Särtryck av konferensens föredragspublikation
A STATISTICAL PROGRAMMING LANGUAGE SURVO 66

Timo Alanka, Seppo Mustonen, Martti Tienari
Helsingfors Universitet
Helsingfors

Introduction

The authors have co-operated since 1960 in programming statistical applications for electronic digital computers. We have worked through the usual stages of system development in this application field. We defined standard statistical programs for different methods requiring extensive computation: correlation, regression, factor analysis and other multivariate methods. We soon noticed the value of a common data standard for different programs, because many statistical problems required the application of different methods, often in an unpredictable sequence. It is, of course, of great practical importance to be able to keep the data material just once although it is subsequently used in different statistical analysis programs. In the same way the intermediate results e.g. correlation matrices should be in a form conforming to the input requirements of the analysis programs. We also found it practical to compute different elementary statistical results e.g. means, variances and cross tabulations, of the data keypunched mainly for the subsequent heavy computer analysis. In this way we came to an integrated statistical program library for our computer, an Elliott 8038 with 8196 words of 36 bit core memory. Similar integrated libraries, statistical program packages, have been reported for many computers e.g. IBM 7090 [1, 2] and IBM 1401 [4].

In the course of the extensive statistical computing service which has been maintained using the integrated statistical program library,
we have been observing the behaviour of the scientists using computer services for their statistical research. The working habits of these scientists were changing. They dared to collect much more extensive data material, more attributes and more items than earlier. During the time of manual statistical computations the statisticians were close to the data. A decision to perform some statistical analysis came after careful reasoning. Now, the scientist -- once he has decided to make use of the computer -- is usually more careless. He often experiments with different analysis methods, sometimes even without any clear a priori hypothesis. The scientist is also often unable to look carefully at his data. The computer service must therefore provide for his thorough quality control, cross tabulation and plotting of the data. In manual computation one uses every conceivable trick and short-cut to avoid extensive straightforward computations. A computer user is tempted to exactly the opposite: a straightforward standard computation is no problem, whereas any fresh, simple idea might lead to slow and costly special programming or to manual computing. It is now wise to guide statistical work in such a way that one can make use of the standard statistical programs.

The observations presented above lead us to aim for radically more flexible statistical programs. There exist, however, some factors which limit the possibilities of an integrated chain of statistical standard programs. Added flexibility usually means added complexity of use; we would hope that the scientist need not be a computer specialist to be able to define in computer language his processing requirements. Many problems are left to the user of any integrated statistical program library with flexible processing facilities. The user is expected to furnish parameters for the programs in the statistical package. He must consider and fit together the different data structures used in the package, and required in his research. It is very difficult to provide adequate mnemonic labelling of different variables and results. A statistical package is usually unable to perform any parallel processing: each program handles the data completely before it is able to deliver control to the next program.

In the end, we felt that the only way to achieve drastically more flexibility in the statistical research process was to create a statistical language, which would be compatible with usual statistical methods. A system was to obviate any methods consumption by the computer.

The process of implementation stages. In 1964, the first system was subsequently implemented in the SURVO 65, which we could not agree with. Finally, a new design SURVO 66 was released in December 1967 for general statistical data analysis. The system is now in use at several
language, which would be comprehensible to any scientist familiar with usual statistical methods. A specific design goal of the system SURVO 66 was to obviate any methods consulting staff between the scientist and the computer.

The process of implementing our ideas proceeded through several stages. In 1964, the first system design named SURVO 66 was elaborated. It was subsequently implemented in a reduced form which we called simply a generalized cross-tabulating system. The following stage was a plan called SURVO 65, which we could not agree to be worth the cost of implementing. Finally, a new design SURVO 66 emerged and was implemented. The system was released in December 1967 for computing service. The handbook of this general statistical data analysis system is published in Finnish [3]. The system is now in use at several university computer centers in Finland.
Basic principles of the language SURVO 66

SURVO 66 is a programming system tailored to the data processing requirements of elementary statistics. The data exposed to an analysis must conform to a special data standard. We presume that the data consists of numbers arranged in a data matrix. A row of the matrix, data vector, represents the data from an object under observation: a person, a unit of sample, a product item, a single experiment. The attributes of the objects are variables: numbers characterizing the object, test scores, replies to questions, measurements. Most statistical data materials can be organized according to this standard. To this end, any qualitative information must be coded in a numerical form; missing observations of attributes are coded as out-of-range numbers. If no symbolic names have been given to the variables, the system calls then X1, X2, X3, ..., XM.

The tasks which a SURVO program is able to do are:
1. Quality control of the data (range of variables, interrelationships of variables).
2. transforming the data,
3. estimation of basic statistical parameters: means, medians, standard deviations, fractiles, correlations,
4. frequencies and cross tabulations,
5. performing tests of significance: t-test, $\chi^2$ - test,
6. simple statistical analysis: analysis of variance, regression analysis.

A task can be carried out selectively: the operations are applied only to the data vectors conforming to a predetermined condition. This feature allows, in effect, even handling of overlapping groups of data and cooperating different data groups in a single computer run. All the objects referred to: variables, tables, correlation matrices, classification scales, classes, conditions etc., can be given alphanumeric names. This is in order to make the SURVO program easier to read. This practice also enables the SURVO system to label the result quantities in an easily comprehensible way.

For the sake of efficiency the SURVO 66 system applies a sort of parallel processing. The data material is usually too extensive to be stored in the fast random access data medium: magnetic tape, punched cards. The packaged computation of elementary statistical procedures is usually too small statistical computation for the available one vector small statistical computation independent programs. Therefore, the programming to several parallel statistical analysis programs.

As an introductory example, consider the computation of means of 20 variables from 100 observations. In the SURVO language, this task is

```
M@20 N@100
MEAN@XI-X20
END
```

The SURVO program is punched on punched cards.

The run of a SURVO program follows the translated program, T2: first, the program stores the program. Storage is cleared. The second stage, T2: the program reads the data file. Program Ti: translation of SURVO program, the translated program, Ti: for the output of results. During Ti: the result of the computation is stored on punched tape. The information in any instance, the instruction COR of squares and products of the program. When all observation vectors of the SURVO program is obeyed once cumulated tables for the last is generated.

In a sense the SURVO...
stored in the fast random access memory. It must be held on an external
data medium: magnetic tape, punched cards or paper tape. For any standard
package computation of elementary statistics it is sufficient to have
the data available one vector at a time. The cost of input makes many
small statistical computations uneconomical, if they must be processed by
independent programs. Therefore in the SURVO 66 system the data is exposed
to several parallel statistical operations within one data input cycle.

As an introductory example we give a program which computes the
means of 20 variables from 100 observations. The description of this job
in SURVO language is simply:

```survo
N@20 N@100
MEAN@XI-X20
ENDS
```

The SURVO program is punched on paper tape and the data on paper tape or
punched cards.

The run of a SURVO program can be divided into three stages:
T1: translation of SURVO program, T2: input of the data under control of
the translated program, T3: final computations on cumulated tables and
output of results. During T1 the SURVO system program reads, checks and
stores the program. Storage space is allocated and sum locations are
cleared. The second stage, T2, consists of reading the data. The dimensions
of the data matrix are read first, as well as a set of parameters describ-
ing the details of data format. While the data matrix is being read, just
one observation vector is in the fast memory at the same time. The whole
SURVO program is obeyed for each observation vector. Each SURVO instruction
collects the information it needs from the current observation vector. For
instance, the instruction CORREL collects a frequency count and sums, sums
of squares and products of the variables referred to in the CORREL instruc-
tion. When all observation vectors have been read and treated in T2, the
SURVO program is obeyed once more. At this stage the computer goes over the
cumulated tables for the last time to get the final results and the output
is generated.

In a sense the SURVO instructions have a dual interpretation. In
stage T2 they lead to different internal function than in stage T3. From the point of view of the statistician, however, the instructions have a single meaning: give the defined results on the basis of the observation matrix.

Programming in SURVO

A SURVO program consists of a sequence of instructions written in the language of the computer. The instructions are of the form

(operator) </

The delimiter symbol @ is used to delimit instructions. The operator tells what type of instruction is used, e.g., a mnemonic operation code, e.g., @ has different requirements for the operator, which are necessary references which are part of the instruction. The instructions of which they are written in the program are defined in the program. Distinct instructions are used in each other. However, the SURVO program is used in an instruction.

The identifiers used in each program are written in the program as END@. Distinct identifiers are used in each other. However, the identifiers are used in an instruction.

The identifiers used are letters, digits, and special symbols which are terminated by the character @. The identifier is unlimited, the identifier is unlimited, the identifier is unlimited.

A variable in the input variable is automatically called a variable. The order number of the variable is given in the mnemonic programs and results. CALL-instructions. E.g. the instruction

```
CALL@ X3 WT
X7 L2
```

resumes X3 and X7 as WEIGHT and
Programming in SURVO 66

A SURVO program consists of the name of the program and of a sequence of instructions written in the SURVO 66 language. The name of the program is used in the output phase to label each page of results. The instructions are of the form

\[
\text{\textlangle operator\textrangle \text{@} \text{\textlangle list of parameters\textrangle}
\]

The delimiter symbol @ is used simply to terminate the operator identifier. The operator tells what should be done, and is expressed by a mnemonic operation code, e.g., MEAN, CORREL, END. The list of parameters has different requirements for different instructions. It establishes the necessary references which are needed in order to obey the instruction.

The instructions of a SURVO program are obeyed in the same order in which they are written in the program. The last instruction of any SURVO program is END@. Distinct instructions are to a large degree independent of each other. However, the SURVO objects (variables, tables, conditions), which are used in an instruction, must be defined in an earlier instruction.

The identifiers used in the list of parameters consist of letters, digits and special symbols (the six symbols @:-<> excepted). They are terminated by the characters "space" or "line feed". The length of an identifier is unlimited; the system, however, considers only the first six characters. The program constants conform to usual programming language conventions.

A variable in the SURVO language may have several names. Each input variable is automatically associated with a standard name Xi, where i is the order number of the variable in the data vector. In order to get mnemonic programs and results it is customary to rename the variables using CALL-instructions. E.g., the instruction

\[
\text{CALL@ Xi \ WEIGHT}
\]

\[
\text{X7 \ LENGTH}
\]

renames Xi and X7 as WEIGHT and LENGTH respectively. New variables and
other SURVO objects are named in the same instruction where they are defined.

There exist means in the SURVO language to shorten long lists of names. The list X1, X2, …, X20 can also be referred to by X1-X20. Other group references can be defined using the NAME-instruction. For instance the instruction

```
NAME@ PART1 X1 X2 X5 X6 X9
© PART2 X2 X9 X7 X8 X10
© ALL  PART1 PART2
```

gives an easier means of reference: PART1 for variables X1, X2, X5, X6, X9, PART2 for variables X2, X9, X7, X8, X10 and an alternative reference ALL for X1-X10.

The variables and constants in SURVO 66 language are integers or fractions which are internally represented as integers scaled with a power of ten. There may also appear Boolean variables. No floating point variables are used, although the system makes use internally of floating point computing. The system is easiest to apply when all the data consists of integers: scaling requires some consideration by the programmer.

The parameter list of a SURVO instruction gives the SURVO objects to be operated upon. It also contains speciality parameters to specify the operation in more detail. The speciality parameters are expressed in the format

```
<speciality identifier> : <parameter identifier>.
```

In the following table we define the different speciality identifiers. They cannot all be used in connection with every SURVO instruction.

<table>
<thead>
<tr>
<th>speciality identifier</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>give a name to be defined</td>
</tr>
<tr>
<td>S</td>
<td>give the scale of a new variable</td>
</tr>
<tr>
<td>L</td>
<td>define the bound for a variable</td>
</tr>
<tr>
<td>U</td>
<td>define the bound for a variable</td>
</tr>
<tr>
<td>IF</td>
<td>define the selective condition</td>
</tr>
<tr>
<td></td>
<td>determine the instr. should be</td>
</tr>
<tr>
<td></td>
<td>omitted for current data</td>
</tr>
<tr>
<td></td>
<td>suggest the method which is</td>
</tr>
<tr>
<td></td>
<td>better suited for the standard</td>
</tr>
<tr>
<td>Function</td>
<td>Parameter Identifier</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>give a name to a new SURVO object to be defined in the instruction</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>give the scaling of a new SURVO-variable</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>define the lower bound for a variable</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td>define the upper bound for a variable</td>
</tr>
<tr>
<td><strong>IF</strong></td>
<td>define the selective condition which determines whether the instruction should be obeyed or omitted for the current data vector</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>suggest the use of a method which is better suited than the standard method</td>
</tr>
</tbody>
</table>
**Control instructions**

- **END**: terminate the program list.
- **WAIT**: suspend program operation if the condition is satisfied.
- **STOP**: transfer to the next data vector if the condition is satisfied.
- **N**: give the length of the data vector ( = m). This is usually the first instruction of any SURVO program.
- **N**: give the number of data vectors ( = n). This instruction may be omitted.
- **SPACE**: set the width of the result print-out to k characters.
- **COMMENT**: the program can be made more readable by using comments.
- **NAME**: give a name to a group of variables.
CALL@ \( u_i \) \(<\text{identifier} > \) give the variables \( u_1, \ldots, u_r \)
new names

DEF@ \( u_1, u_2, \ldots, u_r \) the variables \( u_1, u_2, \ldots, u_r \)
are defined as having the properties defined by the speciality parameters. The variables will
be checked for these properties during phase T2 of the SURVO system.

\( L: \) \(<\text{lower bound}> \)
\( U: \) \(<\text{upper bound}> \)
\( S: \) \(<\text{scale}> \)

Transformation instructions
The transformations can be performed selectively using IF-

\begin{align*}
\text{SET@} & \quad u \quad u_1 \\
\text{ADD@} & \quad u \quad u_1 \quad u_2 \\
\text{SUB@} & \quad u \quad u_1 \quad u_2 \\
\text{MULT@} & \quad u \quad u_1 \quad u_2 \\
\text{DIV@} & \quad u \quad u_1 \quad u_2 \\
\text{MOD@} & \quad u \quad u_1 \\
\text{SQRT@} & \quad u \quad u_1 \\
\text{LOG@} & \quad u \quad u_1 \\
\text{EXP@} & \quad u \quad u_1 \\
\text{MAX@} & \quad u \quad u_1 \quad u_2 \quad \ldots \quad u_r \\
\text{MIN@} & \quad u \quad u_1 \quad u_2 \quad \ldots \quad u_r \\
\text{ORDER@} & \quad u \\
\text{LAG@} & \quad u \quad u_1 \quad \ldots \quad u_k \\
\text{PRINT@} & \quad u_1 \quad \ldots \quad u_r \\
\end{align*}

\( u_1 \) \text{ is the value of the variable } \quad u_1 \text{ in the data vector which}
lies in the data matrix } \quad k 
rows earlier than the current
vector.

A transformed data matrix is printed using the specified
M: \{\text{number of output device}\} 

\text{output device. The vectors}
to be included in the trans-
formed new matrix can be
selected through the IF-
condition.

\text{Boolean instructions}

\begin{align*}
\text{EQUAL} & : e \ = \ u_1 = u_2 \\
\text{LESS} & : e \ = \ u_1 < u_2 \\
\text{LESS} & : e \ = \ u_1 \leq u_2 \\
\text{BETWEEN} & : e \ = \ u_1 \leq u_2 \leq u_3 \\
\text{OR} & : e_1 \ldots e_r \\
\text{AND} & : e_1 \ldots e_r \\
\text{NOT} & : e_1
\end{align*}

\text{Classification and tabulating instructions}

The CLASS-instruction is used to define a set of rules by which
the variable values are mapped to class names or class number.
Every set of classification rules is named to allow subsequent reference.
The classification facility is used in TABLE- and TRANSF-instructions. The
detailed format of the CLASS-instruction is

\text{CLASS} \langle \text{name of classification} \rangle 

\begin{align*}
\langle \text{class name } i \rangle \langle \text{lower bound} \rangle \langle \text{upper bound} \rangle \\
\langle \text{class name } i \rangle \langle \text{lower bound} \rangle \langle \text{upper bound} \rangle \\
M: \langle \text{classification method} \rangle \\
S: \langle \text{scale} \rangle
\end{align*}

The classification rule defined by a CLASS-instruction is
available for use with any variable stored in the scale defined in the
CLASS-instruction. The variable values \( x \) which fulfill the condition
\( a_i \leq x \leq b_i \) are mapped to the class \( I ( i = 1, \ldots, r ) \). The class names
may be partially identical; the classes may thus consist of several
distinct intervals. The class names may be any permissible SURVO identifiers.

The speciality parameter SHORT, FAST guides the compiler to
shorten addressing. This method is somewhat
memory. SHORT method applies a non-
and therefore allows maximal storage.

Closely associated with
transformation instruction. This
defines a new variable applying a
ew variable is the integer class
or a simple count 1, 2, \ldots if al
The format of the TRANSF instruction:

\text{TRANSF} \ u \ u_1 \ c \\
M: m \\
IF: \langle \text{condition} \rangle

where \( u \) is the new variable, \( u_1 = \) name of a classification rule defined
\( m = \) the value to be given if the \( i \).
The TABLE-instruction is for construction of one-way and two-
for the tabulating tasks with the same name are programmed applying conditions
given names for later reference. The CHI2-instruction can be used to
frequency table. The VARAN-instruction is the analysis of variance using me-
structure of the TABLE-instruction.
distinct intervals. The class names are either nonnegative integers or any permissible SURVO identifiers.

The speciality parameter \( M \) has two possible values: FAST and SHORT. FAST guides the compiler to apply direct value indexing in table addressing. This method is sometimes wasteful in using the computer core memory. SHORT method applies a normal search strategy in table handling and therefore allows maximal storage economy.

Closely associated with the CLASS-instruction is a variable transformation instruction. This instruction is called TRANSF, and it defines a new variable applying a classification rule. The value of the new variable is the integer class number defined in a CLASS instruction or a simple count 1, 2, ... if alphanumeric class names have been used. The format of the TRANSF instruction is

\[
\text{TRANSF} \ u \ u_1 \ c \\
M: \ m \\
\text{IF: } \langle \text{condition} \rangle
\]

where \( u \) = the new variable, \( u_1 \) = the variable to be classified, \( c \) = the name of a classification rule defined earlier by a CLASS-instruction, \( m \) = the value to be given if the value of \( u_1 \) is outside the classification intervals.

The TABLE-instruction is used to tabulate frequency counts, percentages, mean values and standard deviations. The instruction is designed for construction of one-way and two-way tables. A TABLE-instruction performs tasks with the same column variable. Tables in more dimensions are programmed applying conditional TABLE-instructions. The tables should be given names for later reference. The tables may be used in analysis instructions. The CHI2-instruction can be used to compute a contingency test for a frequency table. The VARIAN-instruction is able to perform a one-way or two-way analysis of variance using mean value and frequency count tables. The structure of the TABLE-instruction is as follows:
TABLES | (column variable $u_1$) | (classification rule $c$) | (name of moment table) | (weight variable) | (output specification) | (output selection parameters) | (condition)

- $T$: (variable to be tabulated)
- $M$: (output selection parameters)
- $F$: (condition)

Analysis instructions

Estimation of mean values, standard deviations and correlation coefficients is performed using MEAN-, STDDEV- and CORREL-instructions in the following format:

$$(\text{operator}) u_1, \ldots, u_p$$

IF: (condition)

- $N$: (name of moment table)
- $W$: (weight variable)
- $M$: (output specification)
- $T$: (output selection parameters)

where $u_1, \ldots, u_p$ are variables. The sums of squares and sums of products are saved as the moment table, which should be named for later reference.

These moments may be used in an analysis instruction, REGRAN or TTEST.

The MEAN-instruction computes mean values only. STDDEV-instruction estimates both mean values and standard deviations. CORREL-instruction computes, besides mean values and standard deviations, the product moment correlations of the variables $u_1, \ldots, u_p$. In addition to other output options, the correlation matrix with mean values and standard deviations can be punched in an output form which conforms to the input requirements of standard multivariate analysis programs.

The percentage points of empirical distributions can be examined using FRACt-instructions. The estimation of the percentage points is performed using the marginal distribution of a frequency table. The variable subject to investigation appears as a row variable in this table. The variable reference is hence performed in the format of the FRACt-instruction:

$$\text{FRACt@ (name of a table)}$$

where the non-negative integer percentage points selected out of a value which exceeds $i$ per cent results in

$$p_i, q, q_12, \ldots$$

The REGRAN-instruction is used to observations using the method is not designed to operate directly in a matrix to get the necessary input from the experience that slightly different sets of variables. The format is

$$\text{REGRAN@ (name of correl) } x_1, \ldots, x_r$$

In the same way as the earlier CORREL-instruction, the correlation matrix is punched in an output form which conforms to the input requirements of standard multivariate analysis programs.

The specification of whether the analysis is for one way or two way form, as well as the table is referred to by a reference to the table. The variance appears as a T-parameter.

The classifications used in the analysis are gated using the analysis of variance.
The general format of the FRAC1 instruction is:

\[
\text{FRAC1} \langle \text{name of a table} \rangle \ q \ r \ s,
\]

where the non-negative integers q, r, s give the selection rules for percentage points selected out of \( P_0, P_1, \ldots, P_g \); \( P_g \) is the variable value which exceeds i percent of observed values. The instruction gives as results \( P_0, P_{q+r}, P_{q+2r}, \ldots, P_s \).

The REGRAN-instruction fits a linear regression model

\[
y = a_0 + a_1 x_1 + \ldots + a_r x_r
\]

to observations using the method of least squares. This analysis instruction is not designed to operate directly on the data. It needs a correlation matrix to get the necessary information. This arrangement has arisen from the experience that slightly different models are often estimated from the same set of variables. The format of the REGRAN-instruction is

\[
\text{REGRAN} \langle \text{name of correlation matrix} \rangle
\]

\[
y \ x_1 \ldots \ x_r
\]

In the same way as the use of the REGRAN-instruction is based on an earlier CORREL-instruction, the VARAN-instruction uses a TABLE-instruction. The format of this instruction is simply

\[
\text{VARAN} \langle \text{name of the table} \rangle
\]

The specification of whether the analysis of variance is performed in one-way or two-way form, as well as the variable in question, appears implicitly by a reference to the table. The variable subject to the analysis of variance appears as a \( T \)-parameter in the corresponding TABLE-instruction. The classifications used in the tabulation specify the categories investigated using the analysis of variance, as well as whether one-way or two-way
analysis is required. There is a problem in two-way analysis of variance when observation vectors fill the category table in an uneven manner. In SURVO language a heuristic method is used as an approximate solution in that case.

Any frequency table can be analysed for independence of its tabulating variables using the $\chi^2$ - test. This happens applying a CHI2-instruction in the format

CHI2@<name of frequency table>.

The mean values in different groups are tested for equality using the TTEST-instruction. The sums and sums of squares needed for the computations are provided by earlier STDEV or CORREL instructions. This information must have been given a reference name as a moment table. The format of the TTEST-instruction is

either TTEST@<moment table 1> <moment table 2>
or TTEST@<moment table 1> <variable 1>
<moment table 2> <variable 2>.

In the former case it is required that the variables to be compared appear in the same order in the moment tables.
An example of SURVO 66 programming

In order to illustrate SURVO programming we consider a recent statistical research by Dr. Knight on computer characteristics [5]. In this interesting paper the author investigates the functional dependence of computer power and its rental cost. This particular data has been chosen because we felt that most computer people are familiar with the concepts of this research.

The material which Dr. Knight has treated statistically contains 92 data vectors derived from production models of electronic digital computers. The attributes he has measured of each computer are: date introduced, scientific power in operations per second, commercial power in operations per second and inverse of computing cost in seconds of computing per dollar. The data matrix in [5] is of the following form:

<table>
<thead>
<tr>
<th>Date introduced</th>
<th>Scientific</th>
<th>Commercial</th>
<th>Inverse unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month, Year</td>
<td>power(op/sec)</td>
<td>power(op/sec)</td>
<td>cost (sec/$)</td>
</tr>
<tr>
<td>4, 83</td>
<td>21420</td>
<td>9079</td>
<td>44.54</td>
</tr>
<tr>
<td>7, 83</td>
<td>67680</td>
<td>23420</td>
<td>23.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 67</td>
<td>3127256</td>
<td>2755760</td>
<td>15.59</td>
</tr>
<tr>
<td>9, 67</td>
<td>1086342</td>
<td>1021365</td>
<td>29.69</td>
</tr>
</tbody>
</table>

Computer no 303 is omitted here because of an obvious printing error.

We investigate the interdependence of the scientific power $P$ of the computer and the computing cost $C$ using the technological age $T$ of the computer as an external variable to be compensated. The units of measurement for $P$, $C$ and $T$ are 1000 op/sec, $$/hour, and month respectively. We will fit a logarithmic regression model

$$\ln P = a_0 + a_1 \ln C + a_2 T$$

to the data. We also cross-tabulate the average power of computers in three
cost categories for each year 1963, ..., 67 of computer announcement.

As data validity checks we require that the variables "month" and "year" should not be outside the intervals 1-12 and 63-67 respectively.

A reproduction of results is included. We can see that Greshch's famous law \( P = kC^2 \) seems to fit well to Dr. Knight's data.

SURVO program:

```
EVOLVING COMPUTER PERFORMANCE 1963-1967, DATAMATION, JAN. 1968

MPR:
CALL X1 MONTH, X2 YEAR
DEFN X5 S:1
   # MONTH L:1 U:12
   # YEAR L:63 U:67
DIVS SPEED X3 1000 S:1
DIVS COST 1000 X5 S:3
SUM Y1 =* YEAR
MULT Y2 = - Y1
SUM AGE Y2 MONTH
LOGS LSPEED X3 S:3
   # LCOST COST S:3
CLASS COSTCL
CHEAP 0 30,000
MIDR 30,001 90,000
H: SHORT S:3
TABLE YEAR:
   DEVEL COST CLSTCL T:SPEED
CORREL LSPEED LCOST AGE M: CORR
REGRAN CORR LSPEED LCOST AGE
END
```

Results of the SURVO program:

```
CLASSIFICATION: COSTCL
CLASS LIMITS
CHEAP 0.000000 30.000000
MIDR 30.00001 90.000000
EXPNS 90.00001 500.000000

VARIABLES
NO. NAME SCALE
  1 MONTH 0
  2 YEAR 0
  3 X3 0
  4 X4 0
  5 X5 1
  6 SPEED 1
  7 COST 3
  8 Y1 0
  9 Y2 0
 10 AGE 0
 11 LSPEED 3
 12 LCOST 3

EVOLVING COMPUTER PERFORMANCE
N= 91

TABLE: DEVEL
COLUMN VARIABLE: YEAR
ROW VARIABLE: COST CL
FREQUENCIES
   63  64
CHEAP  6  4
MIDR  7  11
EXPNS  6  5
TOTAL 19 21
```

29
Results of the SURVO program:

EVOLVING COMPUTER PERFORMANCE 1963-1967, DATAMATION, JAN. 1968

CLASSIFICATION: COSTCL
CLASS LIMITS
CHEAP 0.0000 30.0000
MODER 30.0010 90.0000
EXPNS 90.0010 500.0000

VARIABLES
NO. NAME SCALE
1 MONTH 0
2 YEAR 0
3 X3 0
4 X4 0
5 X5 1
6 SPEED 1
7 COST 3
8 Y1 0
9 Y2 0
10 AGE 0
11 LSPED 3
12 LCST 3

EVOLVING COMPUTER PERFORMANCE 1963-1967, DATAMATION, JAN. 1968
N = 91

TABLE: DEVEL
COLUMN VARIABLE: YEAR
ROW VARIABLE: COST CLASSIFICATION; COSTCL

FREQUENCIES
<table>
<thead>
<tr>
<th>YEAR</th>
<th>63</th>
<th>64</th>
<th>65</th>
<th>66</th>
<th>67</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHEAP 6</td>
<td>4</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>MODER 7</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>EXPNS 6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>TOTAL 19</td>
<td>23</td>
<td>25</td>
<td>18</td>
<td>8</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

30
Our experiences so far indicate that the language seems to be feasible for a large computer. We have been able to specify their statistical data without any expert help.

We have observed a rise in statistical applications. For example, when the researcher is analyzing data, it is often necessary to specify the statistical equations without any expert help.

There also exists a steady demand for statistical techniques in the field of social sciences. This rise in the number of applications is leading to the increased use of special programming languages.

In system design we have observed that there is a need for a system that can handle both statistical and non-statistical applications. Therefore, the syntax of the language is designed in such a way that it can be used not only for statistical applications but also for other types of applications.

In conclusion, our experiences so far indicate that the language is feasible for a large computer.
Experiences and conclusions

Our experiences so far indicate that the idea of a statistical language seems to be feasible. We shall proceed to implement the system for a larger computer. We have also found that researchers have been able to specify their statistical data processing jobs in the SURVO language without any expert help.

We have observed a remarkable increase in the use of computers in statistical applications. Part of this increase is due to the ease of use when the researcher is able to specify himself his information processing needs. Part of the increase comes from new applications where the prohibiting cost of special programming is now to a large extent removed.

There also exist some negative aspects which we have found in our system. The method of scaling we have used in the system may sometimes cause unpleasant pitfalls. When transferring the system to a faster computer we will introduce more floating point computing to remedy this drawback. There also exists a steady demand from the users’ side for more sophisticated statistical techniques in the SURVO system. A computer with larger memory capacity is needed to satisfy this demand. A final goal is an integrated system for all statistical manipulation needed in usual statistical research.

In system design we have aimed at simplicity where possible. Therefore the syntax of the SURVO language is chosen more in favour of simple compiling than of syntactical beauty. There have been, however, enough reasons to promote this research project as an interdisciplinary effort in co-operation with computer scientists, statisticians and users of computing services.
Acknowledgements

We are grateful to Oy Nokia Ab, Electronics Division and the University of Tampere for the support they have given to this research. In the implementation phase several persons have participated in the project. We want especially to mention the valuable contributions of Leena Lankinen, Tatu Kalin, Matti Ylihen as well as those of Pentti Kanerva and Kari Kärkkäinen.

Literature


