A SYSTEM FOR WRITING INTERACTIVE ENGINEERING PROGRAMS IN APL

by

BRIAN R. SAMSON

BASc., University of British Columbia, 1981

A THESIS IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE in THE FACULTY OF GRADUATE STUDIES

The Department of Civil Engineering

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

OCTOBER 1985

© Brian R. Samson, 1985
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Civil Engineering

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date 15 Oct 1985
ABSTRACT

As the use of computers in engineering becomes more significant and widespread, there is a growing need for interactive computer programs which can be used with a minimum of user preparation. This thesis presents and demonstrates a system for writing interactive engineering programs in APL, a programming language. A good interactive program is sensitive to the needs of the user, and generally includes help features, default options, escape features and check features. To include all of these features in a conventionally organized program is complicated and tedious, especially for longer programs with many interaction events between the program and the user. The system presented here makes it fairly simple to include all of the above features, and provides two additional benefits:

1. The logic of the program becomes more prominent, hence easier to follow and check.

2. The program tends to be highly modular in form, making it more readable and easier to test and debug.
## TABLE OF CONTENTS

**CHAPTER** | **PAGE**
--- | ---
1. INTRODUCTION | 1
2. BACKGROUND | 4
   2.1 General Requirements | 4
   2.2 Literature Review | 5
   2.3 The UBC Commerce System | 7
   2.4 The Use of APL | 9
3. THE C.E. SYSTEM | 10
   3.1 The Interaction Event | 10
   3.2 The Phase Tables | 14
   3.3 Writing, Entering and Editing the Phase Tables | 16
   3.4 The Execute Function | 17
   3.5 The Correlate Function | 18
4. A GUIDE TO USING THE C.E. SYSTEM | 19
   4.1 Basic Ideas | 19
   4.2 Program Structure | 19
      4.2.1 Prompt Table | 21
      4.2.2 Default Table | 21
      4.2.3 Help Table | 22
      4.2.4 Check Table | 22
      4.2.5 Action Table | 23
      4.2.6 Flow Control Table | 23
   4.3 Further Suggestions | 24
   4.4 Simple Example | 25
   4.5 Entering and Running the Program | 25
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 A SIMPLE EXAMPLE</td>
<td>26</td>
</tr>
<tr>
<td>5.1 EXAMPLE: PROMPT TABLE</td>
<td>35</td>
</tr>
<tr>
<td>5.2 EXAMPLE: DEFAULT TABLE</td>
<td>36</td>
</tr>
<tr>
<td>5.3 EXAMPLE: HELP TABLE</td>
<td>37</td>
</tr>
<tr>
<td>5.4 EXAMPLE: CHECK TABLE</td>
<td>38</td>
</tr>
<tr>
<td>5.5 EXAMPLE: ACTION TABLE</td>
<td>39</td>
</tr>
<tr>
<td>5.6 EXAMPLE: FLOW CONTROL TABLE</td>
<td>40</td>
</tr>
<tr>
<td>5.7 A SAMPLE RUN</td>
<td>43</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 THE PHASES OF AN INTERACTION EVENT</td>
<td>13</td>
</tr>
<tr>
<td>4.1 THE PHASES OF AN INTERACTION EVENT (REPEATED)</td>
<td>20</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

The author wishes to thank Dr. S. O. Russell for his guidance in the preparation of this thesis, and the Natural Sciences and Engineering Research Council for financial support.
CHAPTER 1

INTRODUCTION

In the field of engineering there are countless examples of problems which could quickly and easily be solved by computer program, if only the required program existed in a form that was readily usable. Such examples include: pipe flow calculations, uncertainty estimation, culvert sizing, backwater calculations and runoff estimates using unit hydrograph methods. These are just a few examples from the area of water resources, and there are many others from each of the branches of engineering. When an engineer is faced with solving such a problem, he assembles the necessary information (equations, techniques, data, etc.) and then either computes a solution by hand, or writes a computer program to do it, whichever is less time-consuming or costly. If the engineer decides to write a computer program, usually the end result is a program which is specific to the particular problem for which it was written. In addition, the engineer who wrote the program is often the only person who is familiar enough with it to run it. This lack of versatility arises because the engineer generally wants answers as quickly as possible, and cannot afford the programming time required to make the program versatile and user-friendly. After serving its initial purpose, the program, often lacking adequate documentation, is usually stored somewhere. If, some time later, the need arises for a similar program, the old program may be resurrected and, if necessary, altered to fit the new problem. This can be very time-consuming, especially if the person who wrote the original program is not available, or does not have a good memory.
Ideally, a solution to this dilemma would be to have a library of computer programs which can solve many of the problems that a group of users, say a consulting firm, often encounter. These programs would have to be interactive and user-friendly, and preferably would require the user to read little or no documentation. When an engineer is faced with a certain task, he looks up the appropriate program in the library, and solves the problem interactively with the computer. This sounds fine, but there still remains the difficulty of producing versatile, interactive programs.

Generally, the part of the program which does the computations requires less time and effort to write than the part which controls the dialogue and interaction. A good interactive program will contain most or all of the following features:

1. The program will provide explanations at the request of the user.

2. The program will be capable of assuming default values if no response is provided by the user.

3. The user can leave the program at any time.

4. The program will check user input for validity or other conditions, and if necessary, send an appropriate message to the user.

5. The program will allow for easy correction of entry errors.
6. The program will be responsive to the user's needs and will not easily crash as a result of user entry error.

It is quite time-consuming and complicated to include all of these features in a conventionally-organized program, especially for programs containing a lot of interaction with the user. This is not good for engineers, who often are not skilled programmers. This thesis presents a programming system, hereafter referred to as the C. E. (Civil Engineering) System, which simplifies and speeds up the process of writing the interactive part of the program. The C. E. System is easy to learn and use, and should be useful to engineers who want their programs to be more interactive and easier for others to use.

Chapter 2 provides background information and reviews two previously developed systems for writing interactive computer programs. The C. E. System is explained in Chapter 3, while specific instructions for use are provided in Chapter 4, a self-contained guide to using the C. E. System. Chapter 5 presents a comprehensive example consisting of a program to estimate the uncertainty of any function of random variables.
CHAPTER 2
BACKGROUND

2.1 General Requirements

Computer users today are often not programmers, and may not be interested in becoming involved in computers beyond knowing what is necessary to use certain programs. There is a need for a high quality human/computer interface which will not discourage users not experienced in computers or computer languages. The user of an interactive program preferably should not be faced with error messages from the computer operating system or language software system. The manufacturers of micro-computers, in an attempt to reach the widest possible market, now offer operating systems which are amazingly simple and "friendly". In one case, the human/computer interface has been reduced to the most basic communication level, the user simply points to what he wants. However, computer software designers face an increasingly complex task in their efforts to make things easier for the end-users of the software.

Most of the effort and skill required to write a good interactive program generally goes into writing the interactive or dialogue part, rather than the part which does the calculations or data processing. A tool which would facilitate the writing of the interactive part would decrease the time and cost required to develop interactive programs, and would also enable less highly skilled programmers to construct effective interactive dialogue.

There are many difficulties to be overcome when designing interactive computer programs. One problem is that a computer-guided program intended
for very inexperienced users would be too slow and frustrating for experienced
users. On the other hand, inexperienced users might become uncertain or
confused if faced with a very streamlined, user-guided program. One solution
to this problem is to structure the program so that it is fairly easy to adapt the
dialogue part of the program to the various experience levels of the users.
Also, programs can be designed which allow the user to speed up the dialogue by
opting for more streamlined routes through the dialogue, containing fewer
prompts and error correction facilities. The system presented in this thesis
simplifies the task of writing versatile human/computer dialogue, and also
allows for easy editing of the dialogue. Another advantage of the C. E. Sytem
is that it allows the programmer to first write a simplified working version, to
which he can later easily add help messages, default options, input checking
routines, and more complete prompts.

2.2 Literature Review

There have been a number of systems developed to simplify the task of writing
interactive programs. Two recently developed systems will be outlined in this
section. The first such tool is the MACAID system, which has been developed
in the last few years at MacMaster University (Ahmed, 1980). MACAID
utilizes a "master driver" program, and tables of text which contain the
dialogue. MACAID is intended to be used to construct interactive
instructional-type programs. The feasibility of using the MACAID system for
writing interactive engineering programs was investigated, but the following
obstacles were encountered:
1. Due to differences in computer facilities, it took several days of debugging to just get MACAID running.

2. MACAID is a fairly large program designed for some objectives not relevant to the goals of this thesis. (e.g. instructional programs were not the specific aim). It became evident that the effort required to adapt MACAID was not worthwhile.

3. Learning to write programs using the MACAID system seemed almost as difficult as learning a new computer language. A simpler system was required.

Eventually it was decided that MACAID did not meet the necessary requirements.

Another tool for writing interactive programs has been developed by the Faculty of Commerce and Business Administration at U.B.C. Their system employs a master table which contains both the program logic and references to other tables where text and executable lines are stored. (Benbasat, 1981, p. 11).

This system seemed to hold promise, and has been used as a strong basis for the C. E. System, which is designed to be used for writing interactive engineering programs. The main objective of the U.B.C. Commerce System, and the C. E. System as well, is to improve the design of the human/computer interface "from being an ad hoc process to being structured and planned". (Benbasat, 1981, p. 4).
The fundamental idea behind the Commerce system is that a human/computer dialogue can be seen as a series of "interaction events". An interaction event generally consists of a prompt by the computer, a response from the user, and any computer action or processing which occurs prior to the next prompt. An organised collection of these interaction events makes up an interactive program, which also must contain a system of flow control which defines all possible event sequences through the dialogue which a user may choose.

The interaction event itself can be broken down into "phases". Generally there are up to seven phases in each interaction event, although not all of them will necessarily be used in every event. These seven phases are:

1. **Prompt**: The computer asks the user to supply data or instructions.
2. **Input**: The user responds, or uses a default option.
3. **Escape**: Allows the user to leave the program if he wishes.
4. **Help**: If requested by the user, an explanation is provided of what the computer is prompting for.
5. **Check**: The user input is checked for errors, validity, or any other condition.
6. **Action**: Any related processing is carried out by the computer.
7. **Flow Control**: The program determines which event to proceed to next.

One interaction event consists of moving once through the phases in the order they are listed above. To determine the order in which the interaction events are executed, there are three basic types of flow control:
1. The next event is pre-set, regardless of user input or the results of any processing.

2. The next event is selected, depending on user input or the results of some processing.

3. The next event is pre-set unless a certain condition is met, in which case a branch to a different event occurs.

At first, these concepts seem to merely add complexity to the problem of writing an interactive program. However, these ideas can be used to impose a structure on the program which makes it more organized, and easier to write and modify. Both the Commerce System and the C. E. System are based on the concepts described so far, but the two systems differ in the way they implement the ideas. This section will outline the Commerce implementation, while the description of the C. E. System will be covered in the next chapter.

In the Commerce system, the program logic is contained in an "event description table" made up of words which are references to other tables which contain the actual program information. The order of the words in the event description table determines the order in which the program information is executed. Each entry in the event description table refers to certain lines in a particular information table. There are several information tables, with each one containing information of the same type (eg. text) or of the same phase (eg. check). In theory, this system is good, and in practice it works, but when the author attempted to write a program using it, it was found to be somewhat
complicated. The C. E. implementation is simpler in that it does not use an event description table, and has one table for each of six phases (prompt, default, help, check, action, flow control).

2.4 **The Use of APL**

APL, first defined in 1962 by K. E. Iverson, is a general purpose language which is particularly suited to matrix manipulation. Learning APL consists mainly of learning the uses and rules of its extensive built-in operators. These operators, depicted by symbols, result in a language which is terse and powerful, although the programs can be difficult to read. The most unique characteristic of APL is its use of the "workspace" concept. When a user interacts with the computer via the APL interpreter, he does so in a particular workspace. In this workspace the user stores functions (programs) and variable names with their current values. Functions are executed simply by naming them, and lines entered by the user are executed immediately. This conversational environment where all functions and variables in the workspace are easily and automatically integrated, in addition to the powerful array manipulation operators in APL, make the language very suitable for implementing the programming tool described in this thesis.
CHAPTER 3
THE C. E. SYSTEM

3.1 The Interaction Event

The Civil Engineering system of designing interactive programs is based on the concept of a sequence of interaction events, each consisting of ordered phases, but the implementation is somewhat different from that of the Commerce System. The program is written one interaction event at a time, and the flow control determines the order in which the events are executed. Each interaction event consists of six phases:

1. **Prompt Phase**

   The computer displays a prompt, generally asking the user for input (data, commands, etc.) and the user enters his response. If the user enters:

   i) "back", then the next phase executed will be the prompt phase of the previous event. This option is provided so that the user can make changes to input he has already entered in previous events.

   ii) "escape", then the program stops execution. This is provided so the user can easily leave the program at any time.

   iii) nothing (eg. just presses return), then the program proceeds to the default phase.

   iv) "help", then the program proceeds to the help phase.

   v) characters other than "back", "escape" or "help", then the program proceeds to the check phase.
2. **Default Phase**

If the user enters nothing in response to a prompt (just presses return), then this phase is executed. The programmer uses this phase if he wants to give the user the option of just pressing return in response to a particular prompt. The programmer may define certain lines which will be executed if the user enters nothing. In many events there may be no default option, in which case if the user just presses return he will be advised that there is no default, and the prompt will be re-issued. After the default phase is executed, the program proceeds to the action phase.

3. **Help Phase**

If the user enters "help" in response to a prompt, then this phase is executed. The programmer uses this phase to provide the user with an explanation of what is required in response to the prompt. After this phase is executed, the program returns to the prompt phase.

4. **Check Phase**

If the user enters characters other than "back", "escape" or "help" in response to a prompt, then this phase is executed. The programmer uses this phase to check whether the user input meets certain conditions, which the programmer specifies. If the user input is not satisfactory, then the program returns to the prompt phase of the same event. If the user input is satisfactory, then the program proceeds to the action phase.
5. **Action Phase**

In this phase, any required processing or calculations are carried out. Subroutines (APL functions) are generally used to perform the calculations.

6. **Flow Control Phase**

After the action phase is complete, the number of the next event is determined in the flow control phase. The next event may be pre-set, or may depend on the user input or the results of the action phase. After the flow control phase, the next event starts, beginning with the prompt phase.

The flow control within each interaction event is the same regardless of the particular event or program where it occurs. Figure 3.1 on the following page summarizes the six phases which make up every interaction event.

Figure 3.1 applies to every interaction event, and so a universal program, called a driver, has been written to control the flow within each interaction event. It remains for the programmer to write the actual lines contained in each phase of the events, as well as the flow control which governs the order in which the events are executed. The executable lines are organized in tables (character arrays), one for each of the six phases, and each phase table contains lines corresponding to that particular phase of every interaction event in the program. A number at the beginning of each line in the phase table indicates the event to which that line belongs. Thus the entire dialogue of a particular program is contained in six character arrays.
FIGURE 3.1
THE PHASES OF AN INTERACTION EVENT

The arrows represent possible paths between phases. A statement on an arrow indicates the condition under which that path is followed. An arrow without a statement indicates a path always taken between two phases.
3.2 The Phase Tables

When using the C. E. System, a programmer writes each interaction event as a unit. However, the program lines are stored in tables according to the phase of the interaction event to which the lines belong. There is one table for each of the six phases (prompt, default, help, check, action, flow control). The interaction events which make up the program are numbered sequentially, and each line in each phase table is numbered to indicate the event to which the line belongs. In each phase table, up to 10 lines can have the same event number. Thus for each interaction event the programmer writes the prompt lines in the prompt table, the default lines in the default table, etc. Generally, the programmer would write the program out by hand first, and then store each complete phase table in the APL workspace. The phase tables are stored as character arrays with the abbreviated names: prom, defa, help, chec, acti and flow. Specific information regarding each phase table follows. Refer to the simple example on page 26.

The Prompt Table

The prompt table is used to prompt for and receive information from the user. All input from the user is assigned to the character variable called 'answer'. The programmer may also assign the input to any other variable name, as in the example. The first three spaces of each line in every phase table are reserved for the interaction event number. The last line in each phase table must have a zero in the event number columns to indicate the end of the table. (This last line is not executed).
The Default Table

This table contains lines which are automatically executed if the user defaults in response to a prompt. Usually the lines in this table are used to assign values to variables. For any event where the programmer does not wish to provide the user with a default option, he uses the line "default = 0", as in the example.

The Help Table

The lines in this table are executed if the user enters "help" in response to a prompt. The table may contain explanations or examples, whatever the programmer wishes to have displayed. The programmer can also include a system of keeping track of how many times the user enters "help" while running the program. For any of the phase tables, the maximum number of lines which can be used for a particular event is ten. If more lines are required, the programmer can use a separate function.

The Check Table

In this table, the programmer specifies any checks to be made on the user input. Lines in this table are always written in pairs. The first line of the pair is a check line which, when executed, results in a value of either one (if true) or zero (if false). If the value of the check line is one, then the second line of the pair is executed. If the value of the check line is zero, then the second line of the pair is not executed. Certain user input may be unacceptable and in this case the programmer may want to return the user to the prompt phase of the event. To accomplish this, the line "check = 0" is used as the second line of the pair. This is a signal to the driver program that the user is to be re-prompted for the input. (See example).
The Action Table

This table carries out any calculations or other processing which the programmer wants executed before the next interaction event begins. To keep the table reasonably brief and readable, the programmer should use functions for those events where more than one or two lines of programming are required.

The Flow Control Table

In this table, the number of the event to be executed next is assigned to the variable names "next". The number of the next event may be pre-set or it may depend on user input or the results of calculations. The programmer may also want to repeat an event until a certain condition is met. All of these situations can be handled with an assignment line which uses the APL compression symbol, as in the example.

3.3 Writing, Entering, and Editing the Phase Tables

Each phase table is a two-dimensional character array with a four character name (prom, defa, help, chec, acti, or flow). A function has been written to assist the programmer in entering each table into the workspace. A convenient way to write a program using the C. E. System is to first write only the Prompt, Action, and Flow Control tables. These three tables are enough to create a basic working version of the program which is easier to handle and debug. Once this brief version is running, the Default, Help and Check tables can be added to make the program more interactive, user-friendly, and foolproof. A function has been written to assist the programmer in editing the phase tables. More specific instructions for writing a program are given in Chapter 4 entitled "A
Guide to Using the C. E. System", which is meant to be a self-contained guide. Those interested in using the C. E. System should be able to use the guide without having to read this entire thesis.

3.4 The Execute Function

Once the phase tables have been written and entered into the workspace, the program is run by executing the lines in their proper order. As stated previously, the function which accomplishes this is called a driver. For phase tables written using the C. E. System a driver function ("execute") has been written which is based on the logic illustrated in Figure 3.1. Event number one is executed first, starting with the prompt phase and then moving to other phases depending on user input. In each phase, all lines with the current event number are executed in sequence. For some events, the programmer may want to use the prompt phase simply to display a message not requiring user input. The driver function senses those events where no user response is required, and moves on to the action phase of the event.

The driver function continues to execute the lines in the phase tables until the final flow control phase line, which stops execution.

When the user enters "back" in response to any prompt, the driver function automatically moves back to the prompt phase of the event preceding the current one. This is not necessarily the event that was executed immediately previous to the current one, but is the event which has the next lower number. It would be possible to design the driver function so that it would trace backward through the events in the reverse order to the way they were actually
executed. However, this might lead to confusion in those instances where the user had already used the "back" option previously in the same run. For this reason, when the user enters "back", the driver function simply moves backward one event in the phase tables (e.g. to the next lowest event number). The user can back up all the way to the first event if he wishes.

3.5 **The Correlate Function**

So that the execute function does not have to repeatedly search each phase table to determine which rows belong to the current event, the correlate function has been written. After the phase tables have been written and entered into the workspace, the programmer must run the correlate function before his program can be executed by the driver. The correlate function runs through the phase tables once, and records which row numbers in each phase table correspond to the different event numbers. The results of the correlate function are stored in a three-dimensional array which is used by the driver to determine which rows of each phase table to execute as part of each event. This saves execution time when the driver program is run.
1. An interactive program can be written as a series of interaction events between the computer and the user. A basic interaction event consists of a prompt from the computer, a response from the user, and any processing done by the computer before the next prompt is issued.

2. The interaction event can be split into six phases: prompt; default; help; check; action; flow control. Figure 4.1 on the following page represents an interaction event, showing the function of each of the phases. The arrows show how the program user moves through the phases. Each interaction event starts with the prompt phase, and the user response determines the next phase.

4.2 Program Structure

Once written, the program consists of six phase tables which define the dialogue, and any number of subroutines or functions which perform calculations or processing. The phase tables are in the form of character arrays with the first three columns of each row used for the event number, and the remaining columns used for any executable line in APL.

To write the program, start with event number one. Write the prompt (the message you want the user to see) in the prompt table, then default in the
FIGURE 4.1
THE PHASES OF AN INTERACTION EVENT

The arrows represent possible paths between phases. A statement on an arrow indicates the condition under which that path is followed. An arrow without a statement indicates a path always taken between two phases.
default table, then the help message in the help table, etc. Specific comments for writing each table follow below.

4.2.1 Prompt Table

(Named "prom"; for programming convenience, each phase table name has been abbreviated to four characters).

The prompt phase of each event generally consists of the prompt itself, followed by a line which accepts input. eg.

1 'Enter your high estimate of the variable x'.

1 answer = xhigh = $/

The number occupying the first three spaces of each line is the event number. All lines with the same event number will be executed together in sequence. All input is initially in character form ($/) and always is assigned to the variable called "answer". If the programmer wishes to also assign the input to his own variable name (xhigh in the above example) he may do so.

4.2.2 Default Table (named "defa")

In some interaction events, the programmer may want to give the user the option of just pressing the return key, in which case the program will automatically assume a value for the input requested. This is accomplished by placing a line such as the following in the default table.

eg.

1 xhigh = 1000

If the user just hits return in response to the prompt of event number one, then the lines with event number one in the default table will be executed. In the example, the variable xhigh will be assigned the value 1000. If the programmer
does not want to give the user a default option for a particular event, he uses
the line:

```
default = 0
```

In this case if the user just hits return he will be sent a message that there is no
default available, and he will be re-prompted for the required input.

### 4.2.3 Help Table (named "help")

If the user wants an explanation of any prompt he is given, he may enter "help"
in which case the help table lines for that particular event are executed.

**eg.**

```
 1 'Your high estimate of x is the value which'
 1 'you are 90% sure will not be exceeded.'
```

If the user enters "help" in response to the prompt for event number one, the
above message is displayed and then the prompt is repeated.

### 4.2.4 Check Table (named "chec")

The programmer may want to check that the input just entered by the user
satisfies certain conditions before proceeding further. The check table is
different from the other phase tables in that lines are always written in pairs.
The first line of a pair is always a condition. If the condition is true (eg. equal
to one when evaluated), the second line of the pair is executed. If the condition
is false (eg. equal to zero when evaluated), then the second line is not executed.

**eg.**

```
 1 ($EX xhigh) $ LT 0
```

```
 1 'Your high estimate of x must be greater than zero'.
```

For any particular prompt, certain user response may be unacceptable, and the
programmer may want to display an appropriate message, then repeat the
prompt. The following pair of lines written in the check table would cause the prompt to be repeated if xhigh is negative.

```
1 ($EX \text{xhigh}) \text{ LT} 0
1 \text{ check} = 0
```

The second line is interpreted by the execution program to mean "go back to the prompt phase of this interaction event".

4.2.5 **Action Table** (named "acti")

The action table contains any lines which the programmer wants executed before proceeding to subsequent events. Generally this will consist of any calculations to be carried out using information just entered by the user. If more than one or two lines are required, the programmer should use a subroutine since this keeps the phase tables brief and easier to read.

eg. 1 \text{ v} = \text{xlow variance xhigh}

In this line, "variance" is a program which calculates the approximate variance of \( x \) using the high and low estimates xhigh and xlow.

4.2.6 **Flow Control Table** (named "flow")

This table is used to determine which event to execute next. The variable "next" is assigned a value which is the number of the event which is to be executed next. For some events the next event may be predetermined.

eg. 1 \text{ next} = 2

For other events, the number of the next event may be determined by the user input or the outcome of the action phase. In these cases, the compression operator (\%) can be used to assign the appropriate value to "next".
eg. 1 next = ((xhigh $EQ xlow), (xhigh $NE xlow)) %2, 3

In this example, if xhigh equals xlow, then event number two is executed next, but if xhigh is not equal to xlow, then event number three is executed next.

After the flow control phase for event number one is written, the programmer writes the next interaction event, starting again with the prompt phase.

When writing a program, one would usually start by writing just the essential phases required to get the program running, eg. prompt, action and flow control. This simplifies things since the programmer can concentrate on the necessary elements and leave the default, help and check tables for later, after the basic program is running.

The last line of each phase table must have the number zero in the space normally occupied by the event number (eg. the first three columns). If the programmer chooses to leave the default, help and check tables to be written after the basic program is running, then these tables should each have one line, consisting of a zero anywhere in the first three spaces.

4.3 Further Suggestions

To repeat an event a certain number of times, the programmer can use a counting variable which is increased by one each time the event is executed. The flow control phase is then used to check whether the required number of repetitions have been executed, and controls the branching accordingly. (The next event number is set equal to the current event number until the repetitions are complete).
4.4 Simple Example

Table 4.1 on the following page contains a very simple program used to find the sum of two numbers. Note that if an event does not require a particular phase, the programmer simply does not put any lines with that event number in that phase table.

4.5 Entering and Running the Program

The following objects are necessary to run any program written using the C.E. System:

- correlate
- execute
- multch
- multchar
- edit
- locate
- phase

These objects should be copied from the workspace BSAM.INTER.

To enter the phase tables in the workspace, use the lines:

```
prom = multchar  (for the prompt table)
defa = multchar  (for the default table)
help = multchar  (etc)
```

where "multchar" is a function which helps you create the phase tables.

To edit phase tables, use the lines:

```
prom = edit prom  (for the prompt table)
defa = edit defa  (for the default table)
```

and follow the instruction you are given by the function "edit".
The phase tables:

<table>
<thead>
<tr>
<th>Prom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 'Enter the first number.'</td>
</tr>
<tr>
<td>2 'Enter the second number (default is zero).'</td>
</tr>
<tr>
<td>3 'Answer being calculated.'</td>
</tr>
</tbody>
</table>

Comments:

This is just a message to the user, and requires no response.

This means there is no default in this case. n2 will be given the value 0 if the user enters nothing.

<table>
<thead>
<tr>
<th>Defa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 default=0</td>
</tr>
<tr>
<td>2 n2='0'</td>
</tr>
</tbody>
</table>

This means there is no default in this case. n2 will be given the value 0 if the user enters nothing.

<table>
<thead>
<tr>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 'Enter the first of the two numbers to be added together.'</td>
</tr>
<tr>
<td>2 'Enter the second of the two numbers to be added together.'</td>
</tr>
</tbody>
</table>

This checks that the user entry is just a number. If the user has entered anything other than, or in addition to, a number, the prompt will be repeated.

<table>
<thead>
<tr>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+%n1$EP'1234567890. ')$LT($.,n1)</td>
</tr>
<tr>
<td>(+%n2$EP'1234567890. ')$LT($.,n2)</td>
</tr>
</tbody>
</table>

"sum" is a function (listed below), which adds the numbers n1 and n2.

Event 2 is to be executed next.
Event 3 is to be executed next.

<table>
<thead>
<tr>
<th>Act1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 n1 sum n2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 next=2</td>
</tr>
<tr>
<td>2 next=3</td>
</tr>
</tbody>
</table>

"sum<#>" 

| 1 | total=($EXx)+$EXy |
| 2 | 'The sum of the two numbers is ',$FMtotal |
| 3 | $>0 |
Once the phase tables have been entered, type "correlate", invoking the function "correlate" which sorts through the phase tables and records which row numbers in each table correlate to the various events. Then the program is ready to run, which is done by typing "execute". If any phase table is edited, resulting in row additions, row deletions, or changes in event numbers, then "correlate" must be run again.
CHAPTER 5
A COMPREHENSIVE EXAMPLE

5.1 Description

To examine the effectiveness of the C. E. System for writing interactive engineering programs, a fairly extensive example program was written. The program can be used to estimate the expected value, variance and skewness of any given function of random variables. The first, second and third moments of the parameter distributions are combined mathematically to yield estimated moments of the function.

This function moments program is useful whenever a user is evaluating a function whose parameters have uncertain values (cost estimating functions for example). For all but the simplest of functions, the calculations required to estimate the function moments are difficult or impossible to do by hand. But without carrying out these calculations, it is not possible to accurately estimate the effect of a parameter's variability on the moments of the function. Using the program presented here, the user gets his answer after a few minutes of interaction with the computer. In addition, the user can quickly make several runs while varying his parameter estimates or correlation coefficients, and quickly get an idea of sensitivities.

The benefits of using the C. E. System to write this function moments program are as follows:

i) It is relatively easy to incorporate interactive features such as help, default, escape, check and entry correction.
ii) The program is organized into numbered events and phase tables, which make the logic of the program more prominent, hence easier to follow and check.

iii) The program is modular in form, making it more readable and easier to test and debug.

5.2 Program Method

5.2.1 Moments of the Random Variables (Parameters)

The first two moments of a parameter's distribution function can be estimated from the following formulas:

\[
\text{mean} = \frac{1}{4} (L + 2M + H)
\]

\[
\text{variance} = \left( \frac{H - L}{2.65} \right)^2
\]

Where:  
\(L\) is a low estimate of the parameter, likely to be exceeded 90% of the time.
\(M\) is the most likely estimate of the parameter, likely to be exceeded 50% of the time.
\(H\) is a high estimate of the parameter, likely to be exceeded only 10% of the time.

These two equations are reasonably accurate for most distributions with central tendency, although they are less accurate for extremely skewed distributions. (Davidson, 1976, p. 1070)

The third central moment (skewness) of a parameter's distribution function can be estimated using the following procedure. Knowing the low (\(L\)), most likely
(M), and high (H) estimates of the parameter (eg. those values which have a 90%, 50%, and 10% chance of being exceeded, respectively), we can find a value x such that if x is added to L, M and H, the resulting distribution Y, is log normal.

ie.

\[ \ln(H + x) - \ln(M + x) = \ln(M + x) - \ln(L + x) \]

or \[ \frac{H + x}{M + x} = \frac{M + x}{L + x} \], by taking antilogs.

Solving for \( x \) gives:

\[ x = \frac{HL - M^2}{2M - H - L} \]

The following 5 steps can be used to estimate the third central moment (skewness) from the percentage points of a distribution:

1. Calculate \( \sigma_Y = \frac{H - L}{2.65} \) = approximate standard deviation of \( Y \)

2. Calculate \( X = \frac{HL - M^2}{2M - H - L} \) = shift of \( Y \)

3. Calculate \( \sigma_{\ln Y} = \frac{\ln(H + x)}{2.65} \) = approximate standard deviation of \( \ln Y \)

4. Calculate \( m_Y = (M + x) \in \frac{1}{2} \sigma_Y^2 \) = mean of \( Y \) (Ang, 1975, p. 103-105)

5. Calculate \( \mu_Y^{(3)} = \frac{3 \sigma_Y^4 + \sigma_Y^6}{m_Y} \) = third moment (skewness) of \( Y \) (Benjamin, 1970, p. 269)
5.2.2 **Mean, Variance and Skewness of a General Function**

For a function of several random variables

\[ Y = g(X_1, X_2, \ldots, X_n) \]

the mean, variance and skewness can be estimated by the following equations:

\[
E(Y) = g(\mu_{X_1}, \mu_{X_2}, \ldots, \mu_{X_n}) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 g}{\partial X_i \partial X_j} \text{Cov}(X_i, X_j) + \ldots
\]

\[
\text{Var}(Y) = \sum_{i=1}^{n} \left( \frac{\partial g}{\partial X_i} \right)^2 \text{Var}(X_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial g}{\partial X_i} \frac{\partial g}{\partial X_j} \text{Cov}(X_i, X_j) + \ldots
\]

\[
\mu_Y^{(3)} = \sum_{i=1}^{n} \left( \frac{\partial g}{\partial X_i} \right)^3 \mu_{X_i}^{(3)} + \ldots
\]

Where:

- \( \mu_{X_i} \) = the mean of parameter \( X_i \)
- \( \text{Cov}(X_i, X_j) \) = covariance of \( X_i \) with \( X_j \)
- \( \text{Var}(X_i) \) = variance of \( X_i \)
- \( \mu_{X_i}^{(3)} \) = third moment of \( X_i \)
- \( \mu_Y^{(3)} \) = third moment of the function \( Y = g(X_1, X_2, \ldots, X_n) \)

and the partial derivatives are evaluated at \( \mu_{X_1}, \mu_{X_2}, \ldots, \mu_{X_n} \).

These expressions are obtained from a truncated Taylor's series expansion of the general function \( g \) about the expected values \( \mu_{X_i} \), and by using the moment definitions. (Siddall, 1972, p. 49 - 51).
5.2.3 Estimating Partial Derivatives

The first and second partial derivatives (evaluated at the parameter means) are estimated numerically by calculating the effect on the function of small changes in each of the parameters.

For example:

1. To estimate \( \frac{\partial g}{\partial x_i} \),

   Let: \( A = \) function \( g \) evaluated at \( \mu_{x_1}, \mu_{x_2}, \ldots, \mu_{x_n} \)
   \( B = \) function \( g \) evaluated at \( \mu_{x_1}, \ldots, 1.001\mu_{x_i}, \ldots, \mu_{x_n} \)

   \[ \frac{\partial g}{\partial x_i} \approx \frac{B - A}{.001\mu_{x_i}} \]

2. To estimate \( \frac{\partial^2 g}{\partial x_i \partial x_j} \), \((i \neq j)\)

   Let: \( A = \) function \( g \) evaluated at \( \mu_{x_1}, \ldots, \mu_{x_n} \)
   \( B = \) function \( g \) evaluated at \( \mu_{x_1}, \ldots, 1.001\mu_{x_i}, \ldots, \mu_{x_n} \)
   \( C = \) function \( g \) evaluated at \( \mu_{x_1}, \ldots, 1.001\mu_{x_j}, \ldots, \mu_{x_n} \)
   \( D = \) function \( g \) evaluated at \( \mu_{x_1}, \ldots, 1.001\mu_{x_i}, \ldots, 1.001\mu_{x_j}, \ldots, \mu_{x_n} \)

   \[ \frac{\partial^2 g}{\partial x_i \partial x_j} \approx \frac{D - C - B + A}{.001\mu_{x_i} \cdot .001\mu_{x_j}} \]

   \[ \approx \frac{D - B}{.001\mu_{x_j}} - \frac{C - A}{.001\mu_{x_j}} = \frac{A - B - C + D}{.000001\mu_{x_i} / \mu_{x_j}} \]
3. To estimate \( \frac{\partial^2 g}{\partial X_i^2} \),

Let: 

\begin{align*}
A &= \text{function } g \text{ evaluated at } \mu_x, \ldots, \mu_x \\
B &= \text{function } g \text{ evaluated at } \mu_x, \ldots, 1.001 \mu_x, \ldots, \mu_x \\
C &= \text{function } g \text{ evaluated at } \mu_x, \ldots, 1.002 \mu_x, \ldots, \mu_x
\end{align*}

\[
\frac{\partial^2 g}{\partial X_i^2} = \frac{C - B}{.001 \mu_x} - \frac{B - A}{.001 \mu_x} = \frac{A - 2B + C}{.000001 \mu_x^2}
\]

5.2.4 Program Outline

The program is capable of estimating the first three moments of any explicit function of up to ten random variables. As many as five different functions can be stored in one workspace. A function can take the form of either a single mathematical expression, or a subroutine containing a series of calculations. The user of the program supplies the function, the names of all the variables, his low, probable, and high estimates of each of the parameters, and his estimates of the correlation coefficients between each pair of parameters. The program returns the moments of the function and its parameters as well as the calculated covariances and partial derivatives. The program consists of 13 interaction events:

1. User enters name of function to be evaluated.
2. User enters description and units of dependent variable.
3. User enters names of all parameters.
4. User enters descriptions and units for all parameters.
5. User defines the function.
6,7. User enters low, probable, and high estimates of parameters.
8-11. User enters correlation coefficients.
12. Moments of parameters calculated, derivatives calculated.
13. Moments of function calculated results displayed.

5.3 **Program Listing**

The following six pages show the six phase tables, with comments added for explanation purposes. Each event starts with the prompt phase, and the driver program determines which of the other phases are executed, based generally on the user response.
TABLE 5.1

EXAMPLE: THE PROMPT TABLE

1. 'Enter the name of the dependent variable to be evaluated.'
2. 'This is generally an abbreviation and must have'
3. 'no more than three characters.'
1 answer=answer0=$#
2 'The dependent variable ',answer0,' has not yet been defined'
2 answer=answer1=$#
3 'Enter the units to be used for the dependent variable ',answer0,'.'
3 answer=answer2=$#
4 'Enter the names of all of the variables required to'
4 'calculate the dependent variable ',answer0,'. Enter them all on'
4 'one line, separated by blanks or commas. Each name must'
5 'consist of no more than three characters.'
5 answer=answer3=$#
6 'Enter a brief description of the variable ',indep[ndep;count;],'.'
6 answer=answer4=$#
7 answer=indep[ndep;count;]
5 'Enter the expression or the name of the function to be used to'
8 'calculate ',depen[ndep;],'. If entering an expression, enter'
8 'the right side only. (The expression must be in APL language'
9 'and is limited to a maximum of 80 characters in length.)'
9 answer=answer6=$#
6 'You will next be asked to enter your low ,most likely, and high estimates'
7 'of each of the parameters. The low estimate is the number you are 90% sure'
7 'will be exceeded. The most likely estimate is the number which has a 50%
8 'chance of being exceeded. The high estimate has only a 10% chance of being'
8 'exceeded. For each parameter, the three numbers should be entered on one'
9 'line, separated by blanks or commas. The three numbers may be entered in'
10 'any order. If any of the parameters is certain, you may save time if you'
11 'wish by entering just one number.'
11 'Enter your estimates of the parameter ',indep[ndep;count;]
11 'Description: ',desin[ndep;count;]
11 'Units to be used: '.unind[ndep;count;]
11 answer=indep[ndep;count;]
11 answer=answer7=$#
8 'Do you wish to enter or change any correlation coefficients?'
9 'Enter yes or no.'
9 answer=indep[ndep;count;]
10 answer=indep[ndep;count;]
10 answer=answer8=$#
11 'Enter the correlation coefficients in the table below. Just leave'
12 'blank any zero coefficients. Make sure that any numbers you'
12 'enter are within their proper column boundaries '
12 'as defined by the column headings.'
13 heading
10 indep[ndep;count;], ':',(5 2$FMtable[count;])
10 answer=indep[ndep;count;]
10 answer=answer10=$#
10 answer9[count;]=55$1Aanswer10
11 'coefficients being stored'
12 'means and variances of independent variables being calculated'
13 'expected value and variance of ',depen[ndep;],' being calculated.'
0
TABLE 5.2
EXAMPLE: THE DEFAULT TABLE

def a
1    default=0
2    answer1=answer2='not given'
3    default=0
4    answer4=answer5='not given'
5    default=0
7    default=0
8    default=0
0
TABLE 5.3
EXAMPLE: THE HELP TABLE

help
1 'Type in the name of the quantity you wish calculated.'
1 'If this is the first time you have entered this name,'
1 'then you will be asked to enter the expression required.'
1 'If the expression has already been stored then you will'
1 'next be asked for your estimates of the parameters.'
1 'The description may be just the full name of the variable,'
1 'or may be more detailed, although no more than 50 characters'
1 'will be stored.'
2 'The units may be just the full name of the variable,'
2 'or may be more detailed, although no more than 50 characters'
2 'will be stored.'
2 'The main purpose of asking for these descriptions is that'
2 'they will be used to inform or remind future users.'
2 'eg.: For the variable named Q the description could be 'flow''
2 'and the units could be '(m³)/s'. (You would not enter'
2 'the quotation marks though.)'
3 'For example, if the expression were y=x₁*x₂/x₃, the independent'
3 'would be x₁, x₂, and x₃, so you would enter ''x₁,x₂,x₃'' or'
3 '''x₁ x₂ x₃'' (without the quotation marks though).''
4 'Enter any description and units you choose, although only the'
4 'first 50 characters will be stored for the description, 20 for'
4 'the units. The main purpose of asking for these descriptions'
4 'is that they will be used as information or reminders for'
4 'future users.'
5 'For example, if your expression were y=x₁*x₂/x₃, you would'
5 'enter ''x₁*x₂/x₃'' (without the quotation marks).''
5 'If the expression is more than 80 characters long, place it'
5 'in a separate function.'
8 'If the parameters of the function are all linearly independent,'
8 'then the correlation coefficients are all zero.'
8 'In general, a correlation coefficient can take on any value'
8 'between -1.0 and +1.0, and is a measure of the degree of'
8 'linear correlation between two random variables.'
TABLE 5.4

EXAMPLE: THE CHECK TABLE

chec
1 \$answer$GT$3
1 'Variable names must have no more than three characters.'
1 \$answer$GT$3
1 check=0
2 \$answer1$GT$50
2 'The description you have just entered has been truncated to 50 characters.'
2 \$answer2$GT$20
2 'The units you have just entered have been truncated to 20 characters.'
4 \$answer4$GT$50
4 'The description you have just entered has been truncated to 50 characters.'
4 \$answer5$GT$20
4 'The units you have just entered have been truncated to 20 characters.'
5 \$answer6$GT$80
5 'The expression you have just entered has more than 80 characters.'
5 \$answer6$GT$80
5 check=0
7 \$EXanswer7$GT$3$OR\$EXanswer7$EQ$2
7 'Enter three numbers only: your low, most likely, and high estimates.'
7 \$EXanswer7$GT$3$OR\$EXanswer7$EQ$2
7 check=0
8 \$(answer8$EQ$'yes'$$NE$$3$AND\$answer8$EQ$'no'$$NE$$2
8 'You must enter either yes or no.'
8 \$(answer8$EQ$'yes'$$NE$$3$AND\$answer8$EQ$'no'$$NE$$2
8 check=0
TABLE 5.5

EXAMPLE: THE ACTION TABLE

```plaintext
act1
1  ndep=searchdep answer0
2  answer1 storedep answer2
3  nind=sortind answer3
4  desind[ndep;count;]=50$TAanswer4
4  unind[ndep;count;]=20$TAanswer5
4  count=count+1
5  expr[ndep;]=80$TAanswer6
7  $EX(indep[ndep;count;],',',answer7)
7  count=count+1
9  answer9=10 55$,'',
9  table=(nind,nind)$TAcoef[ndep;;]
10  count=count+1
11 storecoef answer9
12 mevar
12 deriv
13 exvar
13 results
10
```
TABLE 5.6
EXAMPLE: THE FLOW CONTROL TABLE

flow
1    next=((stored$E00), (stored$NEG))%2, 6
1    count=1
2    next=3
3    next=4
3    count=1
4    next=((count$GTn1nd), (count$LEn1nd))%5, 4
5    next=6
5    count=1
6    next=7
7    next=((count$LEn1ind), (count$GTn1nd))%7, 8
8    next=((+%(3$, answer8)$EQ'yes')$GE2), (+%(3$, answer8)$EQ'yes'$LT2))%9, 12
9    count=1
9    next=10
10   next=((count$LEn1nd), (count$GTn1nd))%10, 11
11   next=12
12   next=13
13   $>0
0
5.4 **Programming Details**

5.4.1 **Descriptions of Functions:**

- **SEARCHDEP:** checks whether the dependent variable just named by the user is in storage, or whether a new expression is to be entered.
- **STOREDEP:** stores the description and units of the dependent variable.
- **SORTIND:** stores the names of the parameters.
- **STORECOEF:** stores the correlation coefficients entered by the user.
- **MEVAR:** calculates the mean, variance and skewness of each parameter, based on the percentage points (low, probable and high estimates) provided by the user.
- **DERIV:** estimates all first and second partial derivatives of the function.
- **EXVAR:** estimates the expected value, variance and skewness of the function.
- **RESULTS:** displays the results of the calculations.

Listings of these functions are provided in the Appendix.

5.4.2 **Flow Control Procedures**

Usually the lines in the flow control table consist of either a simple assignment statement (eg. `next = 2`) or a branching statement (eg. `next = ((stored $EQ 0), (stored $NE 0)) % 2, 6`). The simple assignment statement is used when the next event is pre-set, regardless of the results of the present event (unless the user enters "escape"). The branching statement is used when the results of the present event determine which event to execute next. If an event is to be
repeated a certain number of times, often a counting variable will be required. The counter is given its initial value (usually 0 or 1) in the flow table of the previous event, and is incremented each time the current event action phase is repeated. The flow control phase tests whether the event has been repeated a sufficient number of times.

5.5 A Sample Run

The following run demonstrates many of the features which have been built into the function moments program. This run shows that it is practically impossible to make the program crash, although this depends on how thorough the programmer has been in providing check and default features. User input is indicated with an arrow to the left of the line.
TABLE 5.7
A SAMPLE RUN

1) This workspace can be used to estimate the mean, variance, and
   skewness of any given function of random variables.
2) The program will store up to 5 different functions, and each
   function may contain up to 10 parameters.
3) To begin execution, type the word 'execute'.
4) To erase all of the mathematical functions stored by the
   program, type the word 'clear' (not while the main program
   is executing though).
5) If you are uncertain as to how to respond to any particular
   prompt, type the word 'help'.
6) If at any time you wish to escape from the main execution
   program, type the word 'escape'.
7) Use the MTS log command to keep a record of a terminal session.
8) When data entry is complete (after entry of correlation coefficients),
   the output begins almost immediately, so be ready with
   the pause key if you wish to view the results as they appear.
9) If, at any time you would like to see this introduction
   again, type the word 'intro'.

execute
Enter the name of the dependent variable to be evaluated.
This is generally an abbreviation and must have
no more than three characters.

help
Type in the name of the quantity you wish calculated.
If this is the first time you have entered this name,
then you will be asked to enter the expression required.
If the expression has already been stored then you will
next be asked for your estimates of the parameters.
Enter the name of the dependent variable to be evaluated.
This is generally an abbreviation and must have
no more than three characters.

y
The dependent variable y has not yet been defined
Enter a brief description of the dependent variable y.

cost
Enter the units to be used for the dependent variable y.

dollars
Enter the names of all of the variables required to
calculate the dependent variable y. Enter them all on
one line, separated by blanks or commas. Each name must
consist of no more than three characters.

back
The dependent variable y has not yet been defined

Enter a brief description of the dependent variable y.

The description may be just the full name of the variable, or may be more detailed, although no more than 50 characters will be stored. For the units, up to 20 characters will be stored. The main purpose of asking for these descriptions is that they will be used to inform or remind future users.

eg.: For the variable named Q the description could be 'flow' and the units could be '(m³)/s'. (You would not enter the quotation marks though.)

The dependent variable y has not yet been defined

Enter a brief description of the dependent variable y.

cost

Enter the units to be used for the dependent variable y.

dollars

Enter the names of all of the variables required to calculate the dependent variable y. Enter them all on one line, separated by blanks or commas. Each name must consist of no more than three characters.

For example, if the expression were \( y = x_1 \times x_2 / x_3 \), the independent would be \( x_1 \), \( x_2 \), and \( x_3 \), so you would enter 'x1, x2, x3' or 'x1 x2 x3' (without the quotation marks though).

Enter the names of all of the variables required to calculate the dependent variable y. Enter them all on one line, separated by blanks or commas. Each name must consist of no more than three characters.

Enter a brief description of the variable x1.

Enter the units to be used for the variable x1.

inflation

Enter a brief description of the variable x2.

percent

Enter the units to be used for the variable x2.

interest

Enter a brief description of the variable x3.

percent

Enter the units to be used for the variable x3.

Enter the expression or the name of the function to be used to calculate y. If entering an expression, enter the right side only. (The expression must be in APL language and is limited to a maximum of 80 characters in length.)

There is no default option in this case.
For example, if your expression were \( y = x_1^2 / x_3 \), you would enter \( 'x_1^2 / x_3' \) (without the quotation marks). If the expression is more than 80 characters long, place it in a separate function.

Enter the expression or the name of the function to be used to calculate \( y \). If entering an expression, enter the right side only. (The expression must be in APL language and is limited to a maximum of 80 characters in length.)

\[(3 \times x_1^3) + (2 \times x_2^2) + x_3\]

You will next be asked to enter your low, most likely, and high estimates of each of the parameters. The low estimate is the number you are 90% sure will be exceeded. The most likely estimate is the number which has a 50% chance of being exceeded. The high estimate has only a 10% chance of being exceeded. For each parameter, the three numbers should be entered on one line, separated by blanks or commas. The three numbers may be entered in any order. If any of the parameters is certain, you may save time if you wish by entering just one number.

Enter your estimates of the parameter \( x_1 \)
Description: inflation
Units to be used: percent
5 6 8

Enter your estimates of the parameter \( x_2 \)
Description: interest
Units to be used: percent
10 12

Enter three numbers only: your low, most likely, and high estimates.

Enter your estimates of the parameter \( x_3 \)
Description: not given
Units to be used: not given
.1 .3 .4

Do you wish to enter or change any correlation coefficients?
Enter yes or no.

If the parameters of the function are all linearly independent, then the correlation coefficients are all zero.
In general, a correlation coefficient can take on any value between -1.0 and +1.0, and is a measure of the degree of linear correlation between two random variables.

Do you wish to enter or change any correlation coefficients?
Enter yes or no.

Enter the correlation coefficients in the table below. Just leave blank any zero coefficients. Make sure that any numbers you enter are within their proper column boundaries as defined by the column headings.
Coefficients being stored.
Means and variances of independent variables being calculated.
Expected value and variance of $y$ being calculated.

Results for expression: $y = (3 \times x_1^3) + (2 \times x_2^2) + x_3$

Estimates of parameters:

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Probable</th>
<th>High</th>
<th>Mean</th>
<th>Variance</th>
<th>Skewness</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>5.000E00</td>
<td>6.000E00</td>
<td>8.000E00</td>
<td>6.250E00</td>
<td>1.282E00</td>
<td>2.323E00</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1.000E01</td>
<td>1.200E01</td>
<td>1.400E01</td>
<td>1.200E01</td>
<td>2.278E00</td>
<td>0.000E00</td>
</tr>
<tr>
<td>$x_3$</td>
<td>1.000E01</td>
<td>3.000E01</td>
<td>4.000E01</td>
<td>2.750E01</td>
<td>1.282E02</td>
<td>2.323E03</td>
</tr>
</tbody>
</table>

Correlation coefficients:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1.00</td>
<td>.80</td>
<td>.50</td>
</tr>
<tr>
<td>$x_2$</td>
<td>.80</td>
<td>1.00</td>
<td>.70</td>
</tr>
<tr>
<td>$x_3$</td>
<td>.50</td>
<td>.70</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Covariances:

Between $x_2$ and:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1.367034532</td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.06407974368</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.1196155215</td>
<td></td>
</tr>
</tbody>
</table>

First derivative of $y$ with respect to:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>351.9141797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>48.024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Second derivative of $y$:

with respect to $x_1$ and:
$$x_1 : 112.6125$$

with respect to $x_2$ and:
$$x_2 : 4.000000001$$

with respect to $x_3$ and:
$$x_1 : 0$$
$$x_2 : 0$$
$$x_3 : 0$$

Expected value of $y$ is 1097.415458
Variance of $y$ is 302303.2533
Standard deviation of $y$ is 549.8211103
Third moment of $y$ is 101250161.1
Skewness coefficient of $y$ is 0.6091601591

Expressions stored in this workspace:

```
y = (3*x1@3) + (2*x2@2) + x3
```

You may store up to 5 expressions in each workspace.
CHAPTER 6
CONCLUSIONS AND FUTURE DEVELOPMENTS

This thesis has presented and demonstrated a programming tool which can simplify and speed up the process of writing the dialogue portion of an interactive computer program. The author believes this tool has potential in the field of engineering to help engineers write high-quality interactive programs. Although at first it may seem that the method only complicates matters, once the few simple rules have been learned, the interactive part of a program can be written quickly and efficiently. It is estimated that the programmer would begin to save time almost immediately, probably with the second program written following the C. E. System. Most of the tedious, repetitive aspects of programming the dialogue are taken care of by the driver ("execute") program, and the programmer provides only the particular prompts, checks, action, etc. he wants in each interaction event.

Future developments of the C. E. System would mainly consist of adapting the method to languages more commonly used with micro and mini-computers, for example, Basic, C, and Fortran. Since the C. E. System takes advantage of some unique features of APL, major alterations will be required to transfer the method to other languages.
LIST OF REFERENCES


APPENDIX

SELECTED PROGRAM LISTINGS
"execute<\#>"
  
  "execute; iiii; jjjj; kkkk; nnnn; xxxx"

[1] /* This program executes the phase tables.
[2] */ An interaction event generally consists of 6 phases.
[3] */ although not all 6 phases will be used in each event.
[4] /* iiii = the phase number:
[12] */ Event #1 is executed first.
[13] */ 'next' is the number of the event to follow the current one.
[14] */ next=1
[15] */ backup=0
[16] */ Any response from the user is temporarily stored in the character array called 'answer'.
[17] */ kkkk is the third coordinate of the array 'corr', in which the row numbers corresponding to each event-phase are stored.
[18] */ answer is initially set to 'no resp req' to handle events where no response is required from the user.
[19] */ answer='no resp req'
[20] */ jjjj=next
[21] */ xxxx is a vector which determines the prompt table columns to be executed.

[22] xxxx=3$DR$.($,prom)
[23] */ nnnn is the number of the prompt table row to be executed.
[24] */ If nnnn=0 then the prompt phase is complete.
[26] $prom[nnnn; xxxx]
[27] */ kkkk=kkkk+1
[28] */ Execute lines until the prompt phase is complete.

[29] $>rep5
[30] */ If the user gives no response.
[31] */ If no response is required.
[32] */ If the user enters 'escape'.
[33] */ If the user enters 'help'.
[34] */ If the user enters anything else (numbers, characters, etc.).
[35] */ Go to prompt phase of previous event.
L4: next = j jj j-1
count = 1
backup = 1
$>rep7
$* Proceed to action phase.
L6: i i i i = 5
$L10
$L10
$* Proceed to help phase.
L8: i i i i = 3
$L10
$L10
$* Default phase begins here.
kkkk = 1
default = 1
$* xxxx determines the column numbers of defa to be executed.
xxxx = 3$DR$.($, defa)[2]
$* nnnn is the line number of defa to be executed.
rep4: nnnn = '$, corr[2; jjjj; kkkk]
$* If nnnn = 0 then the default phase is complete.
$nnnn$E00)%L17
$EXdefa[nnnn; xxxx]
kkkk = kkkk + 1
$* Continue to execute lines until the default phase is complete.
$>rep4
L17: $(default$E00)%L18
$L13
$L13
L18: 'There is no default option in this case.'
$* Prompt phase is repeated.
$>rep7
$* Check phase begins here.
L9: kkkk = 1
check = 1
$* xxxx determines the column numbers of chec to be executed.
xxxx = 3$DR$.($, chec)[2]
$* nnnn is the row number of chec to be executed.
rep8: nnnn = '$, corr[4; jjjj; kkkk]
$* If nnnn = 0 then the check phase is complete.
$nnnn$E00)%L15
$* condition = 1 or 0, depending on whether the test line
$* in the check array is false or true, respectively.
$EXchec[nnnn; xxxx]
kkkk = kkkk + 1
$* If the test line is true, then the next line in the check
$* table is executed. If the test line is false, then the next
$* line in the check table is skipped.
$nnnn$E00)%L16
$EXchec[nnnn; xxxx]
L16: kkkk = kkkk + 1
101  $> \text{rep8}
102  $* If \text{check}=0 \text{ then repeat the prompt phase of the same event.}
103  $* If \text{check}=1 \text{ then proceed to action phase.}
104  L15:$>(\text{check}\$EQ0)$\text{rep7}
105  $* \text{Check phase is complete.}
106  $>L13
107  $* \text{Help, action, and flow phases are all executed by this next section.}
108  L10:kkkk+1
109  $* \text{xxxx is a vector which determines the phase table columns to be executed.}
110  xxxx=3$DR$.($.$EXphase[iii:][2]
111  $* \text{nnnn is the number of the phase table line to be executed.}
112  \text{rep6:nnnn}='$$.corr[iii:jjjj;kkkk]
113  $* \text{If nnnn}=0 \text{ then the phase is complete.}
114  $(nnnn$EQ0)$\times L11
115  $* \text{Execute line nnnn of the phase table.}
116  EX($EXphase[iii:])[nnnn;xxxx]
117  kkkk=kkkk+1
118  $* \text{Execute lines until the phase is complete.}
119  $>\text{rep6}
120  $* \text{If in action phase, proceed to flow control phase.}
121  L11:$>(iii:$$EQ5)$\times L12
122  $* \text{If in help phase, go back to prompt phase.}
123  $(iii$$EQ3)$\times\text{rep7}
124  $* \text{If in flow control phase, proceed to next prompt phase.}
125  $(iii$$EQ6)$\times\text{rep7}
126  L12:iii=6
127  $>L10
128  L13:iii+5
129  $>L10
"correlate<#>"
  " correlate:x;xx:nev; i:n:k;j
[1] $* This program stores, in the array named 'corr', the row
[2] $* numbers corresponding to each event-phase combination.
[3] $* The array 'corr' is used, when the main program is executing,
[4] $* to determine which row numbers are to be executed for a
[5] $* particular event-phase. This is preferable to having the main
[6] $* program search each phase table for every event.
[7] x= 1 2 3
[8] xx=3$OR$.70
[9] $* The size of 'corr' is determined by:
[10] $* [ # of phases ; # of events ; max. # of lines in any event-phase]
[11] corr= 5 13 10 $,0
[12] $* i=phase number=first coordinate of the array 'corr'.
[13] i=1
[14] $* n=row number of phase table
[15] rep2:n=1
[16] $* k=the third coordinate of the array 'corr'.
[17] repO:k=1
[18] $* j=event number=the second coordinate of the array 'corr'.
[19] j=$EX($EXphase[i;]){n;x}
[20] $* The row number of the event-phase is stored in 'corr'.
[21] repI:corr[i;j;k]=n
[22] $* If the event number of the next row is the same, then
[23] $* n and k are increased by one. If the event number of the next
[24] $* row is different, then n is increased by one.
[25] $>(($EX($EXphase[i;]){n+1;x})$NEj)%L1
[26] n=n+1
[27] k=k+1
[28] $>rep1
[29] L1:n=n+1
[30] $* If the event number is 0, then go to the next phase.
[31] $>(($EX($EXphase[i;]){n;x})$EO0)%L2
[32] $>rep0
[33] $* Repeat for each of the 6 phases.
[34] L2:$>(($EX06)]%O
[35] i=i+1
[36] $>rep2
FUNCTIONS USED IN THE COMPREHENSIVE EXAMPLE

"searchdep#x"
"xxxx=xxxx searchdep answerO;iiii;ndep;string;jjjj;last:xxxx"
[1] " * This program searches the array 'depen' (which contains the"
[2] " * names of all of the dependent variables currently stored)
[3] " * for the dependent variable just named by the user.
[4] " * If the name is found, then its position in the array 'depen'
[5] " * is recorded, and the number of parameters corresponding to
[6] " * that dependent variable is determined.
[7] " * If the name is not found, and there is no more space
[8] " * available to store a new expression (up to 5 are permitted).
[9] " * then the user is sent an appropriate message.
[10] " * If the name is not found, and there is space available for a
[11] " * new expression, then the new name is added to 'depen'.
[12] " * If the name is found, but no corresponding parameter names
[13] " * are found in 'indep', then an appropriate message is sent
[14] " * to the user.
[15] answerO=3
[16] iiii=1
[17] rep1:$>((iiiigT5)%L2
[18] $>((+%answerO$EQdepen[iiii])%=L1
[19] iiii=iiii+1
[20] $>rep1
[21] L1:xxxx=iiii
[22] stored=1
[23] string,indep[iiii;
[24] jjjj+0
[25] rep2:1ast=_i$TAnswerstring
[26] string=_1$DRstring
[27] $>(last$NE' ')%=L3
[28] jjjj=jjjj+1
[29] $>(jjjj$GE32)%L6
[30] $>rep2
[31] L3:nind=MA(30-jjjj)/3
[32] $>0
[33] L2:kkkk=1
[34] stored=0
[35] rep3:$>+(%depen[kkkk];$EO' ')%=L4
[36] kkkk=kkkk+1
[37] $>(kkkk$GE6)%L5
[38] $>rep3
[39] L4:xxxx=kkkk
[40] depen[kkkk]=answerO
[41] $>0
[42] L5:'You have already stored five expressions in this workspace,'
[43] 'and no more room is available. Stop execution by typing escape'
[44] 'in response to the next prompt. Then use the clear or erase'
[45] 'command to clear expressions you no longer need. If you do not'
[46] 'want to erase any of the expressions stored, then copy INTER'
[47] 'to a new workspace.'
[48] 'xxxxx=5
[49] $>0
[50] L6:'There is no expression stored for the variable you have named.'
[51] 'You must stop execution, erase the variable name from storage,'
[52] 'and re-define the variable and its expression.'
[53] $>0
"storedep<#>

1) * This program stores the description of the dependent variable
2) * in the array 'desdep' and the units in the array 'undep'.
3) desdep[ndep]=50$TAanswer1
4) undep[ndep]=20$TAanswer2
5) "$>0

"sortind<#>

1) * This program sorts the character string containing the parameter
2) * names into one string per name, and stores them in the array 'indep'.
3) * The program returns the number of parameter names found.
4) iiii=ndep
5) yyyyy=jjjjj=0
6) limit+2*$,answer
7) count=0
8) rep1:xxxx=1$TAanswer
9) answer=1$DRanswer
10) count=limit+1
11) $>((count$EQlimit)%0
12) $>((xxxx$EQ' ')|(xxxx$EQ','))%rep1
13) jjjj=jjjj+1
14) zzzz=
15) rep2:zzzz=zzzz,xxxx
16) xxxx=1$TAanswer
17) answer=1$DRanswer
18) count=limit+1
19) $>((count$EQlimit)%0
20) $>((xxxx$EQ' ')|(xxxx$EQ','))%L1
21) $>rep2
22) L1:indep[iiii;jjjj]=3$TAzzzz
23) yyyyy=jjjj
24) $>rep1

"storecoef<#>

1) * This program stores the correlation coefficients in the array 'coef'.
2) iiii=1
3) rep0:jjjj=0
4) rep1:answer9[iiii]=5$ROanswer9[iiii]
5) aaaa=5$TAanswer9[iiii]
6) jjjj=jjjj+1
7) $>((aaa$EQ' ')$E05)%L1
8) coef[ndep;iiii;jjjj]=coef[ndep;jjjj;iiii]=$EXaaa
9) L1:$>((jjjj$EQnind)%L2
10) $>rep1
11) L2:iiii=iiii+1
12) $>(iiii$GTnind)%O
13) $>rep0
This program estimates the means, variances, and third moments of the parameters.

'mind' is a vector containing the means of the independent variables.

'vind' is a vector containing the variances of the independent variables.

'order' is a nindx3 matrix containing the ordered values.

order=(nind,3)

mind=10

vind=10

thmom=10

jjjj=1

If only a single value has been entered for the parameter, go to line L1.

Put the three estimates in increasing order.

index=(ind,3)

zzzz=(ind,3)

order[jjjj]=zzzz

Calculate the mean.

Calculate the variance.

Test for symmetry of the parameter's distribution.

Calculate the standard deviation.

Calculate the shift required to fit a lognormal distribution.

Calculate the standard deviation of the log.

Calculate the mean.

Estimate the third central moment.

Calculate the covariances.

* Calculate the mean.

* Calculate the variance.

* Test for symmetry of the parameter's distribution.

* Calculate the standard deviation.

* Calculate the shift required to fit a lognormal distribution.

* Calculate the standard deviation of the log.

* Calculate the mean.

* Estimate the third central moment.

* Calculate the covariances.

* Calculate the mean.
This program calculates numerically the first and second partial derivatives of the expression w.r.t all of its parameters. 

Rep1: $i = 1$

Rep2: $j = 1$

The mean of parameter $i$ is: 

$t_1 = \text{EXindep}[ndep;iiii;]$ 

The case where $i = j$ is handled separately.

Rep1: $i = (i + 1.001 * i)$ and $j = 1$.

$t_2 = \text{EXindep}[ndep;jjjj;]$ 

Rep2: $j = (j + 0.001)$.

$a = \text{EXexpr}[ndep;]$ 

Parameter $i$ is increased by 0.1%.

$EX(indep[ndep;iiii;], ' = 1.001*', indep[ndep;iiii;])$

Parameter $j$ is increased by 0.1%.

$EX(indep[ndep;jjjj;], ' = 1.001*'; indep[ndep;jjjj;])$

The mean of parameter $j$ is: 

$t_2 = \text{EXindep}[ndep;jjjj;]$ 

The second derivative is estimated numerically.

$sder[ndep;iiii;jjjj] = (aaaa+cccc-2*bbbb)/1E_6$

Parameter $i$ is reset to its mean value.

$EX(indep[ndep;iiii;], ' = tttt1')$

The process is repeated for all combinations of $i$ and $j$ for which $j = i + 1$.

$iiii = ii + 1$

$s = (iii<$ind$);
This program prints the results of all of the calculations.

Results for expression: `depen[ndep;] = expr[ndep]`

Estimates of parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Low</th>
<th>Probable</th>
<th>High</th>
<th>Mean</th>
<th>Variance</th>
<th>Skewness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Correlation coefficients:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Correlation Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Covariances:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Covariances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First derivative of `depen[ndep;]` with respect to:

Second derivative of `depen[ndep;]` with respect to:
This program is used to calculate the expected value, variance, and skewness of the dependent variables.

sd2 = sder(ndep;);  

exval = (EXexpr + 0.5 * (sder(ndep;)); covar(ndep;)) * sder2;  

var = term1 + term2 + term3;  

skew = fder(ndep;); thmom  

skewco = skew / vari^{1.5}  

$>$ exvar<#>