The data-text system—An application language for the social sciences

by D. J. ARMOR

Harvard University
Cambridge, Massachusetts

INTRODUCTION

The enormous growth of special applications software has been the subject of much speculation and debate. While many of the developments are natural expansions into new application areas, many computer specialists argue that applications programming is a prime example of the tendency to reinvent the wheel. No area receives this accusation more than that of statistical packages or languages, particularly those developed for applications in the social and behavioral sciences. A recent study by Hugh F. Cline shows that in 130 universities which grant higher degrees in these fields, there are over 170 such packages or languages in use. This is more than one per university! No one could possibly maintain that all of these systems are unique; the duplication of effort has been rampant.

Now that I have said that, I want to take the other side. While I cannot defend outright duplication, I want to present some of the reasons why many of these developments were necessary and why they represent a significant stage in the evolution of computer software. In illustrating my argument, I will draw upon examples from the Data-Text system, a data processing and statistical analysis language developed at Harvard. Although other systems could be used to make the same points, Data-Text may represent the most comprehensive attempt in this field to date. Besides, since I helped to develop it, I know it better than others!

EARLY DEVELOPMENTS

One insight into the rationale behind Data-Text and other similar attempts can be gained by considering the origin of languages like FORTRAN. Why was FORTRAN developed and why did it become so popular? All modern computers have had symbolic machine or assembler languages since at least the IBM 704 generation. Why did the symbolic machine languages fail to suffice for all application software, being supplanted almost exclusively by FORTRAN for problems involving numerical computations? In retrospect the answers are obvious. First, programming in a machine language requires a good deal of detailed knowledge of the internal structure of the machine; second, the atomistic nature of the instruction set means a great deal of programming effort even for simple mathematical expressions—not to speak of the coding required for routine input-output procedures. For the engineer or scientist with a particular type of mathematical expression to evaluate, the investment of time and energy in learning and programming in machine language was costly. FORTRAN and other languages like it solved this problem for many quantitative applications in the physical sciences. FORTRAN was relatively machine independent (at least by the end of the 1950's) and could be learned very quickly by those with some background in mathematics. Most mathematical formulas could be expressed in a fairly natural way, and most of the troublesome clerical problems of I/O were handled with a minimum of complexity (at least in comparison with machine language solutions). In other words, the FORTRAN-type languages were "natural" languages for those programmers working in numerical mathematical applications. The same can be said of many other languages, such as COBOL in the business realm and SNOBOL in the field of text processing.

For many other fields, however, FORTRAN is not a "natural" language. In the social and behavioral sciences, where mathematical proficiency is not predominant, a great many statistical analyses are commonly used even though their mathematical bases are not fully comprehended by the user. Two examples are the techniques of regression analysis and factor analysis. The first requires inversion of a matrix of correlation coefficients, and the second requires extraction of eigenvalues and eigenvectors. Although a competent
social scientist can understand the results of these analyses, very few actually understand the mathematical techniques which are required to derive the results. (In defense of the social scientist, I should hasten to add that a great number of physical scientists I have known are likewise in the dark.) Even in the case of less complex statistical methods, many researchers are not able to provide precise computational formulas without careful review of a statistics textbook.

The result of this is that the social scientist finds it difficult to program in a language which requires that he provide the mathematical or statistical formulas for all of his ordinary analyses. Thus languages like FORTRAN, ALGOL, BASIC, PL/I, and COBOL are not natural languages for the social scientist. I think this argument can be extended to other fields as well where standard algorithms exist for data analysis problems. In these cases, if a language is to earn the label “natural,” it should embody terms or macros which call in these entire algorithms, just as FORTRAN calls in a routine in response to a function name such as SQRT or LOG. In other words, FORTRAN and similar languages are natural languages for the programmer who wants to implement a statistical or mathematical algorithm, but they are not natural languages for those who simply want to apply those algorithms to analyze a particular set of data.

I do not want to give the impression that the only limitation of FORTRAN-type languages has to do with statistical algorithms. There are a variety of other data processing problems which are commonplace in the social sciences which can be extremely tedious to program in FORTRAN. One example is the problem of missing observations. These occur whenever a variable cannot be measured for a given subject or some other unit of analysis. There are a variety of standardized procedures which a social scientist follows to deal with this problem but no general language that I know of has implemented them. Another problem has to do with a variety of fairly common transformations which social scientists apply to their variables but which do not exist in the FORTRAN-type languages. Perhaps the most common of these is what we call “recoding” of an original measurement. This involves transforming the original values of a variable into some arbitrary set of new values, as when we want to collapse a continuous-type variable into one with a small number of values or categories. Although these kinds of transformations can be coded in a FORTRAN-type language, they can frequently lead a fairly lengthy and repetitive program. As we shall illustrate, a reencoding operation in Data-Text stands in the same relationship to the necessary FORTRAN steps as an algebraic formula in FORTRAN stands in relation to the necessary steps in a machine language. The same can be said for a number of other transformational capabilities commonly available in the more popular social science languages. Finally, there are a number of other data processing and routine clerical problems which are not easy to handle without special languages. Included are full labeling of variables and their values (or categories), input of multiple-punch data, selecting or sampling sub-sets of the input data, and controls to classify the input data into an arbitrary number of sub-groups.

The earliest attempts to solve some of these problems for the social scientists led to the development of “canned” programs. By “canned” program I mean one written—usually in FORTRAN—to perform a particular statistical analysis, but written so that it can be used over and over again by different researchers with different data. A user prepares “control” or “parameter” cards which specify the necessary information about his data (e.g., number of variables) and any options which are available; the control cards are used by the program to read in the data properly and to produce the desired results. Canned programs are still used widely by social scientists today, and for some applications canned programs continue to be the most efficient computational solution.

As canned programs became a popular method of computer analysis, collections or “packages” were put together at various locations (usually university research centers) and distributed to researchers at other locations. The oldest, most well-known and widely used of these collections is the BMD series, started by Dixon and his associates at UCLA. The BMD collection for the IBM System/360 contains over 40 programs for almost every type of standard statistical analysis found in applied statistics textbooks. Another collection for the IBM System/360 is the OSIRIS package. Developed at the University of Michigan, OSIRIS represents a significant advance in that the canned programs are “integrated” into one large system which recognizes a “standardized” data file. A file of data can be described and then any number of different statistical routines can be requested which use that file.

The canned program approach still leaves many problems unsolved for the social scientist. While canned programs have largely solved the problem of knowing complicated statistical algorithms, little relief is gained for the many other data processing problems which I have mentioned (special transformations, missing observations, labeling, and the like). Moreover, two new problems have surfaced. Both of these problems result from the fact that most canned program developers are not professional programmers, and therefore they are usually more interested in the statistical algorithm than in program elegance.
The first problem is that all kinds of arbitrary restrictions are placed upon various parameters in the problem—the number of variables allowed, the number of subjects, the number of values or categories of variables, the number of analyses, etc. I wonder how many hundreds of researchers have had a 110-variable factor analysis problem with a program which would accept only 100 variables; or a cross-tabulation requiring 60 cells with a program limited to 50? This would be like a FORTRAN programmer being limited to a total of 5 DO-loops or to arbitrary subscript limits of 50. Of course, any language has some limitations, but all good compilers do dynamic core allocation so that limits are encountered only when available memory is exhausted. Most canned program limits are due to programmer short-cuts such as fixed dimension statements.

Second, little effort is given to making the control cards "readable" to the analyst in the sense that FORTRAN is readable to a programmer. Most control cards consist of arbitrary numeric codes in fixed-format positions. It is practically impossible to remember the set-up procedures from one run to another without a program write-up literally in front of you. For the data analysts, having to "read" these control cards is somewhat analogous to a FORTRAN programmer having to read a hexadecimal dump in order to interpret the steps in his program.

THE DATA-TEXT SYSTEM

The Data-Text system was designed to solve some of these problems and limitations of the package approach. Data-Text is designed to be a "natural" language for the social scientist, just as FORTRAN is a natural language for those familiar with mathematical expressions. In this sense, I think Data-Text represents a further stage in the evolution of "problem-oriented" computer languages. With Data-Text, the social scientist can request complex data analyses as easily as the engineer can write complex formulas in FORTRAN.

The first version of the Data-Text system was designed by Couch and others for the IBM 7090/94 series, and it became operational in 1963-64. This version became widely used at more than 20 universities and research centers during the middle 1960s. A substantially revised version of Data-Text has been designed and programmed for third generation computers and a version for the IBM System/360 series became operational in the spring of 1971.

* The original Data-Text system was supported by grants from the National Science Foundation (#GS1427 & GS1424). The revised Data-Text system was made possible by a grant from the National Institutes for Mental Health (#MH15884) and by donated computer time from the Harvard Computing Center.

It should be mentioned that in the latter part of the 1960s other computer systems with similar design goals appeared. One example is the Statistical Package for the Social Sciences (SPSS), created by Nie and his associates at Stanford University and the University of Chicago. Other social scientists have begun experimenting with interactive (or "time-sharing") data analysis systems, such as Meyers' IMPRESS system at Dartmouth College, Miller's DATANAL system at Stanford, the ADMINS system at MIT, Shure's TRACE system at SCD and UCLA, the Brookings' Institution BEAST, and Kuh's TROLL system.

The unique aspects of the Data-Text system in relation to other computer systems is its ability to handle very complicated data processing problems and extremely intricate statistical analyses with a minimum of technical knowledge about the computer and a minimum of familiarity with statistical formulas and algorithms. The requirements for a user are: (1) that he have a concrete research problem which demands a standard statistical analysis of data; (2) that the data are recorded in some systematic format on IBM punched cards (or certain other media, such as magnetic tape or disc); (3) that he understands the type of statistical analysis required for his problem; and (4) that he knows the names of the appropriate statistical analysis procedures. Given such a situation, Data-Text can be used for defining or transforming input variables, for giving them appropriate descriptive labels, and for requesting a great variety of different statistical analyses. The variable definitions and statistical analysis requests are made in a flexible, easy-to-learn "natural" language which makes heavy use of terms familiar to most social science researchers. Data processing features include automatic missing observation procedures, input and manipulation of multipunched data, subgrouping controls, and many others.

A sample of the Data-Text language is probably the easiest way to introduce this idea and to show how Data-Text differs from FORTRAN-type languages. Assume that a researcher has collected data from a group of several hundred college students (the actual number is not important). The data might include measurements of age, sex, race, father's education, family income, college class, an ability test score, existing as well as answers to questions about school activities. We will assume that the non-numeric measures (for example, sex) have been given codes of some kind (for example, 0 for male, 1 for female) and that the data has been punched onto IBM cards with two cards for each person. The Data-Text instructions shown in Figure 1 define the set of original variables, derive some new variables as transformations...
of the original set, and request several types of statistical analyses. (In many cases these instructions are punched onto IBM cards, one instruction per card. In other instances, they might be lines typed at a remote typewriter console.)

The instructions fall into several types. The instruction

*DECK COLLEGE STUDENT STUDY

is a header instruction which signifies the start of a set of Data-Text instructions and which provides a title that will appear on every page of computer print-out.

The next two instructions relate to I/O operations:

*READ CARDS
*CARD(1-2) FORMAT/UNIT = COL(1-4), CARD = COL(5)

The READ identifies the mode of data input (punched cards), while the FORMAT provides information about the number of cards per student and some special identification fields. The term CARD(1-2) indicates 2 cards per subject; the term UNIT refers to a field in each card for identifying the unit of analysis (such as a subject number); the term CARD refers to a field for identifying the different data cards for each UNIT. Thus, this example indicates two cards per UNIT, a subject number in columns 1 to 4 of each card, and a card number in column 5. Although there is some syntactic similarity to FORTRAN in these two instructions, their effect is quite different. First, no looping controls are required around the READ instruction; all data cards will be processed automatically according to the other operations specified in the program. *Second, the FORMAT instruction in this ex-

Figure 1—A sample data-text program

* A special *SELECT . . . IF instruction is available for selecting sub-sets of the data file; see Figure 2.
ample only provides information about the number of cards per subject and the identification fields; variable fields are defined in separate instructions. Moreover, if the UNIT and CARD fields are specified as in the example, the cards for each subject are automatically checked for consistent subject numbers and card sequence (taking the first subject as the prototype). These checking operations are extremely important for some of the large-scale survey research studies.

The next set of instructions define the variables to be used in the statistical analysis requests. For example, the instruction

\[ *\text{SEX} = \text{COL}(7/1) \]

\[ \text{STUDENT'S SEX} (0 = \text{MALE} / 1 = \text{FEMALE}) \]

defines a variable named SEX which appears in column 7 of card 1 for each subject. It also gives a user-supplied descriptive label and, within the parentheses, it describes the values actually punched in column 7 and relates them to user-supplied category labels (e.g., 0 = MALE means that 0's were punched to represent males). The ability to specify labels and values enables an analyst to construct a true "machine readable" code-book for his data set. Moreover, the use of the variable and category labels is not confined to the instruction listing; they are also used in all computer print-out of statistical results which use the variables and/or their categories.

Alphabetic-coded variables can be indicated by preceding the COL-specification with the letter "A":

\[ *\text{RACE} = \text{ACOL}(8/1) \]

\[ \text{RACIAL BACKGROUND} \]

\[ (B = \text{BLACK} / W = \text{WHITE} / O = \text{OTHER}) \]

This definition indicates a variable which was coded in column 8 with the letter values B, W, and O.

Entire arrays of variables can also be defined without the use of explicit looping controls, as in the example

\[ *\text{ACTIVITY} (1-50) = \text{COL}(6-55(50)/2) \]

\[ \text{SCHOOL ACTIVITIES} \]

\[ (0 = \text{NO} / 1 = \text{YES}) \]

In this case the variable and value descriptions will apply to all 50 variables in the array. The example

\[ *\text{ITEM} (1-4) \]

shows how several variables can be defined with the same value descriptions but different variable labels.

The example in Figure 1 illustrates several of the special transformation capabilities of Data-Text. While the logical and arithmetic transformations used to create VAR(4) and VAR(6) resemble those available in most languages, the special functions MEAN, SUM and RECODE used in defining VAR(3), INDEX, and VAR(5) are unique. The MEAN and SUM are "summary" functions which operate across the variables for a given subject, so that an average score (for the MEAN) and a count (for the SUM) are derived for each subject in the sample. Again, no looping controls are necessary for these operations.

The RECODE function embodies a great deal of power. It works in conjunction with a code statement (*CODE(A) in the example) which specifies the re-coding equations. Each equation consists of a series of original values together with the new value they are to be assigned in the transformed variable. Thus, only two instructions are required to transform one set of values into any combination of new values. Although this kind of operation is programmable in FORTRAN, it requires a great many assignment, IF and GO TO statements.

One of the equations in the *CODE statement is

\[ 5 = \text{BLANK} \]

The term BLANK is a special constant in Data-Text which denotes a missing observation. This means that a subject with a score of 5 in VAR(2) will be treated as missing in VAR(5). BLANK values are automatically assigned to variables for subjects with blank columns in the input data fields. That is, if columns 5 and 6 in card 1 are blank for a given subject, then a BLANK value is automatically assigned to the AGE variable for that case. Regardless of where BLANK values come from, they are handled automatically in all transformation and statistical routines. A series of default rules exist which are appropriate to the type of transformation or the type of statistical procedure. For example, BLANK's are ignored by the MEAN function in the VAR(3) instruction, so that the average is taken only of non-BLANK values. In the VAR(6) instruction, however, which uses a regular arithmetic expression, the opposite is true: if either of the arguments ABILITY or AGE is BLANK for a subject, then the result is BLANK. In most of the statistical routines indicated by the COMPUTE instructions, the default is to ignore BLANK values; but options exist which will do other things (like excluding a subject from the analysis if any variable has a BLANK value). This automatic processing of missing observations has proved to be an extremely valuable feature in social science applications.

Before turning to the statistical controls, another instruction should be noted. The instruction

\[ *\text{CHANGE} (1-10) = \text{POSTEST}(1-10) \]

\[ - \text{PRETEST}(1-10) \]

illustrates what we call a "list" expression. The subtraction operator is applied element-wise to the two argument lists, so that each variable in the CHANGE array is the difference between the POSTEST and PRETEST variable in the corresponding list position.
 Lists of variables can be combined into any arbitrary arithmetic expression, and the expression is evaluated for each element in the list. This provides a very powerful transformational ability without the necessity of looping controls.

The COMPUTE instructions are commands for statistical analyses on the variables defined in the preceding instructions. The statistical procedure is requested by name, such as T-TEST or CROSSTAB or CORRELATION. Generally speaking, the only parameters required are the lists of variables to be included in the analysis and any special options which are desired. The instruction

*COMPUTE STATISTICS(AGE, INCOME, ITEM, VAR(6)), SKEW

will compute means, standard deviations, and the number of nonmissing observations for each variable in the list across the whole sample of students; the option SKEW will also cause a measure of skewness to be computed. The instruction

*COMPUTE CORRELATIONS(ACTIVITY), TEST

will cause a 50 by 50 matrix of product-moment correlations to be computed for the array ACTIVITY(1-50), and the TEST option will result in a statistical test of significance for each correlation.

While there is some resemblance here between the COMPUTE instruction and a subroutine call in many languages, the analogy cannot be pushed too far. First, the "arguments" include reference only to the variable names and not to the many other quantities and arrays which are used by the routine, such as variable labels, value description, number of categories, and so forth. Second, and more important, some of the statistical requests have special key words embedded in the variable list which must be encoded by a special compiler tailored to each statistical procedure.

For example, in the instruction

*COMPUTE REGRESSION(ABILITY ON AGE, SEX, VAR(1), INCOME), GROUP BY CLASS

the key word ON indicates the independent variables (AGE, SEX, VAR(1), INCOME) on which the dependent variable (ABILITY) is to be regressed. The GROUP option indicates the analysis is to be carried out on the four college class subgroups defined by the CLASS variable. In the COMPUTE ANOVA instruction,*

*COMPUTE ANOVA(SEX BY RACE BY VAR(4)), ABILITY

the key word BY indicates a 3-way factorial analysis of variance design with ABILITY as the dependent or criterion measure. In other words, the COMPUTE instructions are also expressed in a natural language which contain operators and a syntax meaningful to each particular analysis.

All of the statistical routines are almost literally without limitations. Correlation matrices are not limited to available core; a cross-tabulation can be $n$-ways, with no limit to $n$; a variable used for classification purposes in a cross-tabulation, an anova, or in the GROUP BY option can have any number of categories; and if all the analyses requested will not fit in available core, the data is automatically saved in internal form so that repeated passes can be made over the data to satisfy all the requests. These features reduce the frustrations often encountered with other statistical packages which have established arbitrary parameter limitations.

It is rarely the case that an analyst will obtain all of his statistical results in a single run. Sometimes the same data set will be analyzed several dozen times. Data-Text offers a method for saving all of the defined variables, labels, and data in a standardized binary file. Since no compilation, transformation, and data conversion needs to be done, the time required to process this file is substantially less than the time it takes to process the original raw data.* In our sample program, the instruction

*WRITE DATATEXT DISC 4

creates a standardized file for this set of data.

Input of a DATATEXT file does not preclude the use of the full transformational language. New vari-

* The term ANOVA is a fairly standard abbreviation for "analysis of variance."
* The savings range from 50 percent to 90 percent of CPU time, depending upon the size of the data file (the larger the file, the greater the savings).
that using a DATATEXT file results in a much shorter and simpler program.

PERSECUTION SYMPTOM (10) 13.021 5.814 256
PRESSURE OF SPEECH SYMPTOM (10) 10.820 1.401 256
DRUG TREATMENT DRUG 2.539 1.105 256
NON-AUDITORY HALLUCIN. SYMPTOM (13) 2.387 1.060 256
MEMORY DEFECT SYMPTOM (14) 2.258 2.126 256

ABILITY using the special collapsing function

DISORIENTATION SYMPTOM (2) 1.431 1.216 256
GUILT SYMPTOM (3) 12.413 1.181 256
SOCIAL CLASS CLASS 2.138 1.065 252

ORDER. Some new cross-tabulations are requested,

The following two variables are pre-treatment ratings of overall illness. They are scored so that a 7 means most sick

11 ILLNESS (2) = COL (1-2) = NURSES ILLNESS RATING (I-7=RANGE)
12 SYMPTOM (12) = COL (27-28) = DELUSIONS OF GRANDEUR
13 SYMPTOM (11) = COL (17-18) = SLOWED SPEECH
14 SYMPTOM (10) = COL (13-14) = AUDITORY HALLUCINATION
15 SYMPTOM (9) = COL (19-20) = INCOHERENT SPEECH
16 SYMPTOM (8) = COL (31-32) = NON-AUDITORY HALLUCIN.
17 SYMPTOM (7) = COL (33-34) = MEMORY DEFECT
18 SYMPTOM (6) = COL (25-26) = PRESSURE OF SPEECH
19 SYMPTOM (5) = COL (21-22) = PERSECUTION
20 SYMPTOM (4) = COL (15-16) = DELUSIONS OF GRANDEUR
21 SYMPTOM (3) = COL (9-10) = HOSTILITY
22 SYMPTOM (2) = COL (5-6) = GUILT
23 SYMPTOM (1) = COL (1-2) = HOSTILITY
24 SYMPTOM (0) = COL (0-0) = HOSTILITY

As I have noted, the use of labeling specifications illustrated in Figures 1 and 2 is not confined to a program listing. All of the Data-Text statistical routines use the labels to produce readable output. Since the sample programs were hypothetical, we cannot show output for them. However, Figure 3 illustrates partial output from an actual run using real data. The Data-Text program which defines the variables and requests the analyses is shown on the first page of output.* The remaining pages show the output resulting from the STATISTICS and ANOVA instructions.

Figure 3b

Figure 3c

Figure 3d

* The instructions which begin with double asterisks (**) are comment cards.

Figure 3—Sample output from a data-text run

The Data-Text System 339

The following two variables are pre-treatment ratings of overall illness. They are scored so that a 7 means most sick.

11 ILLNESS (2) = COL (1-2) = NURSES ILLNESS RATING (I-7=RANGE)
12 SYMPTOM (12) = COL (27-28) = DELUSIONS OF GRANDEUR
13 SYMPTOM (11) = COL (17-18) = SLOWED SPEECH
14 SYMPTOM (10) = COL (13-14) = AUDITORY HALLUCINATION
15 SYMPTOM (9) = COL (19-20) = INCOHERENT SPEECH
16 SYMPTOM (8) = COL (31-32) = NON-AUDITORY HALLUCIN.
17 SYMPTOM (7) = COL (33-34) = MEMORY DEFECT
18 SYMPTOM (6) = COL (25-26) = PRESSURE OF SPEECH
19 SYMPTOM (5) = COL (21-22) = PERSECUTION
20 SYMPTOM (4) = COL (15-16) = DELUSIONS OF GRANDEUR
21 SYMPTOM (3) = COL (9-10) = HOSTILITY
22 SYMPTOM (2) = COL (5-6) = GUILT
23 SYMPTOM (1) = COL (1-2) = HOSTILITY
24 SYMPTOM (0) = COL (0-0) = HOSTILITY

As I have noted, the use of labeling specifications illustrated in Figures 1 and 2 is not confined to a program listing. All of the Data-Text statistical routines use the labels to produce readable output. Since the sample programs were hypothetical, we cannot show output for them. However, Figure 3 illustrates partial output from an actual run using real data. The Data-Text program which defines the variables and requests the analyses is shown on the first page of output.* The remaining pages show the output resulting from the STATISTICS and ANOVA instructions.

Figure 3c

Figure 3d

* The instructions which begin with double asterisks (**) are comment cards.
LIMITATIONS OF THE DATA-TEXT SYSTEM

While a computer system may look good on paper, any real implementation always involves sacrifices and compromises. Some of the more obvious limitations and deficiencies involve the actual number of features which are available. Our investment of time and energy in making certain features as flexible as possible meant that other features could not be developed. In the transformational language, for example, there are no DO-loop and variable subscript capabilities. While the list expressions and summary functions available in Data-Text make these features less crucial than in other languages, on occasion a very complex transformation is desired for which these controls would be useful. Perhaps the best way to solve this would be to permit a subroutine call to a regular FORTRAN routine. The addition of such a feature would substantially enhance the transformational power of Data-Text (or any other social science language).

On the statistical side, we have confined the system to what might be called “standard” statistical routines that enjoy a wide usage in the social sciences. Some of the packaged program collections mentioned earlier (BMD, OSIRIS) offer many more statistical routines (discriminant function analysis, canonical correlation, Guttman scaling, etc.). This limitation is not a fatal one, however, since in Data-Text it is possible to write out a new transformed data set which can be fed into these other routines. But it would be more convenient to have all statistical analysis within the same integrated system.*

One of the major goals of the revised Data-Text system is machine independence. Therefore, almost all coding was done in a standard subset of FORTRAN IV. This fact coupled with the heavy overhead common to all large systems means that Data-Text is fairly slow and bulky. The load module requires about 200 tracks of an IBM 2314 disc, and although it has been extensively overlaid, the minimum region size is about 200K (250K is sufficient to run most jobs). Regarding speed, Data-Text is far more expensive to run than a FORTRAN program written to perform the same specific task. For example, the run illustrated in Figure 3 required 637 accesses and 20 seconds of CPU time on an IBM 360/65. In determining cost-effectiveness, of course, one must take into account the time to do the programming. An argument about the cost of Data-Text versus FORTRAN is not dissimilar to the same one about FORTRAN versus assembler language.

Another implication of machine independence is that we were confined to a basic subset of special symbols. This is one of the reasons (although not the only one) why our syntax may look strange to some language designers. The symbols = ) / ( are used heavily as separators and sometimes with different meaning. Slashes mean division in arithmetic statements, but they are also used to separate different category labels and recoding transformations.* Another reason for “strange” syntax has to do with maintaining upward-compatibility with the original Data-Text system. In the original system there was no *START instruction to separate instruction cards from data cards; instead, instructions were recognized by an * in column 1. Since we tried to keep the revised version compatible, we inherited this feature.

COMPARISON WITH OTHER SYSTEMS

I have noted that there are other systems developed for social science data analysis which are similar to Data-Text. Accordingly, there is a great deal of interest in the potential user community in the differences among the systems. While there is not sufficient room to present a detailed comparison here, some of the major differences can be summarized. I must state at the outset that I am not an unbiased observer in this regard, and knowledgeable users may well have different opinions.

Since Data-Text was developed primarily as a batch system, the biggest differences are with interactive systems. We can take IMPRESS2 as a popular representative. The main advantage of IMPRESS is that its command structure assumes that a student or researcher is in front of a console requesting one kind of analysis at a time. The result is that a great deal of communication can occur to help guide the analysis. This is extremely valuable for teaching purposes. For the researcher doing large-scale runs on his own data, however, most time-sharing systems (including IMPRESS) lack a variable definition and labeling capability which works on raw data. The raw data is put into a standardized, labeled form as a separate step (and without a language); in Data-Text this step is integrated with the transformational and statistical language. Also, most time-sharing systems are understandably less concerned about fixed parameter limits.

* The Cambridge Project under way at MIT and Harvard represents one very comprehensive attempt to do this on a time-sharing system.

* Commas could not have been used without making certain expressions look confusing; for example, consider (1 = 2/2, 3, 5 = 1) on a CODE statement.

From the collection of the Computer History Museum (www.computerhistory.org)
Space is generally at a premium, and it is not uncommon to find fairly low limits on the number of cells in a table, the number of variables in a regression, and so forth.

The package collections such as BMDP and OSIRIS are also quite different from Data-Text. For one thing, they offer many more statistical or analytic routines than Data-Text. Also, since they are really sets of individual programs, there is no system overhead and jobs can generally be executed in a smaller region and with much less CPU time than the same jobs in Data-Text. On the other hand, the lack of integration means that a lot of information has to be repeated for each separate analysis; that the control cards for different routines often have different syntax; and that, like time-sharing systems, variable definitions and transformations using raw data comprise a step separate from the analysis.

The system most similar to Data-Text is SPSS. Both are batch-oriented; both are integrated so that variable definition, transformation, and statistical analysis instructions are given in the same program; and both overlap considerably in their analytic routines.

Aside from syntax, the major differences in the transformational language have to do with speed versus power. Unlike Data-Text, SPSS does not have list expression or summary function capability, and it does not handle missing observations automatically in all variable transformations. This means that a SPSS program will generally require many more statements than Data-Text if there are conditions which require these features. On the other hand, for simple problems which do not need these features SPSS will take much less CPU time.

While analytic routines such as cross-tabulation, regression, and factor analysis are fairly similar in both systems, SPSS alone has Guttman scaling, partial correlations, and histograms. Data-Text alone has scatterplots, t-tests, and a generalized analysis of variance routine; the latter makes it possible to request just about any kind of analysis of variance design with a simple language tailored to this statistical procedure. Also, the Data-Text cross-tabulation routine will handle multivalued variables which can arise from multiple-choice response categories in a survey (it can also read in and transform multiple-punched cards). A more important difference for some researchers may be the parameter-free character of the Data-Text routines. The size of the matrix for the Data-Text correlation and factor analysis routines is not limited by available memory. This means that even in relatively small work space (e.g., 70K) a 500-by-500 (or one of any size, for that matter) correlation matrix or factor analysis can be computed.

THE IBM SYSTEM/360 IMPLEMENTATION

A brief amount of information about the implementation of Data-Text will supply perspective. Data-Text is a "compiler/interpreter" system programmed largely in ANSI FORTRAN IV on the IBM System/360 and 370 (a version is also underway for the CDC 6000 series).* During the compile phase the variable definitions and transformations are compiled into an intermediate language, and the statistical COMPUTE requests are encoded into sets of internal signals. During a second phase, the intermediate language is operated on by an interpreter for each unit of analysis to produce the final variables for the statistical routines. In addition, this set of variables is passed to each of the requested statistical routines for accumulation of the necessary quantities according to the encoded signals (e.g., sums, sums of squares, etc.). In the third and final phase each statistical routine computes and prints out their respective results using the accumulations from the second phase and the encoded signals from the compile phase.

The avoidance of fixed parameter limits in statistical routines is achieved in several ways. First, if there are more COMPUTE requests than there is room in available memory to accumulate the necessary quantities, then the data for the variable in question is automatically saved in a special file and subsequent passes are made over the file until all statistical requests are satisfied. Second, the programs for each statistical procedure are written using "dynamic" core allocation. That is, all accumulation arrays are linear, and subscripting is accomplished by use of linear subscript functions. Therefore, each statistical routine will use only as much space as is absolutely necessary for the analysis requested. Finally, certain statistical routines are programmed to use scratch data set space if necessary to process a request that will not fit in available memory. For example, the factor analysis routine will compute and save a correlation matrix in "pieces" if the whole matrix does not fit in available memory; the factoring is likewise computed in an iterative procedure which processes successive "pieces" of correlation matrix.

SUMMARY AND CONCLUSIONS

I began by stating that while there has been a lot of duplication of effort in the field of social science software, much of it was necessary for the normal develop—

* Major parts of the non-statistical systems design and programming has been carried out by Cambridge Computer Associates, Cambridge, Massachusetts.
ment of the field. Perhaps that claim can be better understood now. The standard languages distributed routinely by the hardware manufacturers did not meet many of the special statistical needs of social and behavioral scientists. The result was that almost every university computing center developed a package of canned programs to meet these needs. As the solutions became somewhat standardized, it became possible to develop more comprehensive languages like Data-Text and SPSS. By combining procedures for many different data processing tasks into one integrated language, these systems offer users considerably more power and convenience.

REFERENCES

1 H F Cline
Preliminary summary of a study of computer uses in social science research
Unpublished manuscript Russell Sage Foundation New York 1970
2 D J Armor A S Couch
The data-text primer
Free Press New York 1972
3 W J Dixon Ed
BMD—Biomedical computer programs
University Press Berkeley California 1967
4 Inter-university consortium for political research and institute for social research OSIRIS II
University of Michigan 1968
5 A S Couch
The data-text system
Unpublished users manual Harvard University 1965
6 N Nie et al
SPSS: Statistical package for the social sciences
7 J A David J H Sternick
The IMPRESS primer
Dartmouth College 1971
8 J R Miller II
Datanal: An interpretative language for on-line analysis of empirical data
9 S D McIntosh D M Griffel
The ADMINS primer
Center for International Studies Massachusetts Institute of Technology Cambridge Massachusetts 1968
10 R P Esada
TRACE—Model II user’s guide
System Development Corporation Santa Monica California 1967
11 J W Bean et al
The BEAST
The Brookings Institution Washington DC 1968
12 For information write Walt Maling at the NBER Computer Research Center for Economics and Management Science
545 Technology Square Cambridge Massachusetts 02139