Behavior Research Methods & Instrumentation
1982, Vol. 14(5), 475-481

APL functions for interactive data analysis: Preparation of data files

SELBY EVANS, JERRY D. NEIDEFFER, and LYNN GILFILLAN
Texas Christian University, Fort Worth, Texas 76129

APL functions to support preparation of data files are presented: a function to manage the entry of raw data, functions to display the entered data in formats convenient for checking, a function to support correction of errors, a function to organize the data into tables and file them, and a user interface function that provides menu selection of the data preparation functions. General-purpose support functions to assist in file use and in menu selection are also provided.

Behavioral research often requires substantial amounts of data. Frequently, the data are accumulated over a period of time and require analyses by several different procedures. Under these circumstances, APL's traditional method of storing data in saved work spaces becomes inconvenient. Direct entry into a file offers advantages. Since the entries are not retained in the work space, there is practically no limit to the amount of data that can be entered. The data are better protected from user and computer errors. With functions designed to simplify the entry operations, data entry can be turned over to clerical personnel. Data analysis functions can be written to assume standard file conventions (Evans, Neideffer, & Gage, 1981) that let the user move freely from one analytic procedure to another.

This report presents a system to support file-oriented data entry. The system allows for data entry with detection of errors in the number of entries, display of filed data, error correction, and organization of data into tables in a labeled file for analysis by functions such as PRINAN (Evans et al., 1981) and GRAPHICS (Evans, Neideffer, & Gage, 1980). The system is menu driven; that is, it presents a list of options to the user and interprets the user's response as a selection from the list or menu. Menu selection relies on recognition rather than recall, so that it is easier for the inexperienced user.

Several general-purpose support functions are also presented, providing amenities to simplify the use of the other functions. With these functions, most data entry can be done without the use of APL symbols and without knowledge of APL beyond that needed to load a work space. Data entry and correction can be done on terminals that do not display APL characters and can be done by clerical personnel with little instruction.

The system operates under the control of FILEDATA, a user interface function to be described in more detail later. The interface function presents a menu of options, executes the user's request, and presents the menu again. In the present system, the options on the menu are functions. This design simplifies the interface function and makes it easier to use the same option with more than one interface function. The design requires that the menu functions match the conventions of the interface functions. In the present case, these conventions require that the menu function be niladic, be named with no more than four letters, and use the global variable CL as a transfer variable carrying a list of file component numbers obtained from the user by the interface function. APL programmers will note that new options can be added to the present system by observing these conventions.

The system runs on a Sigma 9 under the CP-V operating system. It also runs under CP-6, offered on some Honeywell computers. Points that might need special attention in transfer to other systems are noted in the text or in the accompanying figures.

OVERVIEW: USER DOCUMENTATION

To provide an overview of the system from the user's standpoint, we present a description in the form of user documentation for the function FILEDATA and its menu. This kind of documentation is provided on-line as a menu option. Functions to support on-line documentation will be given in a later report. The documentation assumes that the system has been stored in the work space ENTRY with an automatic start of FILEDATA on loading. The documentation also assumes a basic knowledge of APL file systems and enough familiarity with APL to load a work space. The system is operated with the computer set to use APL characters, even if the terminal does not have them. If the terminal has only the ASCII character set, the messages from the computer will be in lowercase and the user should type commands in lowercase.

Instructions for Data Entry

To use the data entry system, load the work space ENTRY. It will respond, "FILE:" Type the name of the file you want to put the data in. You can use any mixture of letters and numbers to name the file, up to about

0005-7878/82/050475-07$00.95/0

eight characters. It is good practice to use four to seven characters and to see that the first three characters are enough to distinguish the file from all other files in your account.

If data are to be added to an existing file, type the name of the file. Or hit "return" if you need to be reminded of the file name, and you will get a list of all the files in your account. When the menu is presented, this list can also be obtained by typing FLST.

If the file already exists, the system will access it and report the number of components (i.e., the number of data rows) in it. If it is a new file, the system will create it and report zero components.

The system will then present a menu of options for you to select from: END, FLST, ENTE, FIXD, RSHO, CSHO, TBLE, LABL.

Type the option you want. If you make a typographical error, the system will either guess which option you want or present the menu again. After the system completes work on the selected option, the menu will again be presented, unless the option is one that implies your task is complete.

ENTE helps you enter data, one row at a time. A row would normally represent some reasonable grouping of data, such as the responses from one subject in one experimental condition or the responses of one subject on a questionnaire. It is a good idea to start each row with an ID number in Column 1. Although the remaining columns can be used for data, it is generally best to reserve one or more of these initial columns for data identification. For example, a set-designating number standing for an identifying property of the data row, such as an experimental condition or demographic property, is frequently placed in Column 2.

If you have multiple observations on the same subjects, a good practice is to use ID numbers that distinguish the data rows but retain the subject identity, for example, 101, 102, 103 for subjects in Condition 1 and 201, 202, 203 for the same subjects in Condition 2. If you plan to organize the data into several tables, this organization can be made easier by including a table indicator column with an integer indicating the table in which the row belongs. Normally, you will want to form tables representing homogeneous data sets, that is, representing separate experimental conditions or subject characteristics. The set-designating column mentioned above can also be the table indicator column, if desired, or two different columns can be used. These two practices make it easy to form tables and have the subjects in corresponding rows across tables.

If the file is new, ENTE will ask for the number of entries per row. If there are already entries in the file, the number of entries in the first row will be reported. In either case, ENTE will monitor new rows for conformity to this number.

ENTE will then advise, "TO STOP ENTERING, TYPE END:" Type the data, spacing between each number: 23 9.5 907.2 10. If you have negative numbers, precede each negative number with the uppercase 2, which appears in APL as the flying minus. It is not necessary to complete the row on one line. Each time you hit "return," ENTE will check the number of entries. If the row has the correct number of entries, it will be filed. Otherwise, the system will allow you to continue entering data, file the row as is, or cancel the row and type it in again.

At the completion of each data row, ENTE asks for more data. If you type END, ENTE will end the process by giving you a list of any components filed with the wrong number of entries. That list is also stored under the name ERRORS for use with other menu options, FIXD, CSHO, and RSHO.

CSHO and RSHO display components in a convenient format. When one of these is selected, the system asks "LIST COMPONENTS OR ALL:" You can list component numbers or type ALL to show the entire file. To list a series of numbers, expressions such as 9 to 22 or 17 to 35, 50 to 60 can be used. The system interprets these expressions the same way as the user does. If ENTE has stored something in ERRORS, ERRORS can be typed to show the components listed there.

The difference between CSHO and RSHO is in the arrangement of the display. RSHO presents the file components as rows of a table. CSHO presents the components as columns. The arrangement that best suits your purpose can be selected. CSHO, for example, is convenient for displaying questionnaire data.

RSHO will ask for a format that tells it how to space the columns and handle decimal points. Here, the FORTRAN formatting rules can be used. If the user is not familiar with these rules, he/she can generally use I6 for integer data or F8.2 for data with decimal fractions.

FIXD allows you to fix errors in the data. As with CSHO and RSHO, you will be asked to list components. You can list component numbers, type ALL, or type ERRORS if ENTE has stored something under that name. FIXD will then get each component in turn, present the component number and the first entry to identify it, and let you make corrections by string replacement. The function will ask, "REPLACE WHAT:" Enter a unique string of numbers including the error. The function will then ask for the replacement; type in the correct string. If the first string is not unique, the function will offer to replace all instances, but it will make no change without your approval.

When you type END, the function will let you show the revised component and return to fixing if you wish. Or you can have the revision filed and go to the next component you asked to fix.

TBLE, another menu option, organizes a file containing data rows as components to form a new file in which selected components are rows of tables. The name of the new file is formed by putting the letter P in front of the original data file name. This trick makes it easier for you to return to the same tabled file if all tabling is not done at one time.

TBLE allows for the selection of components for

tables in two ways. Manual selection lets you specify the component numbers of the rows to be included in a given table. You are asked to list the desired components or ALL. If you choose the manual selection, respond to this request with a list of component numbers to be used to form the first table. Then return to TBLE and repeat the process for each additional table.

Alternatively, the system will select components for you, using a table indicator column established when you entered the data. To use this option, enter ALL when asked to list components. TBLE will later ask you to enter the column number for sets or NONE. For manual selection, enter NONE. For automatic selection, enter the column number of the table indicator column you established. The function will display the number of rows identified by each different integer in the table indicator column. (This process may take several minutes if the file is large and the computer is busy.) The function will then ask for set numbers to select the components to be included in the first table. You may list one or more set numbers for each table. The step will be repeated for additional tables until all set numbers have been used.

TBLE will also ask if you want it to round the data to integers. Integer form saves space and processing time. If the data are integers, it is good practice to accept this offer.

In selecting components to form a table, TBLE will report any components with the wrong number of entries and exclude them from the table. It will also reorder the rows of the table so that the numbers in Column 1 are in ascending order.

When tabling is finished, TBLE will offer to label the file. You should label the file only after you have deposited all tables and are ready to start analyzing the data. The label process adds three components at the end of the file. The first carries the title, which the user is asked to supply. The title is two lines of text used by processing functions to title the output. The second is a scale table giving the approximate data ranges for each column; the scale table is used by plotting functions to see that a given column is plotted to the same scale regardless of the table it comes from. The third component is a table of contents carrying standard names for the columns and data tables in the file. If you are familiar with APL, you can change these names to apply specifically to your data. (The table of contents is an array. Be careful to leave Column 1 of the array blank.)

LABL offers the same labeling process as does TBLE, but under the assumption that tabling has already been completed. LABL is needed if you do not label the tabled file at the time you finish tabling. Since LABL operates on the tabled file, it needs a special step. Load a new copy of ENTRY. When it asks for the file name, give the name of your tabled file, the one formed by TBLE.

```
  Z←FLST
  →OT×⍳0≠⍴CL←(∨/CL∊'1234567890')/CL
  CL←⌶29
  OT:CL,' FILES, BE PATIENT'
  Z←0 11↓FLIB CL
 ∇

  T←TY
  ST:⍞T←' ',,1⌽'I3'ΔFMT(3⍴100)⊤⌶25),' - FILE:'
  →TI×⍳2<⍴DAFL←T←T,⎕←(0=⍴T←⍞)/DAFL
  T←⎕←FLST
  Δ,'FILE:'
  →ST×⍳0=⍴T←,T[T INTR⍞;⍳
  TI:TYPE←1,TB←ΔWK←STR←ΔTR←0⍴T←⍴DC←ALL←⍳ΔLM←(DAFL←(T≠' ')/T)FATIE⍳8
 ∇

  P←M INTS W;J;Q;V
  →FM×⍳~∨/Q←W∧.=⍉P←((1↑⍴M),J←V←⍴W←1⌽' ',(W≠' ')/W)↑M
  →0,⍴P←,Q⍳1
  FM:Q←P=V←⌈/P←(2×M[;1]=1↑W)++/(P∊W),-|J-1++/M≠' '
  P←(0≠⍴⍞←((J>V)∨1≠+/Q)/W,' DOES NOT COMPUTE')↓Q⍳1
 ∇

  K←I TO J
  K←(¯1↓I),(¯1↑I)↓0,⍳1↓J),1↓J
 ∇

  BOP J;S;Q;P;K;V;L
  →WI×⍳52≥⍴L←S/⍳⍴S←Δ=O←O,Q←Δ,V←''
  LN:V←(P←L[K←52⌊⍴L])↓O
  S←Δ=O←P↑O
  L←(L←K↓L)-P
  WI:K←0⍴P←1↓S←S/⍳⍴S←1,S
  →FI×⍳1≥+/(OWD+1)<P-¯1↓S
  BO:K←K,⍴O←Q←0⍴P←O
  ST:O←O,((OWD⌊S←¯1+P⍳Δ)↑P),Δ
  Q←Q,(OWD↓S↑P),Δ
  →ST×⍳0<⍴P←1↓S↓P
  FI:S←Δ,ΔWK,DAFL,', ',(0≠⍴K)/'SIDE ',2⌽,'I4'ΔFMT⍴K
  S←S,ΔTR,STR,(' CTR=',,'F4.2'ΔFMT CTR),DT
  O←S,Δ,TB,Δ,O
  (0=⍴⍞←⍎(0≠⍴OPFILE)/'O FAPE 2')/'NO OUTPUT FILE'
  →EN×⍳0=⍴⍞←(TYPE=1)/'PUSH RET'
  (TYPE←'N'≠1⍴⍞,Δ)/O
  EN:→BO×⍳~∧/Δ=O←Q
  →LN×⍳~∧/Δ=O←V
 ∇

  END
  →
  ⍝MONADIC ⍎ IS THE EXECUTE FUNCTION
  ⍝Δ IS CARRIAGE RETURN
  ⍝ΔFMT IS ⎕FMT IN SOME SYSTEMS USING ⎕ FOR SYSTEM VARIABLES.
 ∇
```

Figure 1. APL code for support functions FLST, TY, INTR, TO, BOP, and END. Comment lines on END apply to all functions in this report.

SUPPORT FUNCTIONS

The following support functions provide amenities that simplify the use of APL functions, especially for inexperienced users. APL code for these functions is given in Figure 1. These functions use several functions described in previous reports: REPLACES (Evans, Gage, Neideffer, 1980), LABL, FLAST, and FAPE (Evans, Neideffer, & Gage, 1980), and FATIE (Evans et al., 1981). The commonly available file functions, FREAD and FREPLACE, are also used. These are described under the names quad-FREAD and quad-FREPLACE by Gilman and Rose (1976).

Z ← FLST

FLST gives a list of the files in the user's current account. It is a simplified version of the commonly available function FLIB or quad-FLIB, as described by Gilman and Rose (1976). FLST is simple because it requires no right argument and generally assumes that the listing of files is for the current account. FLST is intended to be called by some other function, FILEDATA in the present case, that provides a value in the variable CL. Unless CL contains character digits, FLST gives a list of the files in the user's current account. This use is illustrated in the functions given later in this report.

If CL contains character digits, FLST treats the contents of CL as an account number and gives the list

of files in that account. Although the functions in the present report do not make use of this feature, functions to be given in a later report will do so.

Programmers should note that FLST drops the first 11 columns from the result of FLIB. In CP-V APL, these columns contain the account number, which is not needed and would interfere with some uses of the result. The location of the account number may differ in other systems, and FLST could be revised accordingly.

FLST gets the user's current account number by invoking the I-beam 29 function. This function is available in processors compatible with APL/360. Processors using quad variables and functions, as illustrated in Gilman and Rose (1976), would probably provide the account number in the system variable, quad-AI. These processors may also accept the I-beam function. The result from either source may vary with systems and may have to be adjusted to meet the requirements for the right argument of FLIB.

Depending on the host operating system, the standard forms of the account number may be either numerical or character data. For example, CP-V uses letters in its account numbers, requiring a character representation in APL. In adapting FLST to a system with account numbers in numerical form, the test to recognize the contents of CL as an account number would have to be changed. The test might be based on the magnitude of the account number. In our functions, no other use of CL is likely to result in numbers greater than 10,000.

TY

TY provides a standard and convenient procedure for accessing a main file. It is invoked by a user interface function designed to operate on a file. TY requests the name of the file. The user can enter a file name or hit "return." If the user enters a file name, TY creates the file, if necessary, makes the tie to the number 1, stores the file name in the global variable DAFL, and stores the date in the global variable DT, making these items available for use by other functions.

If the user hits "return" without entering a file name, TY checks to see if there is a previous file name stored in DAFL. If so, it makes the tie again and displays the name of the file. From the user's standpoint, the work space remembers what file was in use. If several functions using TY are in the same work space, the user can move from function to function without having to reenter the file name.

If a new copy of the work space has been loaded, assuming it was initialized by PARAM (Evans et al., 1981), DAFL will be an empty vector. In that case, the user can hit "return" without entering a file name and obtain a list of the files in the current account. After presenting the list, TY again asks for the file name. In this case, it compares the entry with the actual list, corrects any minor typographical errors, and makes the tie. If TY cannot recognize the file name, it will again ask for a file name rather than create a new file.

Since TY routinely appears at the beginning of all processing functions, the last line is used to set several global variables that control other parts of the system. Some of the variables are used in functions already presented; others are used in functions to be presented in later reports.

TY uses the I-beam 25 function to obtain the date. The comments on I-beam 29 in connection with FLST apply to this function also. In processors using quad variables, the date would probably be provided by the system variable quad-TS. TY expects the result to be a six digit number of the form MMDDYY and applies a decode operation to break it up for formatting. The result from quad-TS would probably be in a different form and require minor revision of the code at this point.

P ← M INTR W

INTR is a simple pattern recognition function to handle typing errors in menu or list selection. It compares a series of letters in the right argument, W, with an array in the left argument, M, carrying permitted entries as rows. If the entry matches a permitted entry, INTR delivers in P the row number of the permitted entry. If an exact match is not found, INTR measures the difference between the entry and each permitted entry, weighting agreement on first letter 2 points and agreement on the presence of any letters (regardless of location) 1 point each. A difference in length is weighted negatively.

If there is a single entry with the closest match and if the positive features outweigh the difference in length, INTR produces as a result the row number of that entry. This value can be used (see TY and FILEDATA, elsewhere in this report) to select a permitted entry. If no acceptable match is obtained or if more than one entry gives the same match, INTR reports that it cannot recognize the input and delivers an empty vector as its result. The calling program can test for an empty vector and return to the entry request.

The features and decision criteria for INTR were suggested by informal examination of recorded typographical errors made by users of menu-driven programs. No doubt, INTR could be improved by a more systematic investigation of error patterns. The assistance provided by the present version, however, seems to be welcomed by our users.

K ← I TO J

TO assists users in creating lists of numbers such as might be needed to select column numbers or file components for a table. TO creates the consecutive integers from its left argument, I, through its right argument, J, delivering the result in K. Thus the user can respond to a request for component numbers with an expression such as 8 to 12. The expression is interpreted as the numbers 8, 9, 10, 11, and 12.

Both arguments can contain multiple elements. TO treats this condition as a request to embed its string of consecutive integers at the point at which TO was inserted in the larger string. The integers generated range from the last element of the left argument to the first element of the right argument. Because of this feature, TO can be incorporated into a series of numbers without the parentheses that APL would otherwise require. The user can enter 1 3 4, 7 TO 15, 18 19, 25 TO 30, 34. The list will be interpreted as people would interpret it. The commas are optional.

BOP O

A revision of the previously presented output function, BOP (Evans, Gage, & Neideffer, 1980), is given here because one of the added features is required to accommodate functions presented later in this report. As described previously, BOP is a general-purpose output function for character vectors carrying the results of data analysis. It presents the character data delivered in O, putting certain identifying information as a header on each unit of output to insure adequate identification.

BOP normally pauses for a carriage return before delivering output, so that the user can make any needed adjustments at the terminal before accepting the output. This pause and the output to the terminal can be suppressed at any time by typing NO in response to the request for a carriage return. The suppression of output continues until the global variable TYPE is reset to 1. In the group given here, TYPE is reset by TY each time the main function is invoked.

As in the previous version, BOP limits each unit of output to the width specified in the work space parameter, OWD. Output wider than this value is separated into two or more parts. The present version provides for easier collation of these parts by labeling them "SIDE 1," "SIDE 2," and so on. The present version also limits the number of lines in a unit of output to 52, not counting the header. Successive units are produced as required. This modification improves the handling and identification of a long series of rows, as may be produced by data display functions presented later in this report. For CRT terminals, users may prefer to change Lines 1 and 2 of BOP to limit the number of lines to 20 or 25.

The contents of the header have also been modified to present the name of the processing function, the file name (stored in DAFL by TY), and the side identification, if needed, on the first line. The name of the processing function must be stored in the variable delta-WK by the function that handles the menu. This feature is used in conjunction with a text processing function that summarizes a file by giving the first line of each component. With the key information on the first line, the list of first lines serves as a table of contents for the output file.

BOP also does its part to insure that each unit of output carries information about parameter settings, time of processing, and study title. Lines 12 and 13 put several global variables at the head of the output. These are CTR, DT, STR, delta-TR, and TB. In our system, CTR is a work space parameter that specifies a threshold for suppressing the display of small correlations. TB normally carries the study identification obtained from one of the components of a labeled file. The use of these variables is illustrated by Evans, Gage, and Neideffer (1980).

DT carries the date, specified by TY as described earlier in this report. STR is intended to carry a report of the set numbers and set column if set selection was operating at the time of processing. Set selection is illustrated by Evans et al. (1981). Delta-TR is intended to carry a report of table numbers identifying the data tables used in processing. In our system, STR and delta-TR are set by menu functions that also aid the user in set and table selection. These functions will be presented in a future report.

TB, delta-WK, STR, and delta-TR are set to empty vectors by TY. Thus, functions that have no reason to define them need not define them. This resetting also insures that values stored in the variables by previous functions cannot be carried over to appear as spurious labeling on new output.

Unlike the previous version, this version of BOP does not require an output file. The global variable OPFILE, which normally carries the name of the output file, can be an empty vector. In that case, BOP does not attempt to file the output but reports the absence of an output file. This feature accommodates novice users who forget to empty the output file until it fills the space allocated for the account.

END

END provides a recognizable menu item for ending a menu-driven function such as FILEDATA. As presented here, END merely ends the function. Programmers may adapt it to the needs of a particular installation, however, by inserting instructions to be presented to the user before ending the function or by inserting a system command such as)OFF if the APL interpreter permits the execution of system commands. If data entry is done by inexperienced users, the message DON'T FORGET TO SIGN OFF might be sent. For more experienced users, the message might be TO RESTART, INVOKE FILEDATA.

FUNCTIONS FOR DATA PREPARATION

This set of data preparation functions includes a user interface function, FILEDATA, with processing functions to provide for data entry, display, correction, and organization into tables. User interactions with this function and its menu options are described in the earlier section on user documentation.

FILEDATA

FILEDATA is a user interface function that integrates all the data preparation functions into a single process. FILEDATA obtains the name of a file, pre-

```
FILEDATA;ALL;T;CL;JBL
TY;' COMPONENTS'
NONE←CL←⍳0
MNU:,' ',JBL←8 4⍴'END FLSTENTERSHOCSHOFIXDTBLELABL'
→MNU×⍳0=⍴T←JBL INTR ∆WK←⍞
→EX×⍳4>T
'LIST COMPONENTS OR ALL'
CL←,⎕
EX:⍎,JBL[T;]
ALL←⍳FLAST 1
→MNU
∇
```

```
ENTE;Q;END;C;F;ER;E;L;CANCEL;FILE;D;R
FILE←F←'F',0⍴CANCEL←C←'C'
→OL×⍳0≠L←FLAST 1
'HOW MANY ENTRIES PER ROW (COMP):'
→GO,R←⎕
OL:,R←⍴FREAD 1 1;' ENTRIES PER ROW (COMP)'
GO:ER←E←END←⍳0
'TO STOP ENTERING, TYPE END OR E'
EN:L+1,Q←''
→ND×⍳0=⍴D←,⎕
CK:→FI×⍳R=D←⍴Q←Q,D
'ENTRIES: ';D;(R>D)/' CONTINUE OR';' TYPE FILE OR CANCEL:'
→¯1↑CK,('FC'=1↑D←⎕)/FI,EN
FI:→EN,ER←ER,(R≠⍴Q)/L←Q FAPE 1
ND:'ERRORS IN ';,ERRORS←ER;(0≠⍴ER)/∆,' STORED IN ERRORS'
∇
```

```
RSHO;Q;V;G
'ENTER FORMAT (EG. I4 OR F8.2):'
G←0⍴Q←⍞
TB←'COMPS '
PS:G←G,∆,(,'I4'∆FMT 1↑CL),'|',,Q ∆FMT FREAD 1,1↑CL
→PS×⍳0<⍴CL←1↓CL
BOP G
→PS×⍳0<⍴CL,G←⍳0
∇
```

```
CSHO;D;C;V;F
'FORMAT: (EG, I5,F8.2)'
F←⍞
ST:D←0 0⍴C←(T←14⌊⍴CL)↑CL
GC:D←((⍴D)⌈⌽0,1↑⍴V←F ∆FMT V,FREAD 1,V←1↑C)↑D
D←D,((1↑⍴D),1↓⍴V)↑V
→GC×⍳0<⍴C←1↓C
D←,∆,D[1;],∆,,'-',[1]∆,∆,1 0↓D
BOP D
→ST×⍳0<⍴CL←T↓CL
∇
```

Figure 2. APL code for data preparation functions FILEDATA, ENTE, RSHO, and CSHO.

sents a menu, accepts a requested operation with correction of minor typographical errors, obtains the user's choice of component numbers if applicable, executes the operation, and presents the menu again. APL code for this function and for the data entry and display functions is given in Figure 2.

ENTE

ENTE supports data entry. It is intended to create file components that will be combined as rows of a table by TBLE, described later in this report. The data values are entered in the evaluated quad mode, allowing the APL interpreter to check the input for proper form as a list of numbers. If an error is detected at this point, APL reports the error and gives immediate opportunity for correction. ENTE itself checks the entry for conformity to the specified number of elements per row, as described in the user documentation. As each row is completed, ENTE appends it as a new file component. If ENTE is reinitiated for work with a partially completed file, it continues appending where it left off.

RSHO and CSHO

RSHO and CSHO display the contents of a file, such as that produced by ENTE, in which the components are numerical vectors. When one of these is invoked, FILEDATA requests a list of the components to be displayed. (The list is transferred via the global variable CL.) The response can be any APL expression. The components to be displayed need not be of the same length. RSHO requests a format specification, to be given in the same notation used for FORTRAN formats. This format allows the user to choose integer or floating-point output and to determine the number of characters allocated to each number. RSHO presents each component as a row, with the component number in front. CSHO presents each component as a column, headed by the component number. CSHO breaks up the components to display no more than 14 components/page. With an I5 format, the result will be 70 characters wide. Since the output is handled by BOP, results too wide or too long for good presentation on the terminal are separated into parts.

FIXD

FIXD supports correction of data filed as numerical vectors. Components are selected as described for RSHO. APL code for FIXD and the tabling function to be described next is given in Figure 3.

TBLE

TBLE permits the user to convert files in which the components are numerical vectors into files organized into tables, labeled and ready for processing by functions such as PRINAN (Evans et al., 1981), GRAPHICS (Evans, Neideffer, & Gage, 1980), and others to be presented in this series. The components of the raw data file become rows of the tables.

When TBLE is invoked, it invents a new file name by putting the letter P in front of the name of the raw data file. With the aid of FATIE (Evans et al., 1981), TBLE creates a file under the new name, if necessary, and ties to it. (The name of the raw data file must be less than the maximum length permitted by the system.) This file receives the tables as they are formed and is intended to become the processing file.

Programmers should note that in forming the name for the new file, TBLE takes the contents of DAFL up

```
FIXD;Q;TXT;END;E
'TO STOP FIXING A COMP, TYPE END OR E',E←END←⍳0
FX:'COMP.';1↑CL;' STARTS ';2↑TXT;4⍴' ';⍴TXT←FREAD 1,1↑CL;' ELEMENTS'
RP:'REPLACE WHAT:'
→PT×⍳0=⍴Q←,⎕
'BY WHAT:'
→RP,0⍴⎕←⎕ REPLACES Q
PT:,⍴TXT;' ELEMENTS - SHOW?'
→FL×⍳0=⍴⎕←('Y'=1↑⍞)/TXT
→RP×⍳'Y'=1↑⍞,⍞←'MORE?'
FL:TXT FREPLACE 1,1↑CL
→FX×⍳0<⍴CL←1↓CL
∇
```

```
TBLE;Q;C;T;V;P;RN;S
1↓⍴Q←(TS←0,⍴FREAD 1 1)⍴0;' ELEMENTS PER ROW, TABLES IN'
,T←(⎕←RSF←'P',(¯1+⌊/DAFL⍳'. ')↑DAFL)FATIE 2;' COMPS'
→UL×⍳0=T
→UL×⍳0=⍴⎕←(' '=1⍴FREAD 2,T)/'LABLED FILE, CAN''T ADD TABLES'
END
UL:RN←'Y'=1↑⍞,⍞←'ROUND TO INTEGERS?'
'ENTER COLUMN NUMBER FOR SETS OR NONE'
→GE×⍳0=⍴V←S←C←,⎕
'    SET     FREQ.',P←⍳0
GA:→BP×⍳C>⍴S←FREAD,1,1↑CL
P←P,S[C]
BP:→GA×⍳0<⍴CL←1↓CL
T←1,1+(V≠¯1⌽V←P[⍋P])/⍳⍴P
'I6'∆FMT(S←V[T]),[1.5](1↓T,1+⍴P)-T
AT:'SETS: ';S;∆,'ENTER SETS FOR TABLE, OR NONE'
→ND×⍳0=⍴V←,⎕
→AT×⍳0≠⍴⎕←(0=⍴CL←(P∊V)/⍳⍴P)/'EMPTY SET'
GE:→SK×⍳0≠⍴⎕←(0≠⍴⎕←(TS[2]≠⍴C←,FREAD 1,1↑CL)/1↑CL)/'WRONG LENGTH'
Q←Q,[1]⍎(RN/'⌊.5+'),'C'
SK:→GE×⍳0<⍴CL←1↓CL
∆;(⍴Q);' TABLE IN ',RSF,' ';Q[⍋Q[;1];]FAPE 2
→AT×⍳0<⍴S←(~S∊V)/S,,Q←TS⍴0
ND:→0×⍳'Y'≠1↑⍞,⍞←'LABEL?'
'YOUR PRIMARY DATA FILE IS NOW ',DAFL←RSF
0⍴DAFL FATIE⍳9
LABL
∇
```

Figure 3. APL code for data preparation functions FIXD and TBLE.

to, but not including, the first occasion of period or blank. These are standard separators between file names and account numbers in CP-V. A separator and account number appear in DAFL only if the raw data file is in an account other than the user's current account. If present, these must be removed to cause FATIE to create the new file. For a system in which other separators are used, Line 2 of TBLE would need to be changed to include the other separator in the character vector. Substitution in or addition to this vector can be made without other changes in the function.

Before filing a table, TBLE reorders the row to put Column 1 in ascending order. This ordering is the point of peak space requirements for the functions in this group. For systems limited to the traditional 32-KB work spaces, programmers may want to sacrifice this feature. The reordering is done at Line 21 in forming the left argument of FAPE. The reordering code can be replaced by Q without other consequence to the function. In general, however, if a table is too large to be reordered, it will also be too large to be processed by functions such as PRINAN in the same work-space limits.

As part of the labeling operation, TBLE sets the global variable DAFL to carry the name of the labeled file. This action is done on the assumption that the next operation will be some analysis of the labeled file. In our system, all processing is done in the same work space, with processing groups being brought in and erased under the control of a master function. With this arrangement, the values of global variables such as DAFL remain available to control new functions as they are brought into the work space.

SUMMARY AND DISCUSSION

The functions presented here provide for efficient entry and organization of data sets comparable in size to those that are commonly processed with data analysis packages. Future reports will describe functions that can analyze data sets of this size, that is, data sets with essentially no restriction on the number of cases. Most of these analyses can be done in the traditional 32-KB work space, provided the number of variables is not too large and the number of rows in each table is limited to 30 or 40. These functions provide many of the advantages of a standardized data analysis system while retaining the advantages of operating in the APL environment.

REFERENCES

EVANS, S., GAGE, F. H., & NEIDEFFER, J. D. APL programs for interactive data analysis: Correlation and data entry. *Behavior Research Methods & Instrumentation*, 1980, 12, 372-375.

EVANS, S., NEIDEFFER, J. D., & GAGE, F. H. APL functions for interactive data analysis: Graphics and labels. *Behavior Research Methods & Instrumentation*, 1980, 12, 541-545.

EVANS, S., NEIDEFFER, J. D., & GAGE, F. H. APL functions for interactive data analysis: Principal components analysis. *Behavior Research Methods & Instrumentation*, 1981, 13, 657-666.

GILMAN, L., & ROSE, A. J. *APL: An interactive approach.* New York: Wiley, 1976.

(Received for publication February 8, 1982;
revision accepted August 9, 1982.)