PRINCIPLES FOR DEALING WITH LARGE PROGRAMS AND LARGE DATA FILES IN POLICY STUDIES

R. Yılmaz Argüden

February 1988

P-7409

PREFACE

This paper presents some principles for dealing with the practical concerns of data analysis in the context of performing a policy analysis study. These principles were followed in implementing the retirement policy analysis models that are part of the Air Force Enlisted Force Management System. It should be of interest to a wide audience of technically competent analysts who already know data analysis and how to write computer programs but who wish to improve their effectiveness by approaching the task of writing large programs and analyzing large data sets systematically.

ACKNOWLEDGMENTS

I have learned most of the principles for dealing with large data bases from Daniel Relles of The RAND Corporation. Without his instruction, this research could not have been completed in such a short time. I also wish to thank Warren E. Walker, project leader of the Enlisted Force Management Project, for his support.

CONTENTS

PREFACE iii

ACKNOWLEDGMENTS v

Section
- Introduction 1
- Know Your Data 2
- Know Your Problem and the Theoretical Models to be Used 3
- Make Time Investments at the Design Stage 6
- Conclusion 10

INTRODUCTION

The formal training of most analysts covers the theory of statistical and quantitative modeling in great detail. But, because of time constraints, even application-oriented courses provide experience in dealing with only small data sets. My experience at RAND indicates that many real-life problems require analyses of large data sets. Although the theoretical concerns are equally applicable to small and large data sets, some practical concerns (such as data cleaning, an analyst's understanding of the data sets, and writing computer code to transform variables) could be considerably more difficult for large data sets. Unfortunately, these subjects are left for the analysts to learn by experience. As eloquently suggested by the Chinese proverb, "What I've heard, I don't know; what I've seen, I vaguely remember; what I've done, I know," experience is the most effective teacher. However, learning by one's own mistakes is a costly and slow process.

I was fortunate enough to learn some basic principles to deal with the practical concerns of data analysis from others' experience--in particular, from Daniel Relles of RAND.[1] Following these principles improved the reliability, replicability, and efficiency of my analysis of retirement policies, and enabled me to complete this analysis in a short time frame.[2]

The purpose of this paper is to briefly explain these principles.[3,4] The paper is written for technically competent analysts who already

[1]See for example, R. Yilmaz Arguden, *Management of Large Data Sets: A Case Study with California Oil Wells,* The RAND Corporation, P-6802, October 1982.

[2]For the theory and results of the retirement policy analysis, see R. Yilmaz Arguden, *Personnel Management in the Military: Effects of Retirement Policies on the Retention of Personnel,* The RAND Corporation, R-3342-AF, January 1986.

[3]For more details on principles of data analysis, see Daniel Relles, *Allocating Research Resources: The Role of a Data Management Core Unit,* The RAND Corporation, N-2383-NICHD, January 1986.

[4]For an excellent book on principles of computer programming, see Dennie Van Tassel, *Programming Style, Design, Efficiency, Debugging, and Testing,* Prentice-Hall, Inc., Englewood Cliffs, 1978.

know data analysis and how to write computer programs but who wish to improve their effectiveness by approaching the task of analyzing large data sets systematically. Given the benefits of these principles in dealing with real-life problems, I advocate making the study of these principles a requirement in any graduate program for statisticians, economists, operations researchers, and other quantitative analysts.

KNOW YOUR DATA

Although the small data sets may be prepared by the analyst himself, large data sets generally are not prepared by the analyst but they are obtained from other sources. Therefore, understanding the contents of the data sets to be used in analyses should be the first step. One could attempt to do this by reading the documentation for the data sets, but my experience suggests that too frequently errors creep into documentation and the documentation may not be current. Hence, reading the documentation cannot replace looking at the data. But visual inspection of large data sets is practically impossible. The following steps can be used to look at the data.

Look at the Frequencies of All the Variables

Frequencies indicate the number of times various values of a variable appear in the data. When the variable takes on too many different values (e.g., income), all the digits beyond a desired number of significant digits can be recoded into a standard value before doing the frequencies (e.g., if the desired number of digits is 2 then the number 11,351 could be recoded as 11,xxx). At this stage, keeping the data in character format (rather than converting to integer or real format) would avoid the unnecessary step of data conversion and would prevent inadvertently changing the value of the data during the conversion step.

Look at Partial Listings of Data

Partial listings improve the analyst's understanding of the data set by allowing him to investigate relationships among variables within a given observation. Proper formatting of partial listings (e.g., listing chosen variables next to each other rather than in the order they appear in the data set, using blank spaces between observations to improve readability, and listing a variable more than once each time a joint relationship with another variable is being searched) can facilitate an analyst's understanding of the data and the detection of errors. Because it is practically impossible to study and comprehend the full data set when it is large, partial listings also improve the understanding of units of measurement, completeness of the information about each variable, and whether the definitions provided in the documentation are accurate.

KNOW YOUR PROBLEM AND THE THEORETICAL MODELS TO BE USED

The next steps involve using the variables in the source database to define others that are more useful for analysis (i.e., recoding of variables), defining the units of observation (if necessary, aggregating or disaggregating raw data to relevant units of observation), merging in other data sets to complete the information needed for analysis, and selecting a relevant set of observations to analyze the policy question at hand.

All of these steps are too important to be left solely to programmers who may not be knowledgeable about the subject matter or the theoretical models to be used. Also, as the analyst's understanding of the problem is increased, these steps may have to be repeated. Unless the analyst himself/herself is fully immersed into the details of recoding, reorganization, merging, and observation selection, many errors can remain unnoticed and the policy recommendations can be misleading.

Recoding of Variables

Recoding raw variables into variables that will be used in analysis and about which analytical assumptions will be made requires good knowledge of the modeling approach. For example, not incorporating observations with some missing variables into the data set could cause serious selection biases. To the extent possible, rules for different variables should be kept separate (e.g., different subroutines), user-defined variables should be defined directly from raw data rather than from other user-defined variables, and the data recoding algorithms should be independent from the structure of the data sets.

Following the above mentioned principles and the programming principles explained below improve comprehensibility of programming code and reliability of the code, particularly during the numerous reruns that would be required as the analyst's understanding improves.

Reorganization, Sorting, and Merging

Reorganization of data is more complicated if the units of observation are being redefined. This may involve aggregating, disaggregating, or changing the structure of the original data set to facilitate analysis. In some major research projects recoding and reorganization steps took the majority of the time, much more than the analysis step. This was time in the Enlisted Force Management Project. However, without this major effort by experienced researchers, many of the analyses would not have been possible.

The addition of new data to a data set is best accomplished by preparing another data set with an index variable that provides the crosswalk between the two data sets. Then the two data sets can be merged by the values of this index variable, with or without sorting. This is preferable to assigning the additional data within the programming code, because it keeps updates of data and code independent, thereby reducing errors in the long run when either the code or the data need to be changed.

In using large data sets, analysts should be careful not to be overconfident about the large numbers of perceived degrees of freedom (i.e., number of observations), because some variables, particularly those that are merged from other data sets, may have very limited actual degrees of freedom (e.g., average annual inflation rate in a data set having 500,000 airmen covering a ten-year time span has only nine degrees of freedom).

Selection of Observations

All selection rules should be compartmentalized. Avoiding introduction of selection biases in choosing the observations to analyze is one of the most difficult steps of analysis. Often errors are introduced to the analysis at this stage. Different analyses may require a different subset of the original data set. By having all the selection rules in one part of the code, the analyst can avoid inadvertently leaving some observations out of the sample.

The analyst may wish to modify some of the selection rules once preliminary analyses are conducted. This will be easier if the original selection rules were compartmentalized. Also, when multiple researchers use the same data set, compertmentalization allows easy comparison (or even standardization) of selection rules.

Looking at Results

In regression analyses, analysis of residuals (checking for patterns, looking at normal plots in an ordinary least squares analysis) and checking for outliers may indicate errors in the original preparation stages. Often visual aids help identify unsatisfied assumptions. Yet, with large data sets visual aids such as residual plots may simply have too much data. In that case, rather than doing without these aids, it is preferable to create summary data sets (aggregating predictions, residuals, residual errors, prediction errors by values of different variables), which can then be used to create meaningful residual plots.

Finally, unlike the situation with small data sets, when a large data set is used in analyses, providing data listings in appendices is infeasible. However, providing tables that indicate summary statistics of the variables gives the readers of a report an idea about the data set. Such tables can also provide the ranges of independent variables to indicate the values of variables that can be used in making out-of-sample predictions with confidence.

MAKE TIME INVESTMENTS AT THE DESIGN STAGE

Whether one is recoding or reorganizing data or writing programs for analysis, the programming principles to be explained below are equally valid. Almost all of them involve time investments when writing the program the first time. Almost any algorithm one writes is likely to be utilized more than the writer would think at the time he is developing it. This is not because all algorithms are ingenious, but because they are likely to be updated to reflect new knowledge the analyst acquires during the study of the subject. Ironically, it is usually the updates that cause errors to creep into the code, particularly when the original code does not follow basic programming principles. Also, once an algorithm for a specific purpose is available, the writer would be more likely to adapt that code for a new purpose than to write a new algorithm. Therefore, taking time to do it right at the beginning saves a lot in the long run, both in terms of reliability and time.

Keep It Simple

This *simplicity in writing programs* principle is the direct application of an important engineering principle, "KISS" (Keep It Simple, Stupid). Any program that does not make apparent sense is likely to require a lot of time to understand (after a while even for the original programmer) at the debugging stage and each time an update is required. This is costly and invites error. The following principles would produce simpler programs.

Selection of variable names is very important for readability of programs. Therefore, use descriptive variable names and avoid variable names that look alike, sound alike, or are spelled alike.

For example,

```
SAVING = INCOME - EXPENSE
```

is better than

```
A = B - C
```

which is better than

```
A1IX1 = AI1X1 - AI1XI.
```

Use of blank lines, indentation, and columnwise spacing of variable lists to indicate program structure improves readability and maintainability of programs (e.g., indenting the lines of code relating to a DO loop). Program maintainability can also be improved by providing more comments than seem necessary at the design stage, because the users (after a long time period even the original programmer) are unlikely to have the same mind frame as the programmer. Also, starting to prepare documentation at the design stage is very important to fully convey the road map of the program. The comments and the documentation should provide a description of the program, inputs, variables, program logic, outputs, and potential modifications that future users may contemplate.

Accomplishing different tasks in different modules (subroutines and functions) keeps each module simple enough to be understood without much effort. Each module should be independent from the larger program in which it will be utilized. This can be accomplished by having all the inputs and outputs passed to the module as arguments of the module. (The use of COMMON statements should be limited to parameters that define the environment and do not get updated within the larger program.) Once such a module is tested to be correct, it can be used as a part of a larger program without having to review its logic. This reduces the complexity of large programs.

Maintainability of a program and programmer/user efficiency are more important than computational efficiency of computer programs, particularly with the recent developments in hardware technologies. Therefore, programming tricks to improve efficiency should be avoided when they are likely to diminish simplicity or readability of the program (e.g., mapping a variable with dimension "n" into another with dimension "m," where m < n, in modules that do not use all "n" values of the variable). Also, spending too much time to improve the computational efficiency of statements that will be executed only a few times in the program is not worth the programmer's efforts.

Minimizing or eliminating the use of GO TO statements, and using them only for skipping a number of lines of code rather than using them to go back in the program improves readability of programs. This is because it is easier to follow a top-down logic than a logic with many branches and loops. Certain programming languages allow use of structured programming more easily than others. Therefore, choice of the programming language should take into consideration the complexity of the task. Complex data recoding can be handled better in languages like FORTRAN and PL1, whereas once the data are in two-way matrices it is easier to use statistical packages like SAS and SPSS to do the analysis and obtain summary statistics and graphs.

Keep Your Programs as General as Possible

When a program is independent from the context in which it is run, errors are less likely to creep in during modifications, and it is easier to adapt it to other purposes in the future. Therefore, programming generality into the code increases the reliability and utility of the program.

One of the important factors in making a program general is to use variables instead of constants as parameters. If a constant is used, each time it needs to be changed all the program lines using that constant need to be changed. This is inefficient and invites error. If certain constants are nevertheless used, those lines should be marked with a special character (e.g., "@") to facilitate locating the lines that need to be changed if and when the constant needs to be changed. This is much easier to do during original programming than it is later, when a change is required.

Making modules independent from each other improves the generality of the program. Each module should also be independent from source of input, output, and past use of the module. This can be accomplished by having all the inputs and outputs passed to the module as arguments of the module, minimizing use of COMMON statements, and initializing local variables (i.e., variables that are used only in a specific module) in the subroutine in which they are used.

Finally, testing each module for a wide range of values for input variables, including normal, extreme, and exceptional cases, increases its reliability. Developing test data, introducing debugging aids into the modules at the design stage, and thoroughly testing each module prior to using it in a major program reduces errors and total debugging time. Investing time at the design stage is much more efficient than trying to put out brush fires after starting the analysis stage.

Leave All Routine Tasks to the Computer

Humans are more error prone than computers. Therefore, leaving routine tasks to the computer by minimizing manual changes to the program for each run increases reliability and allows the analyst to focus on analytical issues. However, this too requires up-front time investments.

Making modifications to the program at each run by electronic means minimizes the risk of introducing errors through manual changes. Electronic modification can be accomplished by preparing a command file (possibly using a text editor on a PC) that automatically modifies all relevant lines of the program and feeding this command file to the system (through a communications package). Also, saving all programs relevant to a particular task in one file makes automatic modification easier and ensures consistency of modifications across programs.

Often errors are introduced in modifying output locations, and in allocating space and time for each run. However, having these as part of the command file could avoid many errors. Particularly if the command file itself follows the programming principles already explained, these errors can be avoided (e.g., leaving constants like output tape numbers as variable names rather than leaving the last tape numbers used would avoid writing over valuable data). Messages to the computer operator to load and unload tapes can be changed automatically with the changes in JCL (job control language), thus ensuring consistency.

Command files can also be used to keep a log of the input values and output locations used for each run by having such information saved automatically rather than leaving this to be done manually. Assigning descriptive labels to output tables and graphs is very important to avoid confusion. When the same program is run at different times with different assumptions, the output labels, which are usually assigned manually, may not be unique. Using command files and including the date and time in the label prevents this problem. The best place to store a general version of such a command file to be updated for each run is within the program itself in the form of comment lines. Finally, updated command files can be saved for future reference, avoiding any uncertainity about the manual modifications made for a given run.

CONCLUSION

This paper has briefly described some practical principles to improve the effectiveness of quantitative analyses using large data sets. Most of these principles are fairly simple and can easily be learned through experience. However, many analysts whose formal training has covered only theoretical aspects of quantitative analyses may not have had the time to figure out and follow these simple principles. Not investing the time required to follow these principles could cost dearly in the long run in terms of unreliability, irreplicability, and inefficiency of the analysis, which in turn may lead to misleading policy recommendations.

P-7409 PRINCIPLES FOR DEALING WITH LARGE PROGRAMS AND LARGE DATA FILES IN POLICY STUDIES R. Yilmaz Arguden