AN ON-LINE COMPUTING CENTER FOR SCIENTIFIC PROBLEMS

M19-3U3

Glen J. Culler* and Burton D. Fried

Revised June 1963

This work was supported by the Air Force Systems Command Data Processing Laboratory, Rome Air Development Center, Griffiss Air Force Base, New York, under Contract No. AF 30(602)-2762.

---

*Currently on leave from the University of California, Santa Barbara, California

TRW COMPUTER DIVISION
Thompson Ramo Wooldridge Inc.
8433 Fallbrook Avenue
Canoga Park, California

ABSTRACT

An on-line digital system allowing an unusually direct coupling between the user (physicist, mathematician, engineer) and the computer is described. This system, which has been successfully operated during the past six months, was designed principally to provide assistance for problems whose structure is partially unknown (and frequently surprising). These typically require the development of new methods of attack, and hence an amount of program experimentation not feasible with classical computer center organizations. With the system described here, the interaction between user and computer is close enough to permit effective use of a scientist's intuition and of his detailed understanding of techniques appropriate to his special field. He is able to construct, with ease, and with no necessity for a knowledge of conventional programming techniques and procedures, machine representations of those tools he considers essential to his area, and then use these, on-line, to study or solve problems of interest. The current system is described in detail and its application to a particular illustrative problem is outlined. Implications concerning the extension of the technique to typical, large digital computer installations are given.

I. INTRODUCTION

Despite the impressive achievements in computer hardware and programming techniques in recent years, the most significant gains presently realizable are associated with new approaches to the use, the organization and the logical structure of computers. One line of research has had as its goal the assumption by the computer of certain activities generally associated with human beings; the resultant study of learning machines, adaptive machines, et al has been very fruitful. Quite a different approach has been taken by those who instead seek ways of improving the man-machine communication so that the computer can more effectively assist the man in those jobs (requiring intuition, judgment, evaluation) for which he is best suited. Although no universally accepted nomenclature seems to exist, it is sometimes characterized as a means of getting a man "on-line" with a computer[1], as opposed to his usual "off-line" status of wading through reams of computer print-out .

Adopting this term, we describe here an operating "on-line" computer research center which provides an unusually close coupling between the man who originates a problem and a (modern, large electronic digital) computer. We hope this example of what can be accomplished using computer hardware well within the existing state-of-the-art will be useful to others concerned with the development of on-line techniques.

This work was initially motivated by the troubles which have been commonly encountered in using a computer to solve research problems whose structure is for the most part unknown and frequently

surprising. It is notoriously difficult to obtain a satisfactory computer program if one does not understand, a priori, the general character of the solution. In fact, information about the general character is often what we really want, rather than quantitative details. It is possible in principle to attempt a kind of experimental mathematics, starting with some promising method of solution and the associated program and modifying one or both in the light of the results obtained. However, the lapse of time between the selection of a new method, or the modification of an old one, and the return of information from the computer to the user is in most cases so long as to make this almost infeasible.

The source of the difficulty is basically the poor communication between the "user" (by which we shall henceforth mean the scientist or mathematician who originates a problem and knows most about it) and the computer, consequent upon considerations of economy and, as well, upon the inherent difficulty of imparting to a programmer the detailed and specialized knowledge one acquires about a particular problem area after working in it for some time. One anticipates a significant improvement in a system, such as that described here, which provides for a rapid, direct, comfortable interchange of information between man and machine. In fact, however, one reaps even greater rewards; if the communication link is established in the proper way it becomes possible for the user to apply, simultaneously, to the problem his own intuition, experience, and knowledge of specialized techniques on the one hand and the tremendous computational power of the machine on the

other. As we shall see, it is possible for him to build a representation, in the computer, of those analytic tools he believes valuable for a particular problem or problem area. Without any necessity for learning conventional programming techniques, he is able, using only the concepts of classical mathematics, to create his own machine language, one tailor-made to his own needs. He can freely manipulate the elements of this language, in precisely the same fashion one composes mathematical techniques, and can easily modify them to incorporate the knowledge gained from their use in problem solving, so that his computing capability grows with his understanding of the problems.

We shall describe this "on-line" system from the point of view of a typical user rather than that of a "computer expert", by which we shall henceforth mean someone skilled in the art of programming, as opposed to the "user" whom we assume to be totally unversed in such matters. The programming principles and details will be the subject of a separate article. We shall only discuss the presently existing system, as it has been operating since August, 1962. While this will inevitably entail the mention of certain specific aspects of the particular computer uses (AN/FSQ-27; RW-400), it should be kept firmly in mind, that while the detailed organizational choices were such as to take maximum advantage of the particular characteristics of this machine, the on-line techniques are in no way dependent upon

these characteristics. In the last chapter we sketch a method for using these on-line techniques with a standard operational computer center, a method consistent with the usual economic constraints on computer time.

In what follows, we restrict ourselves to the use of on-line techniques in the solution of mathematical and physical problems, this being the area of principal interest to us and the only one in which we have actual experience with an on-line system. We believe, however that the techniques can be extended to quite different areas of computer applications, a point to which we return in the last chapter. Meanwhile, we shall, in the interests of clarity, confine ourselves to a very specific description of the present system.

This work is an extension of an initial effort in which a particular problem, the energy gap integral equation of the Bardeen-Cooper-Schrieffer theory of superconductivity, was solved with an on-line approach[2]. However, in that work, which was carried out in the period July through December 1961, all of the subroutines for the problem were programmed in a conventional way. While the user was free to compose these elements in various ways using the control and display capabilities of the control console in solving the gap equation, he had no freedom to modify, on-line, the subroutines or create new ones, a freedom which is an essential characteristic of the present system. Thus, the earlier work comprised some aspects of items A and B, described below, but none of item C, the console programming.

II. THE ON-LINE SYSTEM

At the outset, we should emphasize that each aspect of the design of such a system involves a number of choices. We shall describe here our own, with no representation that they are in every case the optimum ones; in fact, in some cases experience has indicated how some of these might be improved. Nonetheless, the resulting system operates in a very satisfactory manner.

Three principal features, independent but interacting, characterize the system:

A. Functional Orientation

The programming structure is such that in the computer, as it appears to the user, functions (sets of 101 points) rather than individual numbers constitute the elements while the repertoire of "commands" consists of operations on functions (e.g., arithmetic, differential, and integral operations).

B. Control and Display Capability

Central to the operation of the system is a control console having a number of push buttons or keys, which allow for user control of the computer, and two 17 inch CRT oscilloscopes (with line-drawing capability) which provide direct graphical representation of computational results. An 8 inch CRT with alphanumeric capability and a flexwriter provide numerical output when required.

C. Console Programming

A simple procedure allows the user to construct, directly at the console, new subroutines, using as building blocks an initial

set of hand programmed[3] subroutines, plus any subroutines previously created by this "console programming"[4] procedure.

We shall now flesh out this skeletal description with further details. The keys of the control console are divided into three groups: 24 are used for function storage; 30 to designate operations on these functions; and 11 (the digits 0 through 9, $\oplus$ and $\ominus$) for the input of individual numbers[5]. Throughout we mean by a "function" a set of 101 points, i.e., 202 numbers, represented in the computer as 202 machine words, each word having 26 bits . (The choice of 100 intervals for the description of a function is one example of the arbitrary choices mentioned in the first paragraph. With fewer points one cannot adequately represent very much structure, while if 100 are insufficient one should probably use a different scale, or a different representation.) In addition to the 202 numbers, which all lie between -1 and 1, each function carries two scale factors (to base 2), one for abscissa values (s_x), the other for ordinates (s_y). The actual function values are thus the product of the 101 mantissas y_n, $1 \leq n \leq 101$ and the common scale factor, 2^{s_y}. For convenience, a block of 256 words is assigned to each function, the remaining 52 words being available for other labeling, for functional values (in the sense of Volterra), etc.

Because the computer module (CM) of the RW-400 has only 1024 words, functions are stored on an 80,000 word magnetic drum.

Half of the CM memory is used for two "function registers", called the C and D registers, each having a capacity of 256 words. They play a role for functions quite analogous to that which the accumulator in a computer plays for numbers: functions to be operated upon are loaded into the C and D register and the resulting function is eventually stored back on the drum.

Each of the 24 function storage keys addresses a particular section of the drum but this is of no concern to the user, who may think of the keys themselves as the storage locations. These keys initially carry some neutral labeling (e.g., the letters A through X) but as the user stores functions in them he relabels them (using any convenient nomenclature) to indicate the function stored there.

We can now describe some of the basic operator keys. LOAD and STORE bring any desired function from the drum into the D register or, conversely, store the contents of the D register into any specified function location. For example, pressing the LOAD key and then some function key, say R (or in a shorthand notation we shall employ henceforth, in which each word or symbol corresponds to the name of a particular operator or function key, LOAD R) brings the function stored in key R into the D register. Similarly, STORE F transfers whatever function may be in the D register to location F. In both cases, the words written (on the drum or in the CM memory) replace whatever was previously in that location. (The contents of the cells from which the information was taken are left unchanged so that immediately after LOAD A or STORE A both the D register and the A key contain the same function.)

The operator key J-GEN creates the identity function, $y = x$, $(-1 \leq x \leq 1)$ and puts it in the D register. The arithmetic keys (+, -, ·, ÷) cause the computer to carry out the indicated operation on the ordinates of two designated functions, assuming the abscissaes to be the same. For example, pushing the four keys

LOAD A + G

causes the computer to load whatever function is in location A into the D register; to load the function in location G into the C register; to add the y coordinates of the C and D registers, with differences, if any, in the y scale values properly taken into account; and finally to store the result in the y coordinates of the D register, leaving the x coordinates of the D register unchanged. If the user wishes the sum stored in some function storage location, say P, he then pushes STORE P. Alternatively, he can continue on with a series of arithmetic operations, all of which follow the same pattern. (Once the + button has been pushed, one may add as many functions as desired by simply pushing the + button again. This is true in general; once an operator button has been pushed that operation is continued as long as no other operator buttons have been pushed.)

Individual numbers can be put into the computer in a variety of ways. Since constant functions are sometimes required, we use them to represent also constant numbers, but this is by no means necessary. The procedure is simply: push the LOAD button; then type in the sign, followed by a mantissa less than 1, and any

desired power of 10, positive or negative. (For example, 11.56 is entered as ⊕.1156 ⊕ 02.) A constant function whose value is equal to this number is thereby loaded into the D register (i.e., the x values of the D register range as usual from -1 to +1 and the y values are all equal to the desired constant).

The DISPLAY key allows the user to see a graphical representation of any of the stored functions. DISPLAY A, for example, causes the computer to display on one of the 17 inch CRT scopes, the 101 points stored as functional values in location A, with adjacent points connected by straight line segments. Pressing the A key once more will erase the display curve from the scope (although not of course from the drum location where it is stored). Since only the mantissa values of a function are displayed on the CRT, we sometimes wish to check the scale of the whole function. This is done with the DISPLAY SCALE key, which causes the ordinate scale value, s_y, of the D register to be displayed on the alphanumeric scope. One thus has the capability of carrying out arithmetic and algebra on functions and examining the results graphically whenever desired.

The essential elements of calculus are provided by the Δ and $\sum$ keys. The former simply takes differences of adjacent ordinate values in the D register and leaves the result in the D register, e.g., $(y_{n+1} - y_n)$ replaces y_n, $1 \leq n \leq 100$, with a special treatment at the righthand end point (for example, the first difference computed on the basis of a second or third

order fit to the function values at that end replaces y_{101}). Σ is just a cumulative sum of the ordinate values in the D register with the result left in the D register (0 replaces y_1

and $\left(\sum_{k=1}^{n-1} y_k\right)$ replaces y_n, $\quad 2 \leq n \leq 101$).

From these two keys it is easy to construct approximations of any desired accuracy to the derivative and (indefinite) integral operators.

At this point, the general nature of items A and B above should be clear. One has, in effect, a powerful, and exceedingly fancy, combination hand computer and plotting machine. Any desired function can be readily created in the computer (using power series, asymptotic series, etc.) and one can perform all of the operations of classical analysis upon them. Suppose, for example, one wants the sine of some function, f, which has been loaded into the D register and suppose further f is sufficiently small so that the first two terms of a power series

$$\sin f = f - \frac{f^3}{6} \tag{1}$$

suffice. Select two function keys, F and G, as "working space". The following keys would then be pushed:

STORE F · F F STORE G LOAD ⊖.166667 ⊕ 00 · G + F (2)

If f was initially in the D register, sin f, to the accuracy of Eq. (1), will now be there. In precisely similar fashion one could

obtain a representation of the sine function to any desired number of terms of the power series. However, it is clearly infeasible to go through this sequence of key pushes every time one wants the sine function. It is at this point that feature C, "console programming", comes in; like a giant lever (or a strong bootstrap) it provides an enormous multiplication of the capability available to the user.

Clearly, all that is required is that the computer be able to "remember" and suitably record a sequence of key pushes such as that given in the example above. Moreover, it should then, in some sense, attach this list of key pushes to some previously blank key, which thereby acquires significance. The procedure is simple: we select some key, hitherto blank, which we decide will be the SINE key, and so label it. We then "program" this key using the PROGRAM key in the following way. First push PROGRAM; then push the (hitherto blank) key which will henceforth be the SINE key; then push precisely the buttons listed in (2); finally, at the end, push the PROGRAM button again. The result is that the machine goes through a "dry run", i.e., executes the commands (2) in precisely the same fashion as if we had not pushed the PROGRAM button; examination (e.g., via the display capability) of the result of this dry run immediately provides a first check on the console program just created. In addition, however, the computer constructs a list of these key pushes, termed a "subroutine", and "inserts" it under the SINE key. If at any time in the future we

push the SINE key, the computer will go through precisely this sequence of operations[6]. Furthermore, we can in the same fashion program other keys, using as component keys not only the initial hand programmed ones (such as +, etc.) but also any keys which have been console programmed in the above fashion. These new keys can, in turn, be used as components of other console programs, and so on, to a depth limited only by the storage volume. (The present system allows for 256 such console programs but this could easily be expanded by several factors of 2.) In this way the operator creates his own subroutines, of arbitrary complexity, and pyramids these to achieve whatever computing capability he desires.

At this point it becomes difficult to describe adequately the generality and utility of the resultant system, just as it would be difficult to explain to a college freshman (in less than a few semesters) why algebra or calculus, whose basic principles can after all be rather concisely stated, is so useful. The on-line system has a structure very close to that of mathematics in its open-endedness, its generality and the constructive capability it affords the user. We shall therefore simply use the rest of this section to describe briefly the other hand programmed keys which are initially provided to every user when he begins work, and in the next section illustrate how this capability was used for one particular, fairly illustrative problem. A detailed characterization of all the hand programmed keys is given in Appendix A.

To begin with, the 30 operator keys physically present on the control console are by no means enough to encompass the initial hand programs plus the console programs needed in a typical problem. We therefore employ the concept of "overlays". To each overlay corresponds a set of meanings for the operator keys; changing the overlay changes the significance of all of these keys. The number of overlays is limited only by the size of the large volume storage in the computer system; in our case there are 32 overlays. One key common to every overlay is OVERLAY IN. The user changes overlays by simply pressing OVERLAY IN and then typing in, on the numerical keys, the number of the overlay he wishes to use. The 256 word program (which comprises the overlay from the programming point of view) is thereupon brought from the drum into the computer and all further key pushes will be interpreted by the computer in terms of that overlay until the operator makes a change of overlay. (In addition to the OVERLAY-IN and PROGRAM keys, five others, to be described later - REPEAT, OVERLAY-OUT, DISPLAY-OV-NUMBER, INSERT and DO - are common to all overlays, leaving a total of 24 x 32 = 768 keys in the present system, each of which can have a hand program or a console program.) The existence of multiple overlays modifies the console programming procedure slightly in that we must inform the computer not only which key is to be programmed but also which overlay we want that key to be on. The latter is accomplished by simply typing in the desired overlay number (on the numerical keys) immediately after

pushing the key being programmed: for example, PROGRAM SINE 10 followed by the key pushes (2) and then PROGRAM would attach the subroutine (2) to the indicated key of Overlay 10.

In a similar way, the total number of function storage keys available can be multiplied up from the 24 keys physically present to an extent again limited only by storage space. In the present system, we have 6 "banks" of these function keys, giving a total of 144 function storage locations. This too can be accomplished in many ways; at present, in place of the LOAD and STORE keys described above, we have in fact six LOAD and six STORE keys. Thus, $LOAD_I$ A will load into the D register whatever function is under key A on bank I; $STORE_{VI}$ F will store into key F of bank VI whatever function is in the D register, etc. (We use Roman numerals to label banks, Arabic to label overlays, thus minimizing a possible source of confusion.)

In typical operation, a user begins with an initial complement of hand programs and, working for a period of one to two hours[7], creates and checks (by observing the character of displayed curves, examining individual numerical values, running test cases, etc.) the console programs he needs. When his period of operation is finished, he pushes the SYSTEM DUMP button which stores the entire system (contents of the computers and of the drum) into a designated section of magnetic tape. When he next returns to the machine his system is loaded back from this tape and the computer is in precisely the same state as when he left. In the interim

another user comes to the machine, and loads his system from tape. All of the buttons, save for the initial core of hand programs, will typically be different for two users so that arguments concerning the value, efficiency or desirability of any particular console programming need never arise; each user makes his own choice, i.e., literally creates his own language.

When a user returns to the machine, SYSTEM LOAD (the inverse of SYSTEM DUMP), using his tape, restores the system (computer and drum) to precisely the state in which he left it, so that he can continue on, creating new console programs or, when he has built sufficient capability, attacking his problem. If, in the latter case, he immediately discovers a need to create new console programs or modify existing ones he is free to do so. (To reprogram a key he simply programs it as though it were blank; any previous program is simply buried.)

The SYSTEM LOAD and DUMP buttons are part of a "system" overlay which allows one to control the several components of the RW-400 system: to write out a block of data on (or read a block from) a tape unit, a buffer, the drum, etc.; aside from the tape operations, these are of interest principally to the computer expert rather than to the typical user, so we shall leave further details for the Appendix. (See Section A4, System Control Capabilities.)

The remaining hand programmed keys fall into three groups:

1. Mathematical Operations

These include first of all the arithmetic operations (on functions!) which have been mentioned already but require further comment. There are at least three ways of carrying out the function arithmetic: fixed point (with respect to the entire function), floating point (with respect to the entire function) or floating point for each point of the function. Since each has its own virtues, it is desirable to allow the user a free choice. Thus on Overlay 01 the arithmetic is done in a fixed point fashion. For example, when two functions are added, the y scales are compared and the smaller of the two is made equal to the larger one, the associated mantissa values being decreased (i.e., shifted to the right) enough times so that the functional values (mantissa plus scale) remain unchanged; the mantissa y values are then simply added together. If at some point the sum of these happens to be greater than 1, there will be an overflow. This is readily apparent if the sum is displayed, but it may happen in the course of a console program (unless one has correctly anticipated all scaling aspects of the problem) and can then be a considerable nuisance. Similar comments apply to divide, which will overflow at any points where the mantissa of the numerator exceeds the mantissa of the denominator. A second arithmetic overlay, 02, provides protection against such overflows by first floating and then contracting both

summands before addition. In division, the numerator and denominator are first floated and then the numerator is contracted enough times to prevent overflow at any point, if no more than 12 contractions are required, but no more than 12 are made in any case. An Overlay 03, in which the arithmetic is done on a floating point basis for each individual point of the functions, has recently been incorporated into the system but we have not yet sufficient experience to compare it with the other two. Of the latter, Overlay 02 is generally the more convenient, but in special circumstances (larger range of variation within a single function) can lead to more loss of accuracy than would result from the careful use of Overlay 01.

In addition, we have the following: EXPAND y decreases s_y by 1 and doubles all the mantissas of the D register; CONTRACT y is its converse. (They are needed in conjunction with the arithmetic on Overlay 01 and also allow examination (on the CRT) of small amplitude structure of a curve, display of curves at common scale, etc.) FLOAT MANTISSA does EXPAND y as many times as possible without causing overflow at any point. EVALUATE picks out the value of the function in the D register at the x coordinate closest to any designated value. REFLECT interchanges the x and y coordinates of the D register; SUBSTITUTE puts the y coordinates of the C register in place of the x coordinates of the D register. (Using REFLECT and SUBSTITUTE one can create, from the real function arithmetic hand programs, console programs for complex function arithmetic. Using REFLECT and EVALUATE one can find extrema of

a function.) δ-FUNCTION creates in the D register a function which is 1 at any desired point and zero elsewhere. INTEGRAL-TRANSFORM transforms the function f in the D register, using a kernel K(x, x') previously stored on tape as 101 functions of x', and leaves the result, $\tilde{f}(x) = \int dx'\ K(x, x')\ f(x')$, in the x coordinates of the D register. EXPONENTIAL, SINE and COSINE operate on the function contained in the D register, leaving the result in the D register. LEFT-SHIFT and RIGHT-SHIFT perform the indicated operations on the y coordinates of the D register. RELATIVE INTERPOLATE accepts graphical point inputs (via user-controlled crosshairs on the CRT) and modifies the function in the D register so that it passes through these data points, preserving its initial shape between data points.

2. Aids to Console Programming

These include besides the PROGRAM key already described, also REPEAT, a key which allows any console program keys to be repeated any desired number of times, and two keys (TALLY and COMPARE) which provide the capability for branching within a console program.

3. Display and Output Keys

These provide the capability to display on the alphanumeric scope the number of the overlay currently in the computer; to erase all curves from the CRT; to display (on the alphanumeric scope) the binary scale of the function in the D register, as well as the value of the first point of that function; to input

individual points graphically, using a movable crosshair; to print a hard copy of any desired curve on the flexwriter; to produce English language labels for kernels stored on tape, dumps stored on tape, or curves printed out on the flexwriter; to display curves on either of the two 17 inch CRT's; to use other display formats (dots along, crosses, circles, etc.) as well as the usual display of line segments.

In addition to these, there are Aids for the Hand Programmer involving the convenient input of machine words from the console, the display of sections of memory in machine language, etc.. These are described in the Appendix, together with more detailed specification of all the keys already mentioned.

In a class by itself is the SECOND-COMPUTER key which at first glance seems highly specific to the RW-400 and yet really provides an excellent illustration of how to set up an on-line system for an arbitrary computer. In the RW-400 system, there are two identical computer modules, CM-1 and CM-2. The control console is tied directly to CM-1 and this is the only one used in all of the operations described so far. Suppose however that operator key [K] on Overlay 10 is a rather long program, requiring several minutes or more to run. It is then efficient to use the SECOND-COMPUTER key, as follows: press SECOND-COMPUTER, press [K], and type in the number (in this case 10) of the overlay on which [K] is located. CM-2 then performs this program, taking the subroutines and curves needed from the drum in the same way that CM-1 would do.

While this is going on, however, the user is free to operate in the normal fashion with the control console and CM-1, doing computations; examining, if he likes, intermediate results as they are generated by CM-2; preparing new programs; setting up material for the next case; etc. In the last chapter we will explain how this serves as a model for an on-line system using a conventional, large central computer.

AN ILLUSTRATIVE PROBLEM

As an illustration of the use of the on-line system, we describe briefly one problem we have studied, a linear integral equation for a complex function of a real variable. While a single problem can no more exhibit the power and generality of the on-line system than any one application of, say, calculus, can illustrate the utility of classical analysis, we include it to show the ease with which rather sophisticated mathematical techniques can be employed in a system of this sort.

We present the problem as a mathematical one, essentially suppressing the context of physics from which it arose; outline the mathematical methods used; indicate some of the principal console programs generated to implement these methods; and show a few sample results. We can state, but not easily illustrate, one essential point: the method of solution finally adopted was itself the result of experimentation with the on-line system. Several approaches were tried; some quickly proved themselves unsuitable, but as we learned more about the nature of the solutions, we were able to develop satisfactory methods for obtaining them. Thus, quite apart from questions of mathematical convergence (e.g., of an iteration process), one sees a convergence in a kind of space of mathematical techniques.

A study of the electrostatic wave fluctuations in an electron-ion plasma subjected to an external electric field leads to the following integral equation[8] for the fluctuation electric field, of wave number k, as a function of time:

$$E(t) + \int_0^t dt' \left\{ K_e(t-t') \exp i \left[\phi(t) - \phi(t') \right] + \right.$$

$$\left. \delta\, K\,(t-t') \exp i\delta \left[\phi(t') - \phi(t) \right] \right\} E(t') = I(t), \qquad (3)$$

$$0 \leq t \leq T$$

where

$$K_e(t) = te^{-k^2t^2/4} \qquad K_i(t) = te^{-\delta k^2 t^2/4}$$

$$\phi(t) = \mathcal{E}\, t^2/2 + ut \qquad \delta = 1/1836$$

and $I(t)$ is given. (We have chosen units in which the electron thermal speed $(2T/m)^{1/2}$ and the plasma frequency, $(4\pi ne^2/m)^{1/2}$, are unity.) We shall say no more here about the physics of the problem since this is discussed elsewhere[9], and only sketch the on-line techniques used to solve it.

Using an operator notation for the integral transforms in (3),

$$\underset{\sim}{K}_e \cdot E \equiv \int_0^t dt'\, K_e(t-t')\, E(t'), \text{ etc.} \qquad (4)$$

we can write (3) as

$$E + e^{i\phi} \underset{\sim}{K}_e \cdot (e^{-i\phi} E) + \delta e^{-i\delta\phi} \underset{\sim}{K}_i \cdot e^{i\delta\phi} E) = I \qquad (5)$$

Over the time interval of interest ($T \lesssim 10\pi$) the norm of the first operator in (5) is so large, for $k \leq 1$, that an attempt at direct iteration proved useless. (As can be seen from the soluble special case, $k = \delta = \phi = 0$, this corresponds to computing

sin t by a power series on the interval $0 \leq t \leq T$.) However, the transformation

$$F = e^{-i\phi} E \tag{6}$$

gives an integral equation for F,

$$F + \underset{\sim}{K}_e \cdot F + \delta e^{-i\psi} \underset{\sim}{K}_i (e^{i\psi} F) = J \equiv I e^{i\phi}, \qquad \psi \equiv (1+\delta)\phi \tag{7}$$

which can be solved by iterating only the last of the terms on the left hand side. The equation

$$F + \underset{\sim}{K}_e \cdot F = A \tag{8}$$

has the solution

$$F = A - \underset{\sim}{L}_e \cdot A \tag{9}$$

where $\underset{\sim}{L}_e$ is, like $\underset{\sim}{K}_e$, a translate type integral operator and hence specified by a single function L_e

$$\left[\text{i.e., } \underset{\sim}{L}_e \cdot A \equiv \int_0^t dt' \, L_e(t-t') \, A(t')\right]$$

which must obey an equation like (8) with A replaced by K_e:

$$L_e + \underset{\sim}{K}_e \cdot L_e = K_e \tag{10}$$

Having once found (for given k) the function L_e, we write (7) as

$$F = (1 - \underset{\sim}{L}_e) \cdot \left[J - \delta e^{-i\psi} \underset{\sim}{K}_i \cdot (e^{i\psi} F)\right] \tag{11}$$

and solve it by iteration

$$F_{n+1} = (1 - \underset{\sim}{L}_e) \cdot \left[J - \delta e^{-i\psi} \underset{\sim}{K}_i \cdot (e^{i\psi} F_n)\right] \tag{12}$$

With a reasonable initial guess, e.g., $F_0 = J$, we find that this converges splendidly (three iterations). From F we compute, finally, $E = e^{i\phi}F$.

The first step is to find L_e from (10). For the reasons noted above, straight iteration is ruled out. While (10) can indeed be solved with Laplace transforms, the transform of L_e involves the error function of complex argument[9] and hence is difficult to invert. Instead, we take advantage of the fact that problems which are "adjacent" in a mathematical sense are, within the on-line system, adjacent also in a computational sense. If K_e is replaced by

$$R = N^2 te^{-at}, \tag{13}$$

then the inverse kernel function, L_R, satisfying

$$L_R + \underset{\sim}{R} \cdot L_R = R \tag{14}$$

is simply

$$L_R = Ne^{-at} \sin Nt. \tag{15}$$

We therefore write (10) in the form

$$L_e + \underset{\sim}{R} \cdot L_e = K_e + \underset{\sim}{D} \cdot L_e \tag{16}$$

$$D = R - K \tag{17}$$

and choose N and a so as to make the norm of D small (e.g., $a = k$, $N = 2$), thus permitting an iterative solution,

$$L_e^{(n+1)} = (1 - \underset{\sim}{L_R}) \cdot (K_e + \underset{\sim}{D} \cdot L_e^{(n)}) \tag{18}$$

This converges nicely (three or four iterations) to yield a result which will differ from the exact solution of (10) only in consequence of the approximation inherent in numerical methods. However, we can exploit the linearity of (10) to obtain a more accurate solution as follows. Let L be the result obtained by iterating (18) until it has converged. The error in L is measured by the size of

$$P \equiv K_e - L - \underset{\sim}{K}_e \cdot L \tag{19}$$

and the difference $\eta = L_e - L$ satisfies

$$\eta + \underset{\sim}{K}_e \cdot \eta = P \tag{20}$$

or equivalently

$$\eta = (1 - \underset{\sim}{L}_R) \cdot (P + \underset{\sim}{D} \cdot \eta), \tag{21}$$

i.e., an equation identical with (18) save for the inhomogeneous term. If L is determined from (18) up to a percentage error of order ϵ, we can find η from (21), also with a percentage error of order ϵ, and hence get an approximation, $L + \eta$, to L_e which has an error of order ϵ^2. In a similar fashion we can find a correction to η, and so forth.

We now indicate how the on-line system was used to solve (7) by these methods. To begin with, certain function keys are assigned to the constant parameters of the problem and to the principal independent and dependent variables as shown in Table I. Locations for constant functions which one finds it convenient to have on hand are assigned as the need arises. (The notation in Table I

$\mathcal{E}$	k	u	T
δ	N	a	Δt
t	K_e	K_i	R
L_R	L_e	0	$\frac{1}{2}$
L	L'	L''	L'''
f	$\tilde{f}$	kernel source	working space

Table I

Assignment of function storage spaces (keys) on Bank I for the plasma oscillation problem. Significance of the symbols is given in Eqs. (3) through (23).

is the same as in Eqs. (3) through (23); the meaning of other symbols will be explained below.) Once assigned, the labels on the function keys are used in referring to these keys rather than the neutral ones (A, B, ...X) of the preceding chapter.

Operator keys are then created, using the console programming procedure, some of the principal ones being as follows:

$[t]$ This creates the function $t = T(x+1)/2$ (assuming that the desired value of T has been previously stored in the T key of bank I) and stores it in t on bank I. It also computes Δt and stores it in Δt. As an illustration of console programming, we list the key pushes made in programming this key[10], which we suppose is to be on, say, overlay 10:

PROGRAM $[t]$ 10 OVERLAY-IN 02 J-GEN · 1/2

+ 1/2 · T STORE t Δ STORE Δt PROGRAM (22)

INITIAL SET-UP This simply displays on the alphanumeric scope the names of the constant parameters ($\mathcal{E}$, T, k, δ) and, next to each, the value presently stored for it on bank I. If the user wishes to change any of these he can, of course, do so before running the problem. This program also stores the various constants indicated in Table I and finally pushes the $[t]$ key. Thus one knows that everything is in order for the start of a calculation.

$[K_e]$ This simply computes the kernel function $K_e(t)$ and stores it on bank I.

$[R]$ This computes and stores the function, R(t) defined by (13), using whatever values the user has stored in N and a on bank I.

$[L_R]$ This computes and stores the function $L_R(t)$ defined by (15). It, for example, was programmed by:

PROGRAM L_R 10 OVERLAY-IN 02 LOAD t · N SINE STORE L_R LOAD -1 · t a EXP · N L_R STORE L_R PROGRAM.

(Note that in this we have used the L_R key as a temporary working space to store one factor of the final answer.)

We frequently need to generate, on tape, the 101 functions which comprise one of our translate kernels. To produce this capability, we first program a KERNEL-GENERATE-AUXILIARY (KGA) key as follows:

PROGRAM KGA 10 OVERLAY-IN 02 LOAD KERNEL-SOURCE TAPE-WRITE LEFT-SHIFT STORE KERNEL-SOURCE OVERLAY-IN 10 PROGRAM.

This subroutine takes whatever function is in the kernel-source space on bank I, writes it out on tape, left shifts it,(y_{n+1} replaces y_n) and stores it back in the kernel-source space. We end it by calling in the overlay (10) on which KGA has been programmed in order that it be a repeatable key. The key which will actually produce the kernel on tape is then made by simply repeating the KGA key, i.e., we make a KERNEL-GEN key as follows:

PROGRAM KERNEL-GEN 10 OVERLAY-IN 10 REPEAT KGA 101 PROGRAM.

(Repeat $[K]$ followed by a number, n, causes key $[K]$ to be repeated n times.)

It is clearly now a simple matter to make, for any desired value of k, the various functions and kernels needed for (18).

Since we will be taking many integral transforms, it proves convenient to incorporate the hand-programmed integral transform key (which produces in the x-coordinates of the D register the transform of the function in the y-coordinates of the D register) into a simple console program which will produce in the $\tilde{f}$ space of bank I the transform of whatever function has been stored in f on bank I. We designate this as INT-TRANS and, using it, program a new key, ITERATE-L_e which does one pass of (18) as follows. Assume that the kernel D has been stored on tape and, following it, the kernel L_R; that the tape is positioned to the beginning of the D kernel; and that some initial guess, or the result of a previous iteration is in L_e on bank I. We then do

PROGRAM ITERATE-L_e 10 OVERLAY-IN 02 LOAD L_e STORE f OVERLAY-IN 10
INT-TRANS OVERLAY-IN 02 LOAD $\tilde{f} + K_e$ STORE f OVERLAY-IN 10
INT-TRANS OVERLAY-IN 02 LOAD_I f - $\tilde{f}$ STORE_I L_e REWIND-TAPE (23)
OVERLAY-IN 10 PROGRAM

Although the program (23) is adequate, we add to it certain display and storage features which increase the convenience of operation. That is, after checking (with simple examples, special cases, etc.) the correctness of (23), we program another key, $\lceil L_e \rfloor$ which first pushes the ITERATE-L_e key (23) and then goes on to store the resultant L_e in key L, having first moved the contents of L'' into L''', L' into L'', and L into L'. Thus as the new key $\lceil L_e \rfloor$ is repeated, we are able to examine the results of the most recent four passes (and could, of course, save even earlier ones if desired, by using one of the other banks). In

addition, the new key erases the scope and then displays on it the contents of L' (dotted) and the contents of L (usual dot-plus line display). Thus, each time the key is pushed the user sees, as soon as the pass has been completed, both the new result and, for comparison, the previous one, so that he can judge the convergence characteristics of the iteration process (18). (One could as well display the ratio or difference of the old and new curves, etc.)

The reader who has followed the details of the last few paragraphs can supply those omitted from what follows. Having credited the programs needed for (18), we can use them also for (21) and, by repeating the correction process, obtain a very accurate L_e. From it we make an $\underset{\sim}{L}_e$ kernel (using KERNEL-GEN) and also the $\underset{\sim}{K}_i$ kernel and are then in a position to solve (12), i.e., to make a key which will do one iteration of (12). The only new complication lies in the complex character of F, but this causes no real difficulty. We simply write out on tape two copies of $\underset{\sim}{K}_i$ kernel, followed by two of $\underset{\sim}{L}_e$. The ITERATE-F key then multiplies F by $e^{i\psi}$, uses the INT-TRANS key to transform the real part of the product with $\underset{\sim}{K}_i$, stores that in some working-space function key, transforms the imaginary part with $\underset{\sim}{K}_i$, combines the results into a single complex function, multiplies this by $\delta e^{-i\psi}$, subtracts that from J, transforms the real part with $\underset{\sim}{L}_e$, then the imaginary part, etc.

This requires, of course, that one create also keys which produce $e^{\pm i\psi}$ (used only after a change of $\mathcal{E}$ or u) and keys which give complex arithmetic capability. Using REFLECT and SUBSTITUTE, one can easily program, for example, a COMPOSE key (which, given two real functions, $f_R(t)$ and $f_I(t)$, in two standard locations composes them into a single complex function $f=(f_R, f_I)$ with the parameter t eliminated) and a DECOMPOSE key which is its inverse. For example, designate 3 keys as f_R, f_I, f and then make COMPOSE by

PROGRAM COMPOSE 10 OVERLAY-IN 02 · f_R LOAD f_I

SUBSTITUTE STORE f PROGRAM

(We use · f_R as a way of loading f_R into the C register.)
Similarly we have

PROGRAM DECOMPOSE 10 OVERLAY-IN 02 LOAD 0 + f STORE f_I

LOAD f REFLECT STORE f_R LOAD 0 + f_R STORE f_R PROGRAM

(LOAD 0 + f_R STORE f_R is just a way of restoring standard x coordinates to f_R so that it will look normal when displayed.) From these, it is then a simple matter to make the keys for complex function arithmetic.

Using the ITERATE-F key, and the L_e kernels generated by the procedure described above, it is an easy matter to obtain solutions of (3) for a variety of values of the parameters k, u and $\mathcal{E}$. As we see from Figure 2 the kernel L_e, which is a sine wave for k = 0, is increasingly damped with increasing k. (Having computed L_e one can compare it with the result obtained by keeping only the least damped pole of the Laplace transform

of L_e when inverting the latter by contour integration[9]; as shown in Figure 2, the agreement is fairly good save near $t = 0$.) Understanding the somewhat exotic curves which result is assisted by comparing them with analytically soluble problems, e.g., those obtained by omitting the $\underset{\sim}{K}_1$ operator in Eq. (3) or setting $\mathcal{E}$ equal to 0. Some typical results are shown in Figure 3 through 8 for the case $I(t) = 1$. An analysis of the results and a discussion of their significance is given elsewhere[9].

DISCUSSION

The on-line system we have described is specific both as regards the computer used and the area of mathematics emphasized (classical analysis), the choice of the computer being a consequence of its availability, while the selection of problem areas was dictated by the research interests of the participants. On the basis of the experience gained in the design and operation (since July, 1962) of this particular system, the extension of these on-line techniques to other computers and to other areas of application appears rather straightforward.

We first outline a system which would be suitable for a conventional, large central computer and which would permit an operation identical, from the user's point of view, with that we have described here. Besides the central computer itself, one needs a large volume storage element, such as a disc file, and a small satellite computer, one with a memory of the order of 8,000 to 10,000 words and a 5 to 10 microsecond cycle time. As input/output equipment, the satellite computer would have two electric typewriters, whose keys take the place of the control console keys, and two CRT display scopes, each capable of displaying of the order of 1000 points furnished by the satellite computer, and connecting pairs of these with line segments when desired.

The operation parallels that of the present system when the SECOND COMPUTER key is used, the control console and the

first computer module of the RW-400 being replaced by the satellite computer plus its input/output equipment; the drum (where curves and subroutines are stored) being replaced by the disc file; and the second RW-400 computer module being replaced by the (larger and far more rapid) central computer. The satellite computer is used to compose and check console programs and for all trivial computing: examination and comparison of curves, formation of ratios and differences, simple test cases, etc. Only when the user has progressed to a point of having a substantial computational task which he wishes performed, is the central computer involved. He simply presses the CENTRAL COMPUTER key and then any of the keys he has previously console programmed (with the satellite computer). The CENTRAL COMPUTER is not interrupted, but when it finishes the task on which it is presently engaged and returns to its own control system for a next assignment, it is directed to take from the disc file the satellite's request, carry it out and return the results to the disc file. The central computer then proceeds to other work while the user examines the results, perhaps modifies his program or decides on a next case. The central computer is brought in only for significant computational tasks and never waits for the user. The user may occasionally have to wait a short time for the central computer[11], but since the tasks he gives it are only those requiring a considerable amount of computation, this is not unreasonable.

In a sense, the satellite computer functions as a kind of informational impedance matching device between the man and the large central computer. Taken by themselves, these are mismatched with respect to both operations per second and dollars per hour. However, the satellite computer is economically matched to the man (i.e., rents for a figure comparable to his salary) and at the same time is sufficiently well matched to the central computer in terms of data transfer so as to be consistent with the economic constraints concerning the latter's use. Of course, many variants of this basic scheme are possible, some more suited to a particular computer center than others. Because our experience indicates that it is convenient to have of the order of 50 to 60 keys, we specify two typewriter keyboards, but in principle the necessary control capability, including that required for console programming, could be provided with far fewer keys. (Ten, representing the digits 0 through 9, plus one more ___ to, so to speak, change overlays ___ is probably the minimum required to provide a reasonable degree of operational comfort.) Because the simultaneous display of many curves on a single CRT scope gets quite confusing, it is very convenient to have two scopes, particularly for problems where one wishes to examine a mapping from one plane to another. One scope might be sacrificed, but we would argue strongly against the elimination of both, having found the rapid feedback of information in graphical form to be a tremendous asset in studying the structure of a problem and of the tools one creates, in the

form of console programs, to solve it, not to mention its value in checking and trouble shooting the latter. In any case, the effective implementation of such a system will share some of the problems inherent in any time-sharing arrangement[12].

We consider briefly the generalization of on-line techniques to other problem areas. The emphasis on functional orientation is particularly important for non-local problems but it is straightforward to include also a capability for dealing with individual numbers, something which would be useful, for instance, in solving differential or difference equations. This requires only an overlay (04, say) which interprets the function keys as single numbers, i.e., allows the function keys to address individual cells of the computer memory rather than function storage blocks on the drum, the arithmetic on Overlay 04 being just the conventional single number arithmetic of the computer itself[13]. Console programming would allow the composition of operations on this overlay in the usual fashion.

The extension to areas of mathematics other than classical analysis also seems feasible. To handle matrix problems, for example, one would replace the functional format by one in which matrices could be stored in the "function keys", with the basic operators being now those of matrix arithmetic rather than function arithmetic. For an algebra machine or a logic machine, the basic, hand programmed keys would correspond to the fundamental operations of these disciplines, but at present this

is still somewhat speculative. In each case, the general organizational scheme of the present system, including the control and console programming aspects, would be preserved, and only those parts (actually a small fraction) of the programming associated with the functional orientation and with the graphical displays would be altered. While this is true also of other areas of computer application (e.g., those involving information processing rather than mathematics), the identification of the basic operations from which all others can be compounded by console programming appears far more difficult, there being no analog for the experience accumulated in the physical and mathematical sciences during the past 300 years.

We turn now to the system as it presently exists. We note that much of its power derives from the fact that substitution can be carried out at several levels: substitution of numbers, of functions and of programs. Substitution of different parameter values is carried out by simply writing the console programs with the parameters in question represented by certain function keys; one then has only to insert the desired constant functions into these keys before running a program. The capability for functional substitution is provided by the REFLECT and SUBSTITUTE keys. Given two functions u(x) in the C register and v(x) in the D register, the SUBSTITUTE key produces in the D register the function v(u). When displayed,

this will be a curve in the (u,v) plane. Conversely, given such a curve, v(u), we can (as illustrated in the preceding chapter) easily obtain the parameterized curves u(x) and v(x).

Most important perhaps is the possibility of substitution at the program level. Suppose that we wish to make a change in a console programmed key [K] which is one of the components of another key [L], [L] in turn being a component of still a third key, [M]. If we wish to substitute a new console program for the one presently under [K], we simply program [K] in the same way as one does with a blank key; the program formerly associated with [K] will be buried. Alternatively, we may find that the console program associated with [K] is so basic and takes so long to run that it should be replaced with a hand program. (For this replacement the "user" must get the help of a "computer expert".) In either case, the program associated with key [M] will run precisely as before, save for the desired modification in [K], for the program in [M] recognizes [K] only as a key push, regardless of the significance of the subroutine it calls in.

One thus has the ability to manipulate console programs with approximately the same freedom as one juggles the mathematical operations which they represent, a feature not present in conventional programming languages. As a result, problems which are adjacent in the mathematical sense become so computationally as well; one can proceed from the simple to the more complicated, always building upon the results of what one has learned, without

the necessity for redoing all of the programming as new pieces are added or old ones are modified.

While we have characterized the "user" of the on-line system as a scientist unversed in conventional programming methods, it is clear that the creation of console programs involves the very essence of programming, albeit with most of the drudgery eliminated, and that "users" would benefit from the advice of someone familiar with programming. Indeed, operation of the on-line system involves two activities which at first sight appear separable: a) the creation of those console programs needed for a problem; and b) the use of these in its solution. Why not let someone we may call a "console programmer" (since his qualifications will differ somewhat from those appropriate to programmers in the standard meaning of the word) take care of a), the "user" being involved only with b)? The point is just that a) and b) are in fact strongly coupled; as soon as one starts to use b) he typically finds that some changes or additions are needed and he must revert back to a). If the user does not actually do a) himself, he must certainly work very closely with the "console programmer" who does, in order that he thoroughly understand the significance of the keys in his system. Moreover, unless the user is familiar, from hand computation or other experience, with numerical methods, a mathematician skilled in such matters had better be available for consultation; for the privilege of having direct access to a

computer, the user must pay the price of being exposed also to questions of scaling, error accumulation and all the other technical problems which are of course involved in any computational work but which are seen dimly, if at all, by the user when, as in most conventional organizations, he is insulated from the computer by several layers of intermediaries.

In conclusion, we should emphasize that there will be many computer applications for which these on-line techniques will be of little or no value. If one thoroughly understands the structure of a problem and knows a method of solution which is certain to work, then the experimentation and feedback characteristic of the on-line system are unnecessary. Indeed, such problems are handled very nicely by computer centers as presently constituted. It appears, however, that for problems whose structure is not clear, either a priori or on the basis of previous experience, and for which successful solution techniques need to be developed, an on-line system which allows the technical intuition of the user to play a central role in the solution process can be of considerable value. In this system, the user has a direct and convenient access to the computer, a fast response for computations which are essentially trivial, and a graphical representation of information where appropriate. He can build programs consisting of his own constructs within his own field, combine these in any desired way, and, if appropriate, make a

trial-and-error study of the various features of his problem. Indeed, the interplay of the structural elements is often more important than the solution itself in terms of the information desired in a research problem. Since he has control over the transformations, operators and other mathematical objects involved in his problem, he is able to get hold of the pieces and study the ingredients from the point of view of validity as well as from the point of view of structure. When he has found successful methods, he can combine these into an operating program without the necessity of reprogramming. Finally, from the bulk data available after solving a problem, he is able to select only that which he really desires, either as hard copy, numerical output or in the form of pictures of curves displayed on the CRT.

ACKNOWLEDGEMENTS

We are greatly indebted to the Data Processing Laboratory at Rome Air Development Center for support of this work and for the AN/FSQ-27 portion of the equipment; to Dr. Robert Huff, George Boyd and Robert Bolman for invaluable assistance in the programming tasks; and to Professor R. P. Feynman for suggestions concerning the possible extension to an algebra machine. We acknowledge with special thanks the efforts of Professor J. R. Schrieffer, Professor Karl Menger, Professor H. W. Wyld, Jr., Professor K. A. Johnson, Fred Dion and Martin Schultz who spent much of the summer of 1962 as cooperative guinea pigs, using the on-line system

for research problems in their own fields, notwithstanding its then somewhat raw and rough-edged character, thus contributing greatly to its present state of development and to our understanding of the user's needs and desires in an on-line system.

APPENDIX - DESCRIPTION OF BASIC SUBROUTINES FOR AN ON-LINE SYSTEM

The initial hand programmed keys which at present comprise the basic system from which every user starts may be divided into five categories, save for the especially significant SECOND COMPUTER key, which stands by itself:

1. Mathematical operations.

2. Capabilities which provide assistance in the creation of console programs.

3. Programs having to do with displays or with other input/output aspects.

4. Operations involved in management of the computer system.

5. Conveniences for the computer expert who may be concerned with hand programming.

Many other items could be added to the list which follows and some of those given here could be omitted. While our set is neither exhaustive nor minimal, it has proved to be extremely convenient. Names of operator keys are in capital letters; headings not capitalized refer to groups of keys so closely related that to save space we have not listed them separately. In the description of keys we shall, in the interest of simplicity, ignore the multiplicity of function banks[14].

A1. Mathematical Keys

LOAD $LOAD_I$ A brings the function in key A of bank I into the D register. (It also remains in A on bank I.) Similarly for $LOAD_{II}$, ---$LOAD_{VI}$.

STORE $STORE_I$ puts the function in the D register into key A on bank I, leaving it also in the D register. Similarly for $STORE_{II}$ ---$STORE_{VI}$.

FLOAT-MANTISSA The y values of the D register are shifted left as many times as possible without causing any one of them to overflow, and the scale value s_y is adjusted appropriately. The x values of the D register are unchanged.

+ On Overlay 01, + A puts the function stored in key A into the C register, and then compares the y scales of the C and D registers. If the scales are equal, the y values are added together and left in the D register. If the y scales are unequal, the function with the smaller scale is contracted until the scales are equal and the addition is then performed. The x coordinates of the D register are unchanged. If the same operation is performed on Overlay 02, both functions are floated, each is contracted once, and the addition is then carried out as on Overlay 01.

· On Overlay 01, · A loads the function stored in key A into the C register, multiplies its y values by those of the D register, adds the scales, and leaves the result in the D register. In the same operation on Overlay 02, each function is first floated.

$-$ Subtraction is performed in the same fashion as addition.

$\div$ On Overlay 01, $\div$ B loads the function stored in key B into the C register, divides the y values in the D register by those in the C register, subtracts the scale values, and leaves the result in the D register. When the same operation is performed on Overlay 02, each function is first floated and the numerator is then contracted enough times to prevent overflow at any point unless this requires more than 12 contractions, in which case the numerator is simply contracted 12 times.

$\sqrt{\ }$ $\sqrt{\ }$ takes the square root of the function stored in the D register and leaves the result in the D register.

LEFT-SHIFT The y values of the D register are shifted one place to the left: y_{n+1} replaces y_n, $1 \leq n \leq 100$, and y_{101} is left unchanged.

RIGHT-SHIFT The y values of the D register are shifted one place to the right: y_{n-1} replaces y_n, $2 \leq n \leq 101$, and y_1 is left unchanged.

EVALUATE This allows one to evaluate the function in the D register at the value of the x coordinate nearest to any selected number, previously stored as a constant function in one of the function storage spaces. The operation is as follows: EVALUATE B loads the function stored in B into the C register and subtracts the y coordinates of the C register from the x coordinates of the

D register. The y value in the D register corresponding to the smallest of these differences is selected and all y coordinates of the D register are set equal to that value.

EXPAND y The y values of the D register are multiplied by two (shifted one place to the left) and the scale value s_y is reduced by 1.

CONTRACT y The y values of the D register are multiplied by 1/2 (shifted right one place) and the scale value s_y is increased by 1.

Both EXPAND and CONTRACT leave the numerical value of the function invariant since a change in scale appropriately compensates the alteration in mantissa values. However, since only the latter are displayed, the appearance of the function on the CRT is altered. One can use EXPAND to examine in detail the small amplitude structure of a curve, letting the other parts overflow being careful, of course, to retain the original representation of the function in another storage spot ; it thus complements FLOAT-MANTISSA. If one uses Overlay 01 (fixed point arithmetic) CONTRACT is necessary in order to avoid overflow in addition, subtraction. Finally, both EXPAND and CONTRACT are useful in bringing curves to a common scale for visual comparison.

δ FUNCTION This creates a Kronecker-δ type function, i.e., one which has the value 1 at one point and zero everywhere else. To create the function $\delta_a = \delta(x - a)$, load the number a into the D register then push the δ function button. The desired function

then appears in the D register, i.e., all of the y coordinates in the D register are made 0, save the one corresponding to the value of x nearest to (or equal to) a, and it is set equal to 1.

EXPONENTIAL The exponential of the function in the D register is computed and the result is left in the D register.

SINE-COSINE The sine and cosine of the function in the D register are computed. The sine is put in place of the y values of the D register; the cosine is put in place of the x values of the D register. (This may alternatively be considered as a complex exponential e^{if}, where f is the function in the D register.)

J-GEN The identity function, $y = x$, $-1 \leq x \leq 1$ is put in the D register.

REFLECT The x and y values of the D register are interchanged, as are also the scale values, s_x and s_y.

SUBSTITUTE The y coordinates of the C register replace the x coordinates of the D register and similarly for the scale values. This permits, for instance, the cross plotting of two dependent variables which are functions of the same independent variable. Together with REFLECT, it allows one easily to create console programs for complex-valued functions of complex arguments using only real function hand programs (i.e., those described in the foregoing part of this section).

$\underline{\Delta}$ This is a finite difference operation on the function contained in the $\underline{D}$ register: $y_{n+1} - y_n$ replaces y_n for $1 \leq n \leq 100$, while y_{101} is computed by fitting a cubic to the values y_{98}, y_{99}, y_{100} and y_{101}.

$\underline{\sum}$ This is a running sum performed on the function contained in the $\underline{D}$ register: $\sum_{i=1}^{n-1} y_i$ replaces y_n, $2 \leq n \leq 101$ and 0 replaces y_1.

From these two keys one can, using console programming, construct "differentiate" or "integrate" keys of any desired accuracy. Thus, the identity

$$\frac{d}{dx} = \frac{1}{\Delta x} \ln(1+\Delta) = \frac{1}{\Delta x}\left[\Delta - \frac{\Delta^2}{2} + \frac{\Delta^3}{3} + \ldots .\right]$$

shows one way of making the derivative, while its inverse provides an indefinite integral in terms of $\sum$. Of course, one simply uses the EVALUATE operation to get a definite integral.

INTEGRAL TRANSFORM The integral transform of the function

$$\tilde{f}(x) = \int_a^b dx' \, K(x, x') \, f(x')$$

f stored in the D register is computed, using a kernel K(x, x') which has been stored out on magnetic tape in the form of 101 functions of x', one for each value of x. Assuming that the tape has been correctly positioned and that f is in the D register, we simply push the INTEGRAL TRANSFORM key. The first of the 101 functions, i.e., K(a, x'), is then read into the C register and multiplied by the function in the D register. The definite integral is computed and the resulting number is stored in the first x coordinate of the D register. The next function, i.e., K(a + ϵ, x'), ϵ = (b - a)/100, is then read from tape into the C register and the process repeated, the result being stored as the second x value of the D register. At the completion of the operation, which requires 7 seconds, f is still contained in the y coordinates and $\tilde{f}$ is in the x coordinates of the D register. To transform the latter into standard form one could, for example: REFLECT STORE A, LOAD 0 + A (the addition to 0 being one means of restoring the x coordinates of the D register to the canonical form used for the displays).

RELATIVE-INTERPOLATE This uses individual data points, put in with the graphical input techniques described below (Section A3), to modify the function, say f, which has been loaded into the D register. If P_a and P_b are two of the data points, the

program first finds the two points, $\bar{P}_a$ and $\bar{P}_b$, of f whose x coordinates match those of P_a and P_b. If L is the straight line P_aP_b and $\bar{L}$ is the line $\bar{P}_a\bar{P}_b$, the function f is replaced by $(f - \bar{L} + L)$ for $x_a \leq x \leq x_b$. The function is left unchanged between x = -1 and the smallest of the x_a. (Before f is loaded into the D register, it should be displayed on the CRT scope to serve as a guide for placing the data points on the screen with the crosshair.)

A2. Aids to Console Programming

PROGRAM Press PROGRAM; then press the key to which the program to be written is to be attached; then type in the overlay number on which that key is to be located; then press the keys which will make up the desired program; at the end press PROGRAM again.

REPEAT Press REPEAT; then press any repeatable key (i.e., either a hand programmed key, or a console programmed key whose program ends on the same overlay on which the key itself is located); then type in on the numerical keys the number of times the operation is to be repeated. This repeat operation can, of course, itself be incorporated into a console program.

TALLY This is used only within a console program and is one of two capabilities for program branching. TALLY must be imbedded in some console programmed subroutine; that is to say, it must be one of the series of key pushes which make up some console programmed key. When, in the running of that subroutine,

the computer comes to the point where the TALLY key was pushed, it checks the scale value s_y, of the D register. If s_y is positive the computer reduces s_y by 1 and proceeds to the next key in the subroutine; if the scale is 0 or negative, it jumps to the end of this particular subroutine.

COMPARE This operates in the same fashion as TALLY but uses as its criterion the sign of the first y value, y_1, of the D register. If this is positive, the computer continues to the next key push; otherwise, it jumps to the end of the subroutine in which COMPARE is imbedded.

(These keys make possible the incorporation of standard programming techniques - loops, tallys, etc. - at the console programming level.)

A3. Display and Output Keys

DISPLAY OVERLAY NUMBER The number of the overlay currently in the computer is displayed on the alphanumeric scope.

ERASE All curves are erased from the CRT.

DISPLAY DISPLAY A causes the function stored in location A to be displayed; pushing A again erases that curve from the scope. Subsequently pushing other keys will cause the curves stored in them to be displayed also, until some other operator key is pressed.

DISPLAY VALUE AND SCALE, BINARY The mantissa of the first y value of the D register and the y scale, s_y, of the D register are displayed on the alphanumeric scope.

DISPLAY VALUE, DECIMAL If y_1 is the mantissa of the first y value in the D register and s_y the scale of the function, the number $y_1 \cdot 2^{s_y}$ is displayed as a decimal mantissa times a power of 10.

GRAPHICAL INPUT Press POINT-INPUT; then DISPLAY-CROSSHAIR. A crosshair, whose position can be controlled by a lever, is displayed on the screen. After positioning it at any desired point, push TRANSMIT-CROSSHAIR-COORDINATE. The x and y coordinates of the selected point are then transmitted to the computer and a small crosshair symbol is displayed on the scope at that point. The points thus put in are accumulated and can be used in conjunction with the RELATIVE INTERPOLATION key described in Section A1.

PRINT Any curve loaded into the D register will be printed out on the flexwriter in conventional format, i.e., the x and y values will be listed in decimal form.

LABEL After this is pushed, the function keys serve as typewriter keys and can be used to compose any desired alphanumeric message. (Each letter or number is displayed on the alphanumeric scope as it is typed.) This is useful for generating a label to go with a kernel stored on tape, a message which is to be written on tape along with a system dump, or an identifying legend to accompany a hard copy curve when the PRINT key is used.

LEFT SCOPE A word in the display routine is set so that any curve subsequently displayed with the usual DISPLAY key will appear on the lefthand 17 inch CRT.

RIGHT SCOPE A word in the display routine is set so that any curve subsequently displayed with the standard DISPLAY key will appear on the righthand scope.

Alternative Display Formats There are several keys which allow the capability of displaying curves in other than the usual format. Ordinarily 100 straight line segments connecting the 101 points are displayed. However, one can instead display only the 101 dots with no connecting line, or other symbols such as crosses, circles, etc.

A4. System Control Capabilities

The keys in this group allow for convenient management of the entire computer system. Only the first few are needed by the typical user; those with an asterisk are used only by the computer expert and may be disregarded by readers not in that category.

SYSTEM LOAD An entire system - overlays, curves, subroutines, etc. - is loaded from tape into the computer system. Whenever a user starts a period on the machine, he has his tape put on the tape unit and pushes SYSTEM LOAD, thereby putting the entire computer system into the same state it was when he last used it.

SYSTEM DUMP This is the inverse operation to SYSTEM LOAD, and is used at the end of each user's run.

TAPE READ This reada a block of 512 words from magnetic tape into the C and D registers of the computer.

TAPE WRITE This writes contents of the C and D registers as a 512 word block on magnetic tape.

Tape Manipulation There are keys for erasing a block on tape, skipping a block on tape, skipping to an end of file, writing an end of file; rewinding the tape, etc.

DRUM READ* DRUM READ nm reads into the C and D registers, from the drum, the computer program corresponding to Overlay nm. This allows the computer expert to examine or modify the actual machine words comprising the hand programs of this overlay.

DRUM WRITE* This is the inverse of the DRUM READ operation and is used to replace an overlay on the drum after it has been examined or modified.

SUBROUTINE READ* SUBROUTINE READ nml stores the automatic subroutine with identification number nml into the C register where it can be examined or modified by a computer expert.

SUBROUTINE WRITE* This is the inverse of SUBROUTINE READ.

A5. Aids for the Hand Programmer

All of these keys are for the computer expert alone and, like the last items in Section A4, should be ignored by readers who are not in this class. Their descriptions are included here only for completeness.

OVERLAY OUT OVERLAY OUT nm stores the overlay now in the computer onto the drum in the appropriate place. This is necessary after hand program alterations have been made in an overlay.

INSERT This is a convenient method of inserting short hand programs into the system. Press INSERT, then a CM address (4 digits on the numerical keys), then ENTER, then a machine word (10 digits on the numerical keys), then ENTER. Any number of additional machine words can now be typed in (each followed by ENTER); they will be stored in sequence in the locations following the first one. In addition, each word (and the address of the first one) is displayed on the alphanumeric scope.

DO This is simply a convenient means of causing the computer to execute any single instruction on command from the control console. Press DO, then type in on the numerical keys any machine command (again, 10 digits for this particular computer) followed by ENTER.

DISPLAY OF MEMORY CONTENTS This is a convenient means of examining the contents of any portion of the CM memory. Push DISPLAY MEMORY, then type in the four digits specifying an address in the CM. That address and the (10 digit) machine word it contains are then displayed on the alphanumeric scope. If a number n is now entered on the numerical keys, the next n words of the CM memory will be displayed, an operation which can be repeated as often as desired. DISPLAY MEMORY need be pushed again only if one wishes to examine a non-contiguous portion of the memory. The incorporation of this capability into suitable console programs provides the computer expert with a convenient means of dynamically debugging a hand program.

A6. The SECOND COMPUTER Key

In a class by itself is the SECOND COMPUTER key (although in a logical sense it probably could be included in Section A4). The RW-400 system has two identical computer modules, CM-1 and CM-2. The control console normally communicates directly with CM-1, and CM-2 is not used. However, pressing the second computer key, then any previously programmed key [K], followed by the number of the overlay on which [K] is located, causes the SECOND COMPUTER to carry out whatever program is associated with key [K], taking from the drum the subroutines, curves, and other information needed in doing this. While this is going on, the user at the control console is free to use CM-1 for any of the normal operations, e.g., to examine the results being generated by CM-2, to prepare for the next case to be run, to create new console programs, etc.

REFERENCES AND NOTES

1. J. C. R. Licklider and W. E. Clark, "On-Line Man-Computer Communication", Proc. of Spring Joint Computer Conference (May 1962). The first two pages of this contain a cogent statement of the motivation for an on-line system and of some general principles to be observed in constructing one.

2. Glen J. Culler and Robert W. Huff, "Solution of Non-Linear Integral Equations Using On-Line Computer Control", Proc. W. J. C. C., May, 1962.

   Glen J. Culler, Burton D. Fried, Robert W. Huff and J. Robert Schrieffer, "Solution of the Gap Equation for a Superconductor", Phys. Rev. Letters 8 399 (1962).

   Glen J. Culler, Burton D. Fried, Robert W. Huff and J. Robert Schrieffer, "Use of On-Line Computing in the Solution of Scientific Problems" (RW Research Laboratory Report, April, 1962, unpublished).

3. We use the term "hand program" to denote a computer program of the classical sort, i.e., a list of machine words, in contrast to a "console program", which, as described below, is essentially a list of key pushes.

4. The term "automatic programming" might be better, but unfortunately it already has a different significance.

5. The numerical keys include also ENTER, which must be pushed at the end of any sequence of numbers. When we give below illustrative lists of key pushes we shall generally omit ENTER, although in actual operation it must always be included.

6. This example is only illustrative, since in an operating system it is more sensible to make SINE a hand programmed key. In the early stages of development of the present system, however, we actually used, for SINE and COSINE, console programs based on 10 terms of the power series for functions with scale $s_y \leq 1$ and repeated use of the double angle formulas when $s_y > 1$.

7. Experience shows that operation by one user for less than one hour or more than two tends to be inefficient.

8. B. D. Fried, M. Gell-mann, J. D. Jackson and H. W. Wyld, Longitudinal Plasma Oscillations in an Electric Field", J. Nuclear Energy; Part C 1 190 (1960).

9. Glen J. Culler and Burton D. Fried, "Plasma Oscillations in an External Electric Field", (to be published).

10. To avoid confusion between operator keys and function keys having the same label, e.g., t, we shall, in writing lists of key pushes like (22), use [ ] for any operator key whose label coincides with that assigned to some function key. Of course, in actually using the console no confusion arises since the operator keys are physically distinct (and separated) from the function keys. Also, since all functions involved in our examples are on the same bank, we omit the bank subscripts, writing simply LOAD and STORE in place of $LOAD_I$ and $STORE_I$.

11. We assume the on-line work to occur during a period of the day when the central computer is being used only for short problems (maximum of a few minutes running time).

12. See, e.g., C. Strachey, "Time Sharing in Large, Fast Computers", Proc. of International Conference on Information Processing, UNESCO, p. 336 (June, 1959), and A. L. Leiner, W. A. Notz, J. L. Smith and R. B. Marimont, "Concurrently Operating Computer Systems", loc. cit., p. 353.

13. Although "complete" in the sense of being very useful for a wide range of problems, the present system is still growing in the sense that this overlay and some of the other extension described in this chapter will be incorporated in the near future.

14. The only complication which can arise concerns operations with functions stored on two different banks. Thus, in place of $Load_I$ A + B which we would use to add two functions stored on I, we use $Load_I$ A $Load_{III}$ + R to add functions on banks I and III; here the $Load_{III}$ key functions only as a bank indicator.

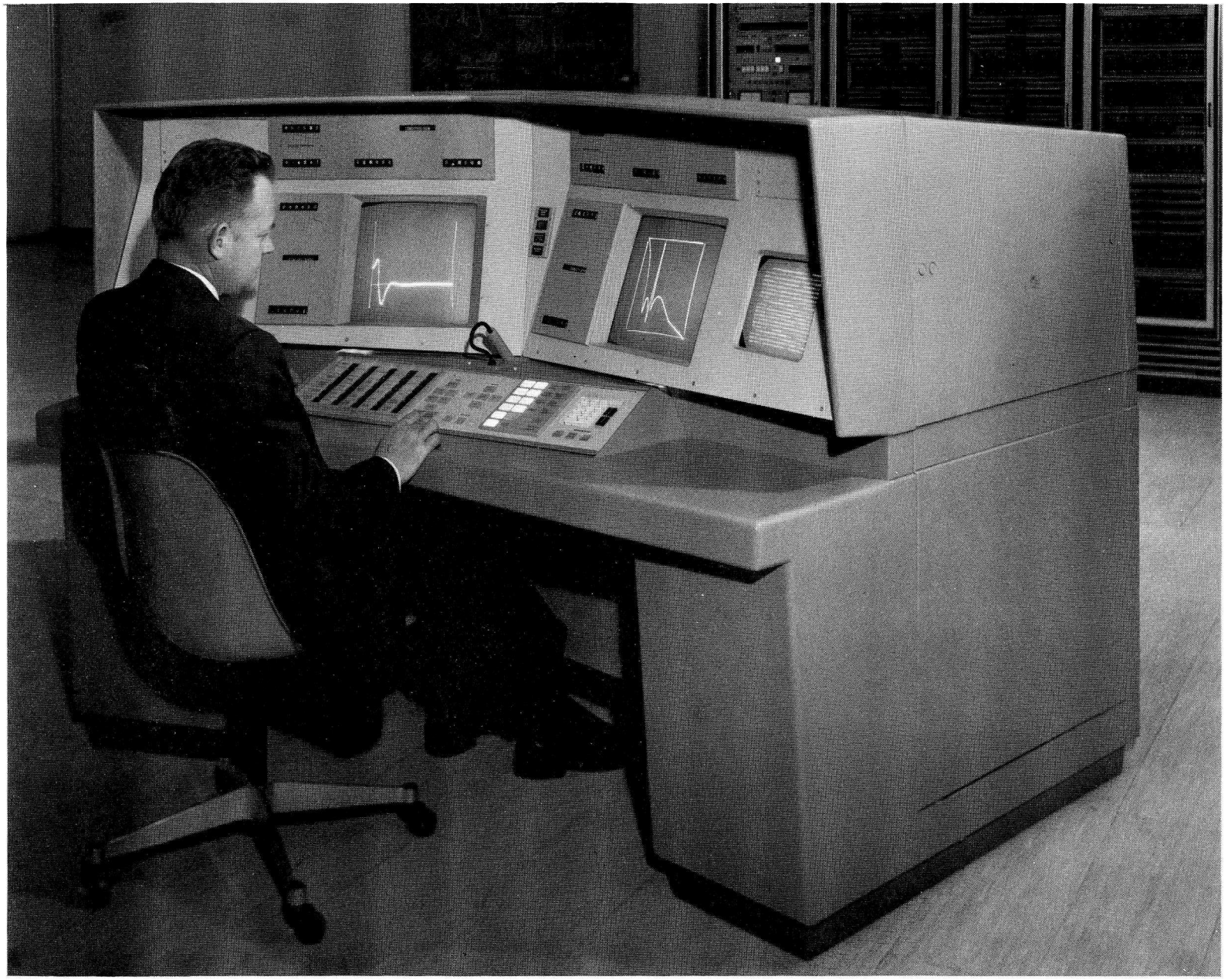

Figure 1. Control console used in the On-Line Computer Center.

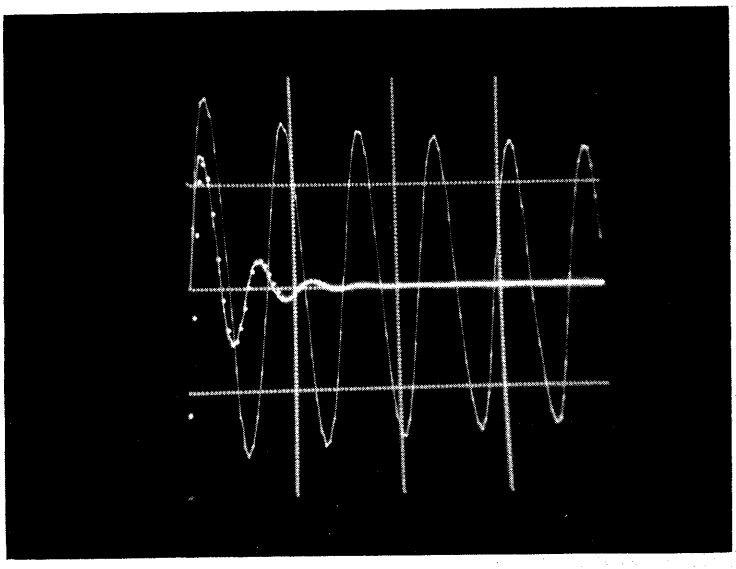

Figure 2. The kernel function L_e, defined by (10), as obtained by iterating (18), correcting the result using (21), etc. The slightly damped curve is for k = 0.4; the strongly damped one is for k = 1.0. A dotted curve shows the result for k = 1 obtained by retaining only the two least damped poles in the Laplace transform of L_e when inverting the transform by contour integration. Grid lines: y = 0, $\pm$ 1/2; t = 7.5, 15, 22.5.

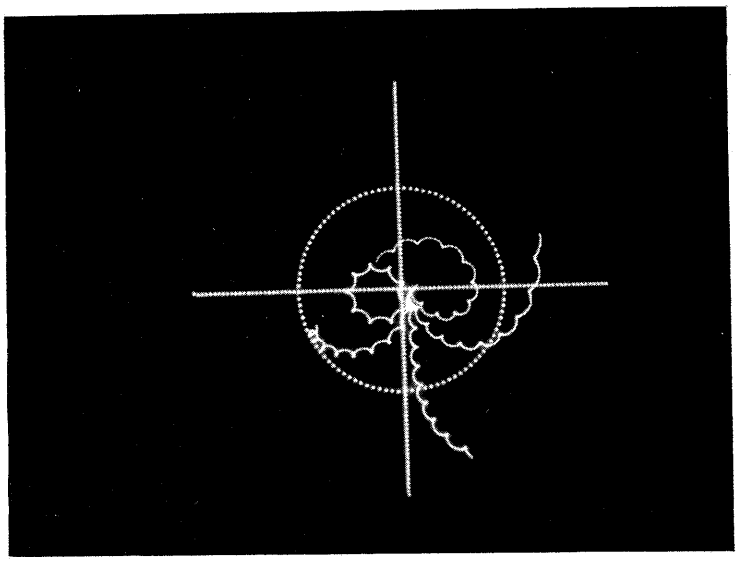

Figure 3. E(t) in the complex plane for $1 \leq t \leq 30$ with k = 0, ξ = 0 and (reading from left to right) u = 0.8, 0.9, 1.0, 1.1, 1.2. These u values bracket the region of resonance, i.e., of growing waves. The curves all start at the point E = 1. Also shown are the real and imaginary E axes and the circles $|E| = 8$ and $|E| = 16$.

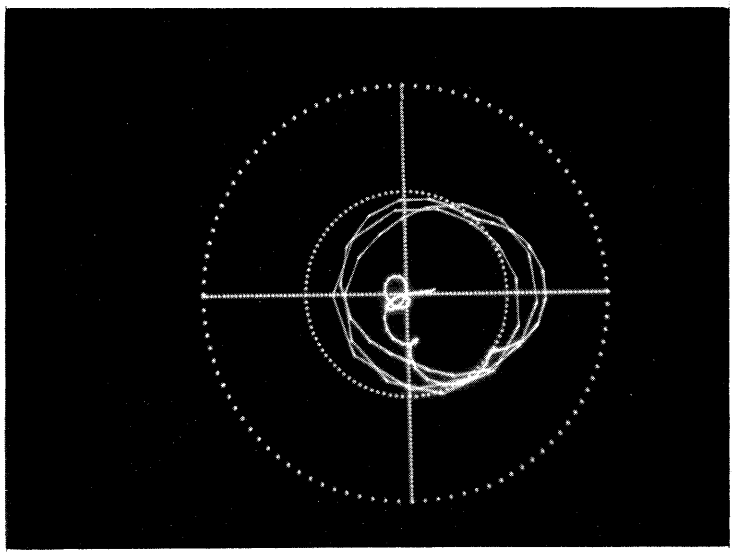

Figure 4. E(t) for k = 0, $\mathcal{E}$ = 0.1. The range of $d\psi/dt$, $0 \leq \dot{\psi} \leq 3$, includes the values used in Figure 3 (where $\dot{\psi}$ is constant). The approach of E to an approximate limit circle centered at E(t = 0) can be predicted analytically for the single species case (δ = 0). The circles |E| = 4 and |E| = 8 are shown.

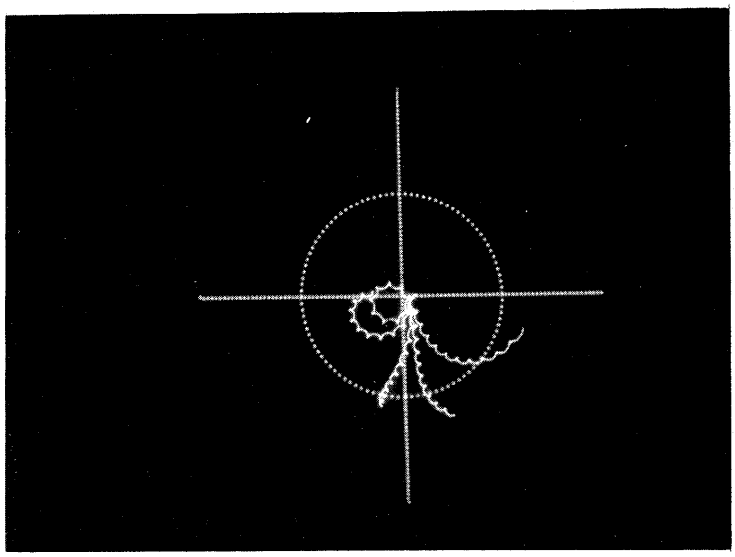

Figure 5. E(t) for k = 0.4, $\mathcal{E}$ = 0 and (from left to right) u = 0.9, 1.0, 1.1, 1.14, 1.2. Note the increased damping as compared with the k = 0 case. The circle |E| = 8 is shown.

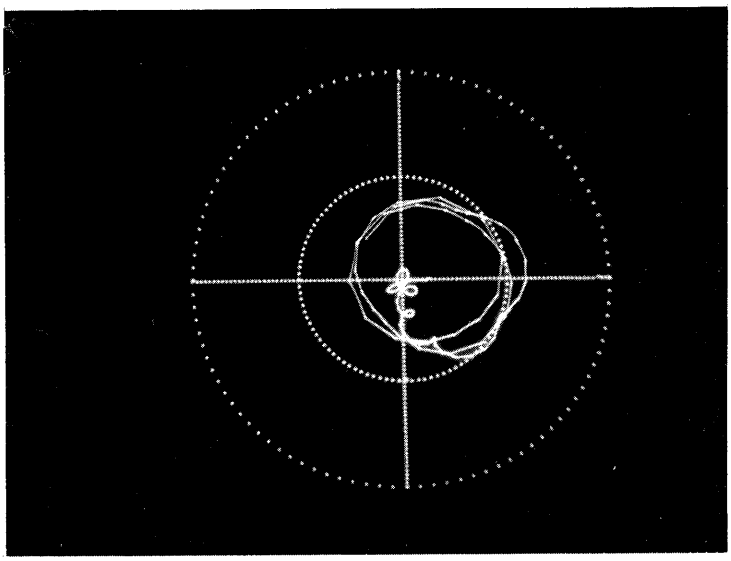

Figure 6. E(t) for k = 0.4, ε = 0.1. The analytic solution for the single species case (δ = 0) shows that the radius of the limit "circle" approached by E should be a slowly decreasing function of t. The circles |E| = 4 and |E| = 8 are shown.

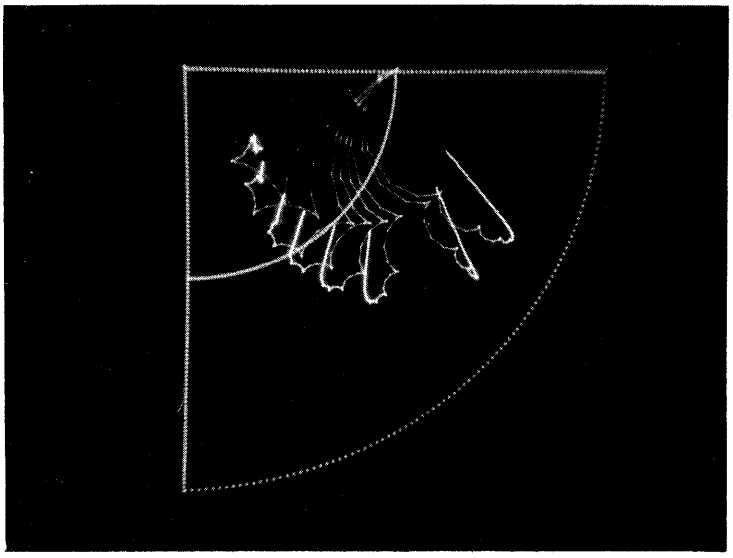

Figure 7. E(t) for k = 1.0, ε = 0 and (left to right) u = 0.8, 1.0, 1.2, 1.3, 1.4, 1.5, 1.7, 1.8. The high frequency (electron plasma) oscillations, $\omega \approx 1$, damp out completely during the time interval depicted, leaving only low frequency waves (associated with ion motion) which appear as bright "tails" on the curves. Arcs of the circles |E| = 1 and |E| = 2 are shown.

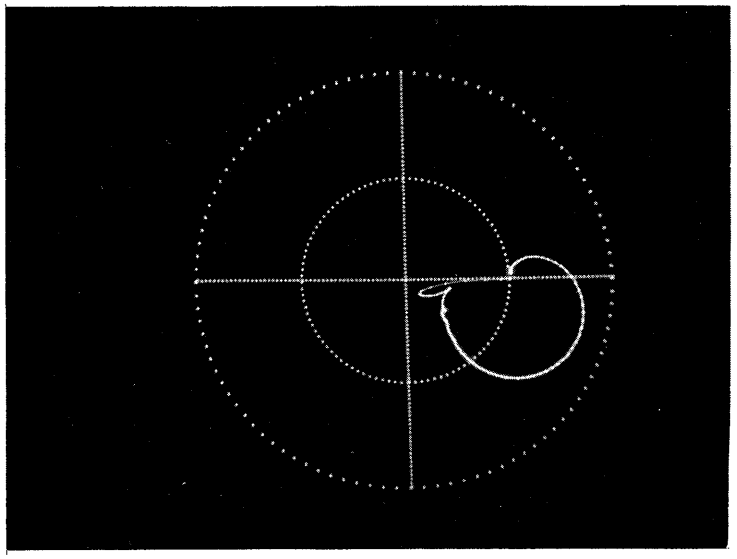

Figure 8. E(t) for k = 1, ε = 0.1. The limit circle for δ = 0 in this case should damp as $e^{-0.4t}$. The circles |E| = 1 and |E| = 2 are shown.