot\cdot a_1 b_4 q_3 q_5 a_4 q_3, \right]$$
$$\left.\left. b_6 = {}_{03} + b_3 \cdot a_1 a_4 q_3, \right) \right\}$$

Aeroballistic Research Department
External Distribution List for Applied Mathematics (X3)

No. of Copies	
	Chief, Bureau of Ordnance Department of the Navy Washington 25, D. C.
1	Attn: Ad3
1	Attn: Ad6
1	Attn: Ree
1	Attn: Re9a
	Chief, BuAer Washington 25, D. C.
3	Attn: TD-414
	Commander, U. S. NOTS Inyokern, China Lake, Calif.
1	Attn: Technical Library
1	Attn: Code 503
	Commander, NAMTC Point Mugu, California
2	Attn: Technical Library
	Superintendent U. S. Naval Postgraduate School Monterey, California
1	Attn: Tech Rpts Section
	Director, NRL Washington 25, D. C.
1	Attn: Code 2021
	Officer in Charge, NPG Dahlgren, Virginia
2	Attn: Technical Library
	Office, Chief of Ordnance Washington 25, D. C.
1	Attn: ORDTU
	Chief, AFSWP Washington 25, D. C.
1	Attn: Document Library Br.
	Commander, WADC Wright-Patterson AF Base Ohio
3	Attn: WCOSI-3

No. of Copies	
	Commanding General Aberdeen Proving Ground, Md.
2	Attn: Tech Info Br.
2	Attn: Director, BRL
	Commanding General Redstone Arsenal Huntsville, Alabama
1	Attn: Aero. Lab, GMDD
5	ASTIA Document Service Center Knott Building Dayton 2, Ohio
	NACA High Speed Flight Station Box 273 Edwards AF Base, Calif.
1	Attn: Mr. W. C. Williams
	NACA Ames Aeronautical Laboratory Moffett Field, California
1	Attn: Librarian
	NACA Langley Aeronautical Laboratory Langley Field, Virginia
1	Attn: Librarian
1	Attn: Adolf Busemann
2	Attn: John Stack
	NACA Lewis Flight Propulsion Lab. 21000 Brookpark Rd. Cleveland 11, Ohio
1	Attn: Librarian
	NACA 1512 H Street, N. W. Washington 25, D. C.
	Commanding Officer, DOFL Washington 25, D. C.
1	Attn: Lib., Rm 211, Bldg. 92

Aeroballistic Research Department
External Distribution List for Applied Mathematics (X3a)

No. of Copies	
	Chief, Fluid Mechanics Section National Bureau of Standards Washington 25, D. C.
	National Bureal of Standards Washington 25, D. C. Attn: Applied Math Div.
6	Commanding Officer Office of Naval Research Br. Off. Box 39, Navy 100 Fleet Post Office New York, New York
	Langley Aeronautical Laboratory Langley Field, Virginia Attn: Theoretical Aerodynam. Division
	Naval Research Laboratory Washington 25, D. C. Attn: Dr. H. M. Trent Code 6230
	Case Institute of Technology Cleveland 6, Ohio Attn: G. Kuerti
	Massachusetts Institute of Tech. Cambridge 39, Mass. Attn: Prof. Joseph Kaye Room 1-212
	The Johns Hopkins University Charles and 34th Streets Baltimore 18, Maryland Attn: Dr. Francis H. Clauser
2	Director Inst. for Fluid Dynamics and Applied Mathematics University of Maryland College Park, Maryland
	Cornell University Graduate School of Aero. Eng. Ithaca, New York Attn: Prof. W. R. Sears

No. of Copies	
	AERCON, INC. 560 Punahou St. Altadena, California
	University of Michigan Randall Laboratory Ann Arbor, Michigan Attn: Prof. Otto Laporte
	Brown University Providence 12, Rhode Island Attn: Prof. William Prager
	University of Michigan Engineering Research Institute Ann Arbor, Michigan Attn: Mr. H. E. Stubbs Research Associate

No. of Copies

1 Oak Ridge National Laboratory
Oak Ridge, Tennessee
Attn: Dr. A. S. Householder

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. C. C. Elgot

1 Los Alamos Scientific Laboratory
Los Alamos, New Mexico
Attn: Dr. S. M. Ulam

1 University of Wisconsin
Madison 6, Wisconsin
Attn: Dr. S. C. Kleene

1 National Bureau of Standards
Washington 25, D. C.
Attn: Dr. H. Antosiewicz

1 National Bureau of Standards
Washington 25, D. C.
Attn: Dr. F. L. Alt

1 Pennsylvania State College
State College, Pa.
Attn: Dr. H. B. Curry

1 Electronic Computer Project
Institute for Advanced Study
Princeton, N. J.
Attn: Dr. H. H. Goldstine

1 Applied Mathematics Laboratory
David Taylor Model Basin
Washington 7, D. C.
Attn: Dr. H. Polachek

1 University of California
Berkeley 4, California
Attn: Dr. D. H. Lehmer

1 Raytheon Company
Waltham 54, Mass.
Attn: Dr. R. F. Clippinger

No. of Copies

1 Division of Computing Services
Institute of Math. Sciences
New York University
New York 3, N. Y.
Attn: Dr. E. Bromberg

1 Westinghouse Electric Corp.,
Pittsburgh 30, Pa.
Attn: Dr. C. W. Adams

1 Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge 39, Mass.
Attn: Dr. Jay Forrester

1 Sperry Rand Corporation
Philadelphia, Pa.
Attn: Dr. Grace Hopper

1 University of Pennsylvania
Philadelphia, Pa.
Attn: Dr. S. Gorn

1 Data Processing Department
Computing Devices of Canada
P. O. Box 508
Ottawa 4, Ontario
Attn: Mrs. M. Larmour

1 Applied Mathematics
University of Illinois
Graduate College
168 Engineering Research Laboratory
Urbana, Illinois
Attn: Mr. J. P. Nash

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. John W. Carr III

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. J. B. Wright

No. of Copies

1 Methods Division
The Prudential Insurance Co. of America
Newark 1, New Jersey
Attn: Mr. J. E. Coachman

1 Director
National Security Agency
3801 Nebraska Ave., N. W.
Washington 16, D. C.
Attn: LIB Acquisitions

1 Electronic Data Processing Development
General Electric Company
One River Road
Schenectady 5, New York
Attn: Mr. J. W. Pontius

1 Air Force Cambridge Research Center
L. G. Hanscom Field
Bedford, Massachusetts
Attn: Mr. Bert F. Krauss, CRRI

1 Computer Control Co.,
10966 LeConte St.,
W. Los Angeles, Calif.
Attn: Mr. Frank Stockmal

1 The Equitable Life Assurance Society
393 Seventh Ave.,
New York 1, N. Y.
Attn: Mr. Walter L. DeVries

1 Metropolitan Life Insurance Co.,
One Madison Ave.,
New York 10, N. Y.
Attn: Mr. Robert D. Acker

1 Digital Computer Laboratory
Convair
San Diego 12, Calif.
Attn: Mr. Ben Ferber

1 Computer Techniques Development
Investigations Section
AGT Development Department
Building 300
General Electric Co.,
Cincinnati 15, Ohio
Attn: Mr. D. L. Shell

No. of Copies

1 New York University
College of Engineering
University Heights
New York 53, N. Y.
Attn: Mr.Emanuel Mehr

1 Burroughs Corporation
Research Center
Paoli, Penna.
Attn: Mr. Jos. Deutsch

1 Programming and
Operation Research
Hughes Aircraft Co.,
Culver City, Calif.
Attn: Mr. Leon Gainen

1 Systems Analysis Dept.,
Remington Rand
Engineering Research
Association Division
1902 West Minnehaha Ave.,
St. Paul W4, Minnesota
Attn: Mr. B.F.Cheydleur