

AD-761 996

A METHOD FOR EXTENDING FORTRAN V FOR
THE INTERACTIVE GRAPHICAL SOLUTION OF
NUMERICAL PROBLEMS

Alan C. Reed

Utah University

Prepared for:

Advanced Research Projects Agency

December 1968

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va 22151

960192 07

A METHOD FOR EXTENDING
NORTON'S FOR THE INTERACTIVE
GRAPHICAL SOLUTION OF
NUMERICAL PROBLEMS

Alvin C. Reed

TECHNICAL
REPORT 4-70

NATIONAL TECHNICAL
INFORMATION SERVICE

A METHOD FOR EXTENDING FORTRAN V FOR THE INTERACTIVE
GRAPHICAL SOLUTION OF NUMERICAL PROBLEMS

December 1968

COMPUTER SCIENCE
Information Processing Systems
University of Utah
Salt Lake City, Utah

Advanced Research Projects Agency • Department of Defense • ARPA order 829

Program code number 6D30

•
/

ABSTRACT

During the more recent history of Computer Science, a great deal of effort has been directed toward the development of more rapid methods of interaction between man and machine than is possible with current batch processing schemes. One specific subset of this field of study is the area of Interactive Computer Graphics, where tools such as display scopes and keyboards, among others, are used to effect more efficient problem solution. In connection with this, a number of programming languages are being written which are geared to and which take into account the capabilities of graphical systems. However, a great number of programs now exist in industry which were written in FORTRAN or COBOL. In particular, many of these FORTRAN programs were written for the solution of various physical problems using numerical analytical techniques.

One important characteristic of such programs is that a significant amount of coding is required to handle breakdowns in the algorithm which occur during the problem solution. For example, the non-convergence or divergence of an iterative procedure might be handled by attempting to change some program parameters such that convergence may then be achieved. The final solution of a curve fitting problem may require several iterations, each with a different fitting function or norm. In each of these cases, the additional amount of decision-making code required to adequately control the algorithm may be very large. To be more accurate, the amount may range from simple code, which simply

terminates the program, to that which is necessary to manage and correct every conceivable breakdown.

The simple code, of course, requires very little in the way of programming effort, but generally must be submitted into the batch several times in an iterative manner before the final solution is obtained. The more complex code, on the other hand, is longer in development, but probably requires fewer runs to complete the problem. An entire realm of compromise or trade-off exists between the two extremes, and it is there that most of the existing numerically oriented programs are found, probably for reasons of economics, or available development time, or both.

Interactive computer graphics offers much in the way of bringing the program user closer to the problem solving algorithm. Techniques may be employed which provide for the execution of the program to be monitored and action taken to help the program find the solution, or to find it more quickly. Logic for handling breakdowns in the algorithm, much of which would not normally be coded into the program, may be economically employed, provided means are available to implement these techniques in existing FORTRAN programs. This paper is the result of a first step toward providing those means.

INTRODUCTION

At the University of Utah, a graphics laboratory has been established to provide the Computer Scientist with tools of research into graphical techniques.

The equipment in the graphics laboratory includes:

A PDP-8 computer connected on-line to the University's Univac 1108. The PDP-8 acts as a controller for the graphics equipment and serves as the communications link between this equipment and the 1108.

A model 35 teletype that serves as the interactive keyboard.

An IDI display scope which may be used to provide a window into the solution space of the problem, and may be combined with the teletype to form an interactive console with keyboard and display.

Descriptions of other equipment present in the graphics laboratory are omitted since knowledge of its existence is not necessary background material for this paper.

In order to provide the means by which a FORTRAN program may be given interactive capabilities, a set of subroutines has been written which allows a programmer to declare two new types of FORTRAN variables. The first type is called an "interaction variable", the value of which may be changed or simply retrieved by the program user. The second type is called a "command variable". A command variable, when typed, causes transfer of execution to a statement number in the program and may also have an associated value.

METHOD OF IMPLEMENTATION

Namelist is a FORTRAN V feature which allows unformatted input and output of declared variables (1). The basic form of a namelist statement is:

NAMelist/NAME/VAR1,VAR2,...,VARN

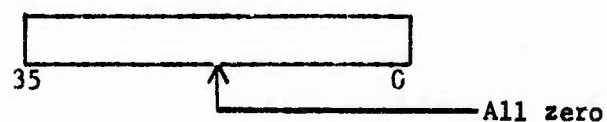
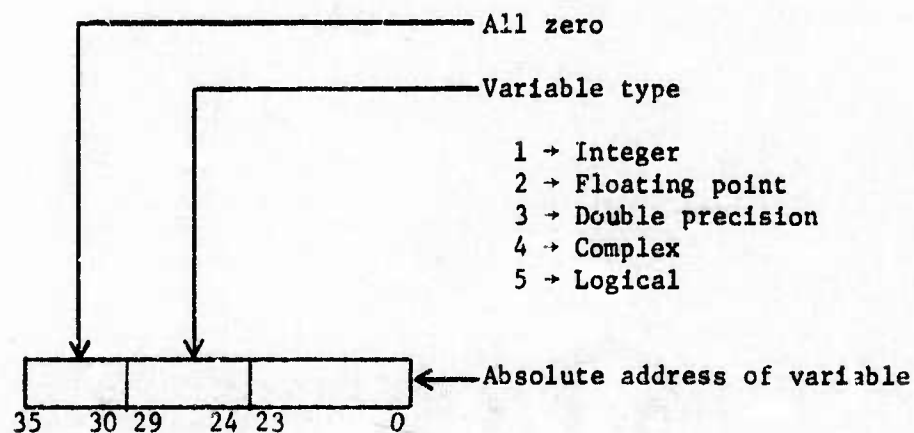
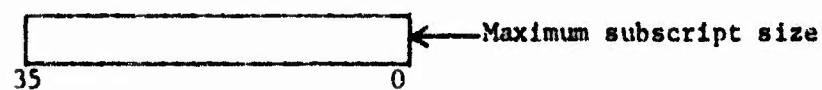
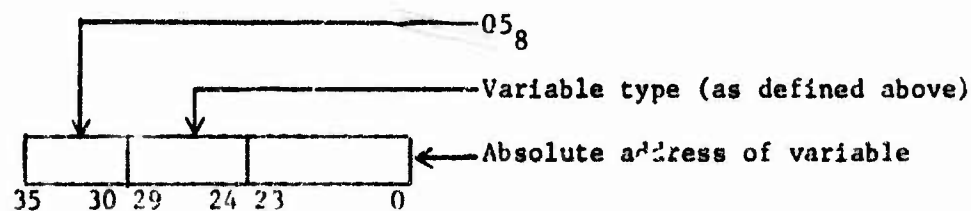
where NAME is a namelist name; and VAR1,...,VARN are simple variables, subscripted variables, or array names.

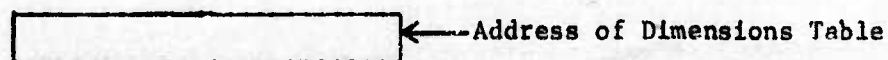
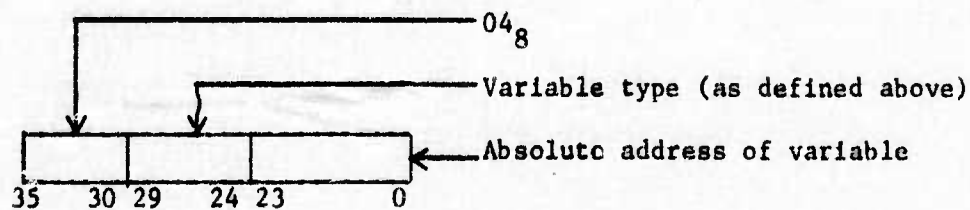
In order for a namelist to provide input/output, a table of information regarding each variable declared in the list is generated by the compiler and exists in memory at execution time. It is this information which is used to obtain addresses and other necessary information for this implementation. The exact structure of the table is perhaps best described by a semi-pictorial description which assumes that N variables were declared.

NAMelist TABLE

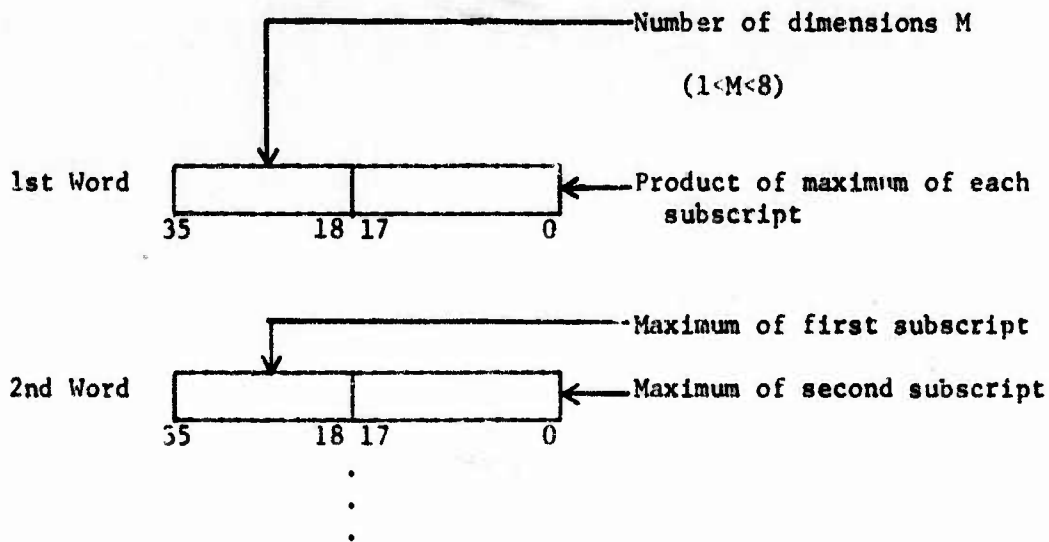
<u>Relative Location</u>	<u>Configuration</u>	<u>Explanation</u>
1		← Absolute location corresponding to relative location $N + 2$ The characters \$\$\$ in fielddata code
2		← Namelist name in fielddata code, left adjusted, with blank filler
3		← N Variable names in fielddata code, left adjusted, with blank filler
$N + 2$		
$N + 3$		
$N + 4$		← N pairs of words, each pair corresponding to a variable name
$3N + 1$		
$3N + 2$		

The information contained in the word pairs shown above depends upon whether the corresponding variable is non-subscripted, singly subscripted, or multiply subscripted.

Non-SubscriptedSingly Subscripted

Multiply Subscripted

The Dimensions Table Contains:



(Repeat to Mth subscript)

The transfer of control associated with a command variable has been implemented by taking advantage of the code generated when a statement number is specified in a subroutine call list. This is done whenever the RETURN K feature of FORTRAN is used (1). The FORTRAN statement:

```
10 CALL SUB ($10,$20)
```

where the numbers 10 and 20 are statement numbers in the program, would result in the machine language equivalent of the following Univac 1108 assembly language code:

10L	LMJ	X11,SUF	. TRANSFER TO SUB
	J	10L	. JUMP TO STATEMENT 10
	J	20L	. JUMP TO STATEMENT 20
	(W.B.)		. ERROR WALK BACK

It may be seen that a table of jump instructions is produced that provides branches to various statement numbers in the program.

An assembly language subroutine, LGCATE, has been written for the purpose of providing access to the namelist and jump tables of information for a FORTRAN driver program. It is called by a statement of the form;

CALL LOCATE (LABEL,VEC)

where LABEL is a location containing the name of a variable declared in a namelist, left adjusted, and with blank filler. VEC is a singly subscripted array of eleven locations which contains the information listed below when the variable is found and the subroutine returns to the calling program:

<u>Position in VEC</u>	<u>Contents</u>
1	Absolute address of variable
2	Variable type (as previously described)
3	Product of maximum of each subscript
4	Number of subscripts
5	Maximum of first subscript
6	Maximum of second subscript
.	.
.	.
.	.
11	Maximum of seventh subscript

If the variable is not located, all positions in the array VEC are set to zero. LOCATE is capable of searching several namelists, (the present maximum is fifty), and they may be declared in either the main program or a subroutine.

The programmer declares a namelist to LOCATE by a call to subroutine SETLST. Transfer locations for command variables are declared by calling subroutine JUMPS. The use of these two routines will later be described in more detail. Three other entry points that are of interest here are SET, TRANS, and UNSET. Subroutine SET is used just prior to a call on LOCATE for the purpose of enabling transfer of control if the variable subsequently located is a command variable. Any such transfer, however, does not occur until subroutine TRANS is called. If no transfer has been enabled, TRANS simply returns to the calling program. UNSET may be used to disable a transfer which was enabled by SET. It is useful whenever a command variable input is found to be in error after the variable has been recognized and the transfer established.

The set of routines described above makes it possible to perform all line scanning and value conversions in a program written with a higher level language than is assembly language.

TTY is a FORTRAN subroutine which was written for the purpose of scanning a fielddata character string and interpretively executing the statement according to a set of specific rules. For this job, TTY uses LOCATE, SET, TRANS, and UNSET. It may, from the programmer's point of view, be thought of as the subroutine to be called whenever input from the user at the teletype is required or desirable.

At the moment, the capabilities of TTY are limited in scope, but will be more generalized in the near future. Its primary functions are to:

Store values typed by the user at the addresses of user designated interaction variables.

Retrieve and type to the user the current value of any interaction variable requested.

Effect the transfer of control associated with a command variable.

USAGE INFORMATION

In order for interaction and command variables to be defined, they must be declared to LOCATE through the entry point SETLST. The namelist name must be passed to SETLST in order that the namelist table of information be made available. Unfortunately, a namelist name may only appear in a READ or WRITE statement. For this reason, a call to subroutine SETLST must be followed by a READ or WRITE statement as demonstrated by the following example:

```
NAMELIST/NAME/VAR1,VAR2,...,VARN
.
.
.
CALL SETLST
READ (5,NAME)
```

SETLST anticipates that a READ statement of the form shown will immediately follow the subroutine call. The information given in the READ statement may be thought of as the argument list for SETLST. The subroutine returns to the next statement following READ.

An interaction variable is one which has been declared in a NAMELIST where the name of that namelist has been provided to subroutine SETLST at execution time.

A command variable is an interaction variable which is further declared in a call to subroutine JUMPS.

```

NAMELIST/NAME/VAR1,VAR2,...,VARN
.
.
.
CALL SETLST
READ (5,NAME)
.
.
.
CALL JUMPS ('NAME',$STMT1,$STMT2,...,$STMTM)

```

In the above example, $M \leq N$. The call to JUMPS declares the first M variables appearing in namelist NAME to be command variables. When VAR1 is typed, $1 \leq I \leq M$, the program transfers to STMTI in the program.

Finally, whenever the teletype is to be interrogated, a call to subroutine TTY is made. Usually, the call will be the character interrupt processing location specified in a call to CHRINT as demonstrated by the following hypothetical example (2):

```

LOGICAL FLAG
DIMENSION I (10,10)
NAMELIST/NAME/PLOT,STOP,A,GO,I,J,FLAG @ DEFINE
CALL SETLST @ INTERACTION
READ (5,NAME) @ VARIABLES
CALL JUMPS ('NAME',$10,$20,$30,$40) @ DEFINE COMMAND VARIABLES
CALL CHRINT (1,$5) @ GO TO 5 WHEN 1 CHAR IS TYPED
2 CALL IDLE @ WAIT FOR
GO TO 2 @ INTERRUPT
5 CALL TTY @ INTERRUPT OCCURRED, CALL TTY
CALL INTRET @ RETURN TO IDLE LOOP
10 CALL PLOTFR (I,J) @ 'PLOT' COMMAND WAS TYPED
CALL INTRET @ RETURN TO IDLE LOOP
20 CALL EXIT @ 'STOP' COMMAND WAS TYPED
30 NEWVAL = A**3 + .5 @ NEW VALUE OF 'A' WAS TYPED
CALL INTRET @ RETURN TO IDLE LOOP
40 CALL COMPUT (NEWVAL,FLAG) @ 'GO' WAS TYPED
CALL INTRET @ RETURN TO IDLE LOOP

```

The above program would call IDLE repeatedly until a character interrupt caused a transfer out of IDLE to statement number 5 where TTY would then be called (2). TTY would wait until a carriage return () was typed at which time it would scan the input line. What happens next would depend upon the line that was typed. Some examples will illustrate:

<u>INPUT</u>	<u>EFFECT</u>
PLOT	Transfer to statement 10
STOP	Transfer to statement 20
A=3.5	Value 3.5 is stored in A then transfer to statement 30
A=3	Value 3.0 is stored in A then transfer to statement 30
A	Transfer to statement 30
I(1,1)=37	Value 37 is stored in I then return to statement after CALL TTY
J=2	Value 2 is stored in J then return to statement after CALL TTY
I(J,J)=40	Value 40 is stored in I(J,J) where the <u>current</u> value of J is used, then return to statement after CALL TTY
I(1,25)=2	Characters '25' underlined with '*' (subscript out of range) then wait for more input.
A=	Type out current value of A. Do <u>not</u> transfer. Return to statement after CALL TTY.

The above list includes most of the common forms of input, but is by no means exhaustive. The following rules govern the use of TTY:

Values must agree in type with variable names as defined in the program. The only exception is that a floating point variable may be given the value 3 in lieu of 3. or 3.0.

Floating point input may be either of the F or E format type.

Complex values are input as two floating point values separated by a comma.

Subscripts may be either integer constants or non-subscripted interactive integer variables.

If a label and an equal sign are typed but not a value, the program assumes retrieval is wanted and types out the current value of the variable.

The user may effect the transfer of control associated with a command variable either by typing only the name of the variable, or by specifying a value for the variable.

Command variable transfer will not occur when there is error in the input string or when the value of the variable is being retrieved.

Whenever an error is detected in the input, the element found to be in error is underlined with the character '*', and the subroutine waits for more input.

All input must be followed by a carriage return.

APPLIED EXAMPLE AND CONCLUSIONS

A complete practical test of this system was initiated several months ago when an existing FORTRAN program was converted to the interactive mode. The problem solved by the program is the fitting of a set of data points collected at the Material Test Reactor (MTR) at Arco, Idaho. The data points represent the cross section, σ , of a test sample as a function of the energy, E , of neutrons bombarding the sample. The theoretical model for describing this function is the single-level Breit-Wigner formula:

$$\sigma_t(E'') = \sum_{i=1}^m \left[\frac{4\pi g \Gamma_1^n [\Gamma_1^y (E_1^0/E'')^{1/2} + \Gamma_1^n]}{(E'' - E_1^0)^2 + (\Gamma_1/2)^2} + \frac{16\pi g \Gamma_1^n (E'' - E_1^0) R}{(E'' - E_1^0)^2 + (\Gamma_1/2)^2} \right] + 4\pi R^2$$

where:

$$\Gamma_1 = \Gamma_1^y + \Gamma_1^n$$

g is a statistical weight factor

Γ_1^n , Γ_1^y , and E_1^0 are resonance parameters to be determined in the fitting process

R is the radius of the nucleus of the sample

m is the number of resonances

The formula is corrected for Doppler effects by computing the function:

$$\sigma_{\Delta}(E') = K'(E', E'') \sigma_t(E'') dE''$$

The transmission, T_{Δ} , may then be calculated:

$$T_{\Delta}(E') = e^{-N\sigma_{\Delta}(E')}$$

where N is the thickness of the sample.

T_{Δ} must be corrected for an effect caused by the finite resolution of the measuring equipment.

$$T_R(E) = K(E, E') T_{\Delta}(E') dE'$$

Finally, the corrected cross section is:

$$\sigma(E) = \frac{1}{N} \ln (1/T_R(E))$$

This cross section is then compared to the data points. The resonance parameters Γ_1^n , Γ_1^0 and E_1^0 are changed, and the function again computed successively until a good fit is obtained.

It should be mentioned here that other mathematical models, such as a Fourier series, could be used to fit the data. However, the Breit-Wigner formula retains the useful theoretical concepts involved in the phenomenon being observed. A Fourier series or some other method, although providing an analytical description of the data, would allow the physics to be lost.

The FORTRAN program which calculates the fit has been converted to a subroutine and a FORTRAN control program written to provide the interaction and display capabilities.

The basic structure of the control program is similar to that of the hypothetical example previously described. Each command variable defines a particular process to be performed when that variable is typed. In addition to the main control program are program elements for scaling the data, drawing axes on the ID1 display screen, and for performing an inverse indexing operation. The latter is necessary to allow the program user the convenience of specifying parameters in the units of the original data, rather than in the units of the displayed image which are typically far removed from his knowledge.

There are several command variables for controlling the program. The first such variable is DATA, which is used to tell the program how many data points are to read from the input file. Another command is

VIEW, which is used to select an energy field of view. VIEW is defined in the program to be a complex command variable and is therefore input as a complex constant. The real and imaginary parts of the constant are associated with the lower and upper limits of view, respectively.

After a view has been selected, the program may be forced to scan the data to the right or left with the respective commands SCANR and SCANL. When wither is typed, the picture begins changing in an animated manner, as though the user were performing a pan operation with a camera while observing through the view-finder. The scan stops whenever the end of the data is reached, or if a value is typed with the command and that value is reached, or when the command HALT is typed.

Once a view of the data is selected of which a fit is desired, the user may type FIT to initiate a call upon the fitting program.

At present, the only interaction possible with the actual fitting program is through some interaction variables. It is expected that the fitting program will be replaced by one which uses the multi-level Breit-Wigner formula in the near future. Therefore, interaction with the present fitting program has assumed a very low priority.

The control program evaluates the fitting function at a specified number of points across the interval and displays line segments connecting

the values of the function at those points. Fitting parameters may then be changed and the fit again computer if necessary.

When a given interval has been fitted satisfactorily, a hard-copy of the results may be obtained by typing PLOT. The program generates the copy on a Calcomp plotter.

The command, SWAP, may be typed at any time after all data points have been read, and is a signal to the program to subsequently swap out of the computer whenever it is waiting on the user for input (3).

The STOP command terminates the program.

A problem was run in the interactive mode on a set of data collected at MTR from an experiment using a sample of ^{243}Am . Figure 1 shows an interval of data points and the fit computed with the initial values of the resonance parameters. The next step in the process was to raise the peaks to conform more to the data. The results of these changes are shown in Figure 2. For a final correction, Figure 3 shows the result of broadening one peak slightly. The total time for obtaining this solution was approximately one-half hour.



Figure 1 - Data points and single-level Breit-Wigner fit using the initial resonance parameters.

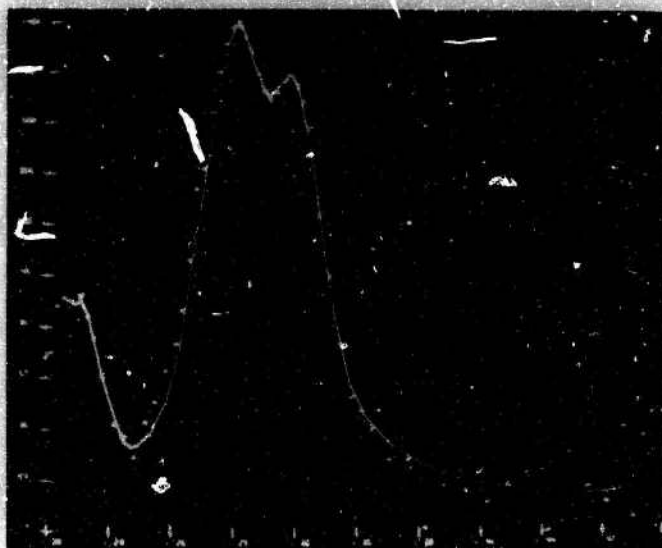


Figure 2 - The effect of raising the peaks by interactively modifying the values of some parameters and fitting again.

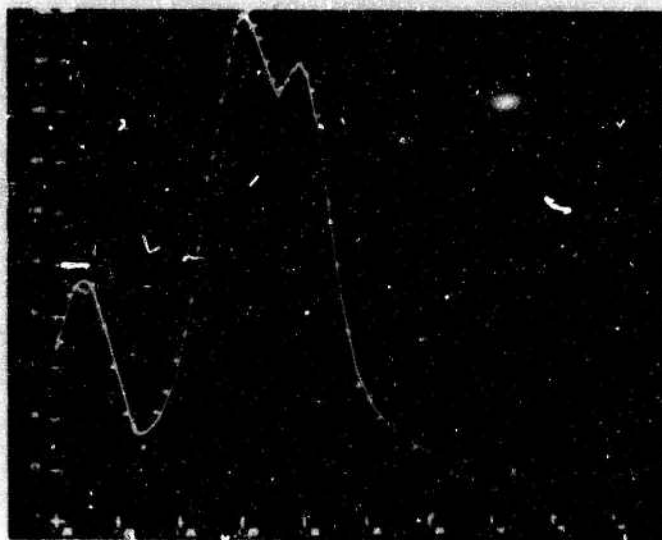


Figure 3 - The final solution obtained by broadening one resonance slightly.

When the program operated in the batch mode, the data was plotted and values of the resonance parameters determined on the basis of that plot. The program was then given the data and the estimated parameter values. It then calculated the fitting function. The process was repeated until a good fit was obtained. The time required to obtain the fit was generally of the order of four or five months.

By contrast, the interactive program allows the values of the resonance parameters to be determined at run time allowing the fitting process to be accomplished in one sitting. In addition, the control program and associated display routines required approximately one man-week of development time. The point to be made here is that interactive graphics may save very significant amounts of time in the solution of numerical problems, and that with a suitable means of implementation, the time required to convert an existing batch mode program to the interactive mode is often negligible.

DIRECTION OF FURTHER RESEARCH

The demonstrable usefulness of interactively working with some types of FORTRAN programs logically points to the possibility of designing an entirely new interactive language.

The language should:

Be oriented, in some sense, towards mathematics to facilitate a variety of numerical problems.

Provide list processing capabilities for ease of handling a large and complex data base.

Be capable of performing input/output with a large variety of digital and analog devices.

Be syntactically consistent and allow a highly sophisticated input/output syntax to be defined by the programmer, possibly even at execution time.

Provide well designed debug capabilities for both the static and dynamic programs.

It is quite possible that a carefully designed language of this type would prove invaluable for the computer solution of a great number of complex problems in science and engineering.

LITERATURE CITED

1. Sperry Rand Corporation. UNIVAC Division, 1108 Multi-Processor System Fortran V, Programmer's Reference Manual, UP 4060. New York, c 1966.
2. Copeland, Lee and Carr, C. Stephen, Graphics System. Salt Lake City, University of Utah, Computer Science Information Systems, Technical Report 4-1, Nov. 15, 1967.
3. Reed, Alan C., Dallin, D. E. and Bennion, Scott T., A Fortran V Interactive Graphical System. Salt Lake City, University of Utah, Computer Science Information Processing Systems, Technical Report 4-4, April 3, 1968.