# On the User's Point of View

BURTON D. FRIED

*University of California, Los Angeles*
*and*
*TRW Systems, Redondo Beach, California*

In this discussion, I shall try to identify those general criteria for interactive on-line systems which seem most important for the experimental solution of mathematical problems. To illustrate some of these, I shall refer to Professor Glen Culler's on-line system [1, 3], which has been in operation at the University of California at Santa Barbara since 1966. While it is far from an ideal system, I believe that it still ranks first in terms of the number of real problems (of at least moderate difficulty) which have actually been solved, by a variety of users, with the aid of this system or one of its earlier versions. Its strengths and weaknesses have, therefore, some general relevance to a discussion of on-line systems for experimental applied mathematics.

A close analogy can be drawn between houses and interactive systems, with the architect corresponding to the system designer, the builder corresponding to the system programmers who implement the system, and the people who live in the house corresponding to the users. In both cases, considerations of practical convenience must play an important role in the basic design. In addition, a good architect will give strong weight to matters of aesthetics, while the designer of an interactive system for mathematical applications must be guided by a strong sense of mathematical structure.

The architect interacts strongly with the prospective "users" in setting specifications, and then initiates an essentially creative process involving many individual decisions. If these are made correctly, then the finished house has an attractive appearance, "lives well," and is aesthetically pleasing. However, if enough of the decisions are wrong, then the resulting house will be uncomfortable to live in, perhaps even ugly, even though it conforms to the formal specifications. All of these comments apply, *mutatis mutandis*, to interactive computer systems. As a general rule, houses designed by a talented

architect turn out better than those in which a builder selects a design (often based more on constructional conveniences than on the detailed desires of the client) and then builds a house.

Among successful houses, there is of course no unique design; the owner's needs and *tastes* must have a determining influence. So, too, with on-line systems. I clearly cannot represent, as the title implies, THE user's point of view; at best I can try to speak for those users, i.e., theoretical physicists, applied mathematicians, engineering analysts, etc., who need to solve difficult, albeit mathematically formulable problems. In this paper, I shall set forth some general principles which I think must be considered in the design of interactive systems for experimental applied mathematics. Like the satisfied occupant of a series of well-designed houses, I am necessarily influenced by the experience which I, and others, have had in solving problems using Professor Culler's systems, but I have attempted to abstract from this experience those general features which are invariant to the details of system implementation.

## I. The UCSB System

Since Professor Culler's system will be used for illustrative purposes, we very briefly list here some of its principal features. For a more complete account, the reader is referred to Karplus [1], which contains a reproduction of the user's manual for the 1966 UCSB system. We shall occasionally refer to an earlier version of this system, which has been in operation at TRW Systems since January, 1965, and also the to newest 1968 version of the UCSB system [3]. (The latter serves a number of local users and also, via phone line, remote consoles at UCLA, the Lawrence Radiation Laboratory (Livermore), and the University of Kansas. It is the most powerful system of this type, since it is implemented on a 3rd generation computer (IBM 360/65) and is part of a mixed on-line/batch system, with the on-line side providing an improved version of the earlier systems.)

Physically, a console, which *is* the system, so far as the user is concerned, consists of two keyboards and a (storage) oscilloscope for graphical and alphanumeric displays. One keyboard is designated the operator keyboard; each of its keys initiates a subroutine. The other, the operand keyboard, is used to designate addresses for data storage. In each case there is essentially a one-to-one correspondence between keys of the operator keyboard and subroutines, and between keys of the operand keyboard and data locations. The system is basically oriented toward functions, in the sense of classical mathematical analysis, in that subroutines typically work on a real or complex vector, having up to 125 components. Each user has, within the

computer, a "vector register," i.e., a set of core locations which can hold one such vector, together with a binary scale (for display purposes). Most of the basic macros of the system are either transformations of this vector accumulator, called the *y-register*, or display operations. For example, the exponential operation, initiated by pushing the EXP key on the operator keyboard, computes the exponential of the contents of the *y*-register, component by component, and leaves the result there. Pushing the STORE key of the operator keyboard, followed by any key of the operand keyboard, say G, (these keys are labeled as on a normal typewriter), transfers the vector contained in the *y*-register to location G; LOAD H transfers the vector in location H to the *y*-register; DISPLAY A plots the vector in location A on the scope face; DISPLAY 13 followed by carriage return (to indicate the end of a numerical sequence) displays the decimal value of the 13th component of the *y*-register, etc.

Different levels of mathematical operations, e.g., scalars, real vectors, complex vectors, etc., are assigned to different "levels" of the operator keyboard in a fashion analogous to shifts on a typewriter, save that some 18 such levels are available. Level changes are accomplished by a set of operator keys labeled with roman numerals. I for real scalars, II for real vectors, III for complex vectors, etc. For example, the operations of Level III act on a pair of real vector registers, termed the *x*-register and the *y*-register, which together constitute a complex vector register. Basic macros of the system are provided on Levels I through IX. An additional set of "user" levels, accessible by pushing a key labeled USER and then a roman numeral key, are initially empty. All of their keys are available for storage of user-generated programs, called console programs (see Section II.C).

## II. Some General Requirements

On the basis of several years of experience in solving mathematical problems with on-line systems, there appear to be several characteristics which a user should demand of any such system, irrespective of the particular details of its implementation.

### A. Rapid Response

In a discussion of "interactive" systems, it might be assumed that this requirement of rapid response would be satisfied automatically, but it is so important (and there are enough examples of systems which violate it) that it properly heads the list. It must appear to the user that "trivial" requests are executed immediately (i.e., in a fraction of a second), including return of

information to the user, if desired. The basic analytic macros of Culler's system provide good examples of such "trivial" requests. To perform an elementary unary mathematical operation on a vector of the order of 125 components: square, square root, running sum, first difference, sine, cosine, exponential, etc., or an elementary binary operation on two such vectors: sum, difference, product, or quotient, requires only a few milliseconds of computer time at most. This is likewise true of the elementary display operations such as displaying a curve or a numerical value.[1] For more substantial requests, a longer response is appropriate, but longer *only* in proportion to the work requested, so long as the user elects to remain in an on-line, interactive or foreground mode. He should have the option of sending jobs to the batch side of the system, and of accepting then the associated queuing and turn-around times.

A nontrivial contribution to rapid response is, of course, the use of oscilloscopes, rather than typewriters or plotters, although the latter do have their proper place, as discussed in Section J. Early oscilloscope consoles have been expensive and unwieldly, but current technology offers several inexpensive, neat solutions such as storage scopes, closed circuit TV systems, and cheap refresh techniques. Therefore we shall assume in what follows that the computer output, graphical or alphanumeric, is available to the user at a suitable oscilloscope console.

## B. Appropriate, *a priori*, Internal Organization

The system must provide a convenient representation at the level of classical mathematical analysis, so that the user need not contend with the tedious business of organizing single numbers into the functions, vectors, matrices, etc., used in higher mathematics. It is this unrewarding activity which constitutes the bulk of scientific "programming," Fortran and other languages notwithstanding, and which makes it so difficult for the user to keep his attention focused even on the mathematics of his problem, let alone on its significance in terms of the original application.

In Culler's system, for example, the basic macros, each initiated by a single key push, perform an operation on an entire vector of 125 components, making it possible for the computational steps to be actually at the level of functions, i.e., at the same level of abstraction as ordinary analytical work.

[1] Of course, whatever the maximum quantum of time which the system allots for one macro, there will always be some elementary operations which cannot be completed in that time. Display of a very long curve, and operations like the exponential on a complex vector are examples in present systems; matrix operations will join the list in forthcoming systems. These operations must be segmented, with at least one segment being executed each time the on-line system is serviced.

The user can think in terms of functions, push keys which generate operations at the functional level, and see a (graphical) display of any function at any time. He can, of course, work with single numbers when appropriate, evaluating a function at a point, or conversely, substituting a particular numerical value, etc. Early versions of Culler's system, including the one currently operational at TRW Systems, did not include convenient implementations of these single number operations, and the user was obliged to circumvent them, using the Dirac–Kronecker delta function operator together with right and left shifts. The dramatic difference in user convenience which resulted from the incorporation of these single number operations as basic macros of the system indicates the critical importance of a correct choice of basic macros in an interactive system. Extension in the other direction, allowing the user to work with two-dimensional arrays or matrices as basic objects is, of course, highly desirable for applications to partial differential equations, etc. This is one of the items promised in the 1968 UCSB system.

### C. Simple On-Line Programming Procedure

Each user must be able to create and modify his own programs, on-line, in a convenient way, without the necessity for learning a formal programming language. Since the users considered here have, by definition, facility with advanced mathematics, this can and should be exploited in the system design. One solution to the on-line programming problem, exemplified by Culler's system, takes advantage of the fact that every user must at least be able to operate in manual mode, i.e., to use the system as a fancy hand computer by pushing keys. Therefore, the programming need consist only of collecting or listing key pushes and of assigning this list to a previously blank key. The resulting programming structure is almost trivially simple (from the user's point of view!), is easily learned, and makes modification or elaboration of programs very simple.

For example, to differentiate one vector (say the one presently stored in the $y$-register) with respect to another vector which is considered as the "independent variable" (and stored, say, in location T), using a simple difference quotient approximation, we would, in Culler's system, push the following keys:

> DIFF (this computes the first forward difference of the $y$-register), STORE W (here W is just a working space, or temporary storage location), LOAD T (which brings the vector T into the $y$-register), DIFF INV (which leaves $1/dt$ in the $y$-register)$\cdot$W.

At this point, the difference quotient, $\Delta y/\Delta t$, is in the $y$-register and we might choose to store it for later use, say in location D, and also to display it by pushing STORE D DISPLAY D.

Pushing these keys will cause the computer to compute $dy/dT$ in real time, i.e., each operation is executed as soon as the key is pushed, and the display of $dy/dT$ appears on the scope immediately after the last key push. To run in this "manual mode" requires, of course, a certain familiarity with the system, that is, the location of the keys, the general vectorial structure, the nature of the basic macros, etc. We need just this kind of knowledge to operate any on-line device, be it slide rule, desk calculator, or automobile.

Once the user has learned to operate the system in "manual mode," he knows *in principle* all he needs to know about programming the system. The essential point is that this knowledge should suffice *in fact*, as well as in principle. In Culler's system, for example, a user wishing to create a *program* which would carry out differentiation as described above simply pushes a special key labeled LIST; then pushes exactly the sequence of keys given above; pushes LIST to signify the end of the sequence; and finally stores the list of key pushes thus created, that is to say, the new program, under any unused operator key on any user level. For instance, to create this program and then store it under the DIFF key on User Level II, he pushes

LIST DIFF STORE W LOAD T DIFF INV · W STORE
D DISPLAY D LIST STORE USER II DIFF.

The first push of LIST puts the computer in a list-making mode, so far as that console is concerned. Thereafter, it no longer executes the subroutines corresponding to the keys pushed, but instead constructs a list of these key pushes (both internally and as a display on the scope), thus helping the user keep track of his key pushes. The second push of LIST restores the computer to its normal state, ready to store the program just constructed (and, at that point, fully displayed on the scope) wherever the user wishes.

This program, created at the console and hence referred to as a *console program*, is now "stored" under the DIFF key on User Level II in the following sense: at any future time, the user need only go to User Level II (by pushing the keys USER II) and then push the key DIFF. The computer will fetch this list of key pushes and execute them in sequence, terminating with the display of $dy/dT$. Given that the console programs so created can themselves be part of other console programs (that is, among the key pushes constituting some other console program, one may include . . . USER II DIFF . . . ), so that console programs can freely call one another, as well as the basic macros of the system, it is clear that the pyramiding feature which constitutes the power of computer programming is preserved, without the linguistic elaborations commonly encountered.

Several factors account for this simplicity. We have assumed the users to be mathematically sophisticated, so that the system can, should, and does, take advantage of the complicated structure of mathematics which each user brings

with him in his head. Similarly, it seems to be true that successful on-line systems oriented to programming applications exploit the specialized knowledge which their users can be presumed to have. In addition (and partly in consequence of this) the basic macros are appropriately structured (see Section B), thus freeing the user from most of the kind of programming details for which elaborate languages are in fact necessary.

It can be argued that a system like Culler's involves a new "language" for users, but this misses the basic point. Any on-line device, automobile, desk calculator, typewriter, or console, requires that the user master a certain kinesthetic or manual skill. We may call this a language, if we wish; what is desirable is just that, having mastered the *operation* of a console, the user should not need to know more in order to construct programs at the console. This is a rather elementary point; that it needs mention at all is simply a consequence of the fact that there is still so little experience with sophisticated, powerful on-line systems. As these become more common, this point of view will probably be generally accepted.

It should, of course, be easy for the user to inspect, edit, modify, and document his console programs. All of these features are realized in Culler's system, the last only in primitive from. The key pushes USER II DISPLAY DIFF will cause the computer to display, on the scope, the constituent key pushes of that program, i.e., the same display as that which existed when the user finished making the console program originally. At this point, he has available (in the 1968 UCSB system) a very sophisticated and convenient on-line editing capability for modifying that program. He can then store the new version, either in the location from which it came (USER II DIFF) or in some other location. A descriptive, English language, comment concerning the program can be constructed, stored with the program (the second LIST key push is followed by STORE USER II CTX DIFF), and redisplayed at will (USER II DISPLAY CTX DIFF). Finally, a new name, *d/dT* for our example, can be associated with any key of any user level, with the consequence that when that key is used in some other console program, the scope display will show, not its native name (...USER II DIFF...), but rather the name assigned to it by the user (...USER II *d/dT*...).

### D. Availability of Data

An important principle, neglected in some current systems, is that data resulting from a sequence of on-line calculations should be available thereafter to the user for further computations after examination, or modification, by the user. These data should reside at a level of mass memory appropriate to the size of the data block, unless or until the user replaces them with other data, or in some fashion indicates that they are no longer to be considered current.

### E. Direct User Control

The user should have full control of his part of the computer, i.e., of the section of memory (both fast core and mass memory) assigned to him by the system, as well as control of the actions of the central processing unit during the times when it services his requests. This includes his initiation of subroutines, either those provided by the system or those constructed by him at the console, and requests for computer output, graphical or alphanumeric.

### F. Development of Individual User Systems

It is absolutely essential that each user have convenient ability to accumulate his own subroutines, as well as data, and to build a user system specialized to his particular problem area. When he leaves the console, he must be able to store his system (in mass memory) so that when he returns and reloads his system, the computer is, from his point of view, restored to the same condition as when he left it. It is simply not possible to attack mathematical problems of much significance if this ability to store user systems is not provided.

Experience to date has shown that even for rather modest problems, the user may need space of the order of a few thousand words for storing his own subroutines. The additional space for data can vary widely, depending on the problem, but a total of 10 to 15 thousand words for data plus subroutines is probably a minimum for nontrivial problems. The requirements for *live* mass memory, available when the console is being used, may be 2 or 3 times greater than this.

### G. Graphic Input

Although the usefulness of graphic output for problem solving in applied mathematics is well established, there is, so far, little experience concerning the role of graphic input. At present, it appears that the only interactive system which has a power comparable to that of Culler's UCSB system, and which also allows for graphic input, is the three year old TRW Systems On-line Computer, where a stylus device (the BBN Graphacon or Rand Tablet) has recently been interfaced to one console of the system. The stylus can be used to sketch a curve, which the computer then puts into normal digital representation for curve fitting or other analytic processing. In addition, Robert Schreiner has prepared simple console programs which allow a user to employ portions of the tablet as a pseudokeyboard, particularly in connection with elementary graphic input operations pertinent to computer aided design. Although this appears to be a significant addition to the system's capabilities, a careful assessment must await further experience. One point which should

be noted is that a stylus device of this sort can accomplish all of the tasks for which a light pen is normally used, and in addition, do things which are at best awkward with a light pen because of the parallax, e.g., accurately trace a curve from a sheet of paper into the computer.

### H. Availability of Hard Copy

Considering the stacks of computer printout which clutter the offices of most heavy users, this requirement may seem quite superfluous. However, in a proper on-line system, wherein any data is instantly available at a console in graphical or alphanumeric form, we may almost forget that hard copy has its proper and very important place. This is, to be sure, far different than in a batch system; the on-line user can be quite selective, examining a large volume of data at the console and sending to the hard copy device only that portion which he really wants to have on paper. The proper time to arrange scales, ranges and increments of variables, graphical format, and the like to suit the user's needs or tastes is while the information is still in the computer. Only then, after appropriate checking with the scope, should hard copy be called for, thus saving the nuisance of further transformation of data by hand computing and graphing which so often follows the computer solution of a problem.

For many purposes, a polaroid camera snapshot of the scope face is the most convenient form of output, but this needs to be supplemented with a printer or typewriter of some kind, to handle extensive alphanumeric output. Some type of plotter for graphical output is also desirable, since it is often a great convenience to be able to annotate curves in a way which cannot easily be done on a photograph, and then give the whole page to a draftsman for copying in india ink on vellum, as when preparing manuscripts for publication.

In consequence of the ready availability of results at the console, a user can scarcely make a strong case for having hard copy quickly. Information which he is very anxious to have can be obtained at the console; pictures of the scope can be snapped if he wants something to take away. For the hard copy, particularly in large volume, an overnight turn-around time should generally suffice. Correspondingly, it is not really necessary for the hard copy to be available at the remote console; it could simply be produced at the computing center.

### I. Convenient Construction of Problem Elements and Their Subsequent Composition

The activities involved in the on-line solution of mathematical problems divide into two not wholly disjoint parts. In the first, the user simply constructs

the analytic elements in terms of which the problem is stated, or those elements needed to implement a particular method of solution. The second, and more challenging task, experimental and inductive in nature, involves the combination of these elements to provide a successful technique for generating the desired solution or solutions. This second aspect is hard to illustrate; problems whose solutions are not yet understood can scarcely provide an instructive demonstration, whereas solving one whose structure is known *a priori* really comes under the first category, i.e., involves only direct construction. Some attempts to illustrate both of these aspects of on-line problem solving may be found in Karplus [2] together with a discussion of the complex interrelation between them. A user begins with the constructive elements, moves on to the problem solving itself, drops back to the constructive mode as he discovers the need for new tools, etc. This entire activity is fruitful only if the system as a whole has been properly designed, particularly as regards the subject matter of Sections A, B, and C.

### J. Flexibility Regarding User Needs

As noted earlier, an appropriate choice of basic macros is of vital importance. Although we can be guided by the general principle of avoiding either a starkly irreducible set or an overly redundant one, nothing can substitute for experience. The adequacy and convenience of Culler's set of macros, for example, is partly a consequence of experience (by many users) with the five earlier versions of the present system. On-line system designers could take advantage of this experience by using this set as a starting point, adding new elements as needed. Regardless of the intial complement chosen, however, it is imperative that systems programmers pay close attention to user needs. Of course, it can happen that the correct and appropriate response to a user's complaint, that he cannot conveniently carry out a certain operation or process, is simply to show him that indeed the system already contains the desired capability. If in fact it does not, it may still be possible to construct a console program which does what is required, even if awkwardly or inefficiently. The user can then experiment with this, modifying it and finally arriving at an accurate specification for a new basic macro which the systems programmers should then be able (and willing!) to implement quickly (which of course implies a certain sophistication and modularity for the underlying system).

These, then, are the criteria of at least one user; their practical realizability is proven by the example of Culler's system. As noted earlier, that system provides a reasonable starting point for further developments, which ought to include:

1. implementation on a larger computer (IBM 360/91, CDC 7600, or

equivalent) so that on-line techniques could be used also for problems requiring heavy computing (e.g., particle simulation problems in kinetic theory, three dimensional fluid dynamics problems, etc.),

2. provision for use of vectors with more than 125 components (with appropriate penalties regarding speed and user storage allotments),

3. addition of basic macros for two dimensional arrays and for matrices (e.g., algebraic operations plus diagonalization for Hermitian matrices),

4. extension to algebraic (symbol manipulation) types of problems,

5. easy communication between the batch and on-line sides of a mixed batch/on-line system, allowing on-line users to send appropriate jobs to the batch side (e.g., production versions of problems constructed and checked out on-line) and subsequently to examine the results on-line and further modify the console programs, etc.

All of these items, and others besides, are promised for the new UCSB system; most, if not all, have in fact been suggested by Professor Culler himself. The crucial point here, and one of general applicability, is that given a well-structured system such as Culler's, having generality (in the console programming sense) and power exceeding a certain threshold, each addition to the repertoire of capabilities leads to a very impressive increase in the overall power of the system. An on-line interactive system for applied mathematics should aim at incorporating, as much as possible, the structure and open-ended character of mathematics itself. Such a system will then be comfortable and convenient for people with mathematical training to use, and growth, both of the on-line system itself, and of individual users' systems, will be easily and naturally achieved.

## REFERENCES

1. CULLER, G. J., Appendix, Users' Manual for an On-Line System, *in* "On-Line Computing" (W. J. KARPLUS, ed.), p. 303. McGraw-Hill, New York, 1967.
2. FRIED, B. D., Solving Mathematical Problems (Chap. VI), *in* "On-Line Computing" (W. J. KARPLUS, ed.), p. 131. McGraw-Hill, New York, 1967.
3. UCSB On-Line System Manual, Univ. of California at Santa Barbara, California, October 1967.