IBM

Technical Newsletter No. 13

APPLIED SCIENCE

IBM

APPLIED SCIENCE

TECHNICAL NEWSLETTER NO. 13

APRIL 1957

We wish to extend thanks to the authors of these papers for their cooperation in contributing to this interchange of technical information.

CONTENTS

A PUNCHED CARD METHOD FOR SUCCESSIVE INTERVALS SCALING[a]

Samuel J. Messick and Ledyard R. Tucker
Princeton University and Educational Testing Service

and

Harry W. Garrison
Educational Testing Service

INTRODUCTION

The scaling method of successive intervals has appeared in several forms (e.g., 1, 3, 5) with variations in the computational labor required depending upon the simplifying assumptions made. Generally, the more rigorous the solution, the more laborious the computations (see reference 4); but, in any event, as the number of stimuli to be scaled increases, the amount of computation required by any suggested procedure soon becomes prohibitive. Recently, however, an iterative least squares solution to successive intervals[2] was developed, for which punched card procedures were appropriate; and thus a feasible computational routine for scaling large numbers of stimuli was made available without sacrificing the rigor of a least squares approach.

A punched card procedure for a general weighted least squares solution to successive intervals is described below. Use of zero weights permits application of this procedure to cases of incomplete data. The procedure outlined suggests use of an IBM 101 Electronic Statistical Machine for obtaining frequency counts. Conversion of frequencies to normal curve deviates and

[a]This research was jointly supported in part by Princeton University, the Office of Naval Research under contract N6onr-270-20, and the National Science Foundation under grant NSF G-642, and in part by the Educational Testing Service.

assignment of weights is accomplished on a gang punch. The main procedure is an iterative process with repetitive cycles of computation using a 602A Calculating Punch and an accounting machine.

THE GENERAL LEAST SQUARES SOLUTION TO SUCCESSIVE INTERVALS

The experimental procedure for the method of successive intervals requires n stimuli to be sorted N times into (k + 1) categories on some attribute continuum. From this sorting procedure the number of times, f_{ig}, that the ith stimulus was placed in the gth category can be readily obtained. The frequencies in each row of the n x (k + 1) table generated by these data are then cumulated so that each entry, F_{ig}, now represents the number of times the ith stimulus appeared below the gth category boundary, t_g. These cumulated frequencies are converted into proportions, p_{ig}, and then to normal deviate values, z_{ig}. At this point, some set of weights, w_{ig}, can be applied to the normal deviates to take account of differences in the reliability of the various proportions.

The general least squares solution to successive intervals [2] provides formulae for the n scale values, m_i, the n discriminal dispersions, s_i, and the k category boundaries, t_g, as follows:

$$s_{i\alpha} = a_i \sum_g^k w_{ig} t_{g\alpha} z_{ig} - b_i \sum_g^k w_{ig} t_{g\alpha}, \qquad (1)$$

$$m_{i\alpha} = c_i \sum_g^k w_{ig} t_{g\alpha} - b_i \sum_g^k w_{ig} t_{g\alpha} z_{ig}, \qquad (2)$$

where

$$a_i = \frac{\sum_g^k w_{ig}}{(\sum_g^k w_{ig} z_{ig}^2)(\sum_g^k w_{ig}) - (\sum_g^k w_{ig} z_{ig})^2}, \qquad (3)$$

$$b_i = \frac{\sum_g^k w_{ig} z_{ig}}{(\sum_g^k w_{ig} z_{ig}^2)(\sum_g^k w_{ig}) - (\sum_g^k w_{ig} z_{ig})^2}, \qquad (4)$$

$$c_i = \frac{\sum_g^k w_{ig} z_{ig}^2}{(\sum_g^k w_{ig} z_{ig}^2)(\sum_g^k w_{ig}) - (\sum_g^k w_{ig} z_{ig})^2}, \qquad (5)$$

$$t_{g(\alpha + 1)} = \frac{v_{g(\alpha + 1)}}{\sqrt{\frac{1}{W} \sum_g^k v_{g(\alpha + 1)}^2 \sum_i^n w_{ig}}}, \qquad (6)$$

where $$v_{g(\alpha + 1)} = \frac{\sum_i^n w_{ig} m_{i\alpha} + \sum_i^n w_{ig} z_{ig} s_{i\alpha}}{\sum_i^n w_{ig}} \qquad (7)$$

and $$W = \sum_g^k \sum_i^n w_{ig} .$$

The subscript α was introduced into the above equations to indicate the various cycles of approximation in an iterative procedure. Thus, if some tentative first estimates, t_{g1}, of the category boundaries were available to start the iteration, equation (1) could be solved to obtain first estimates, s_{i1}, of the discriminal dispersions; equation (2) could be used to find first estimates, m_{i1}, of the scale values; and equations (6) and (7) could be solved for second estimates, t_{g2}, of the category boundaries, etc. The procedure could be iterated until two successive t-estimates were as similar as desired, i.e., until $\left[t_{g(\alpha + 1)} - t_g\right]$ was negligible. It should be noted that the coefficients a_i, b_i, and c_i are written solely in terms of the data and do not involve iteration subscripts α ; they need to be computed only once for the entire iterative procedure.

A suitable first estimate of t_g with which to start the iterative procedure must now be determined. In order to be consistent with the above equations, an origin and unit for the t-scale should be defined so that

$$\sum_g^k t_g \sum_i^n w_{ig} = 0 \qquad (8)$$

and

$$\sum_g^k t_g^2 \sum_i^n w_{ig} = W. \qquad (9)$$

This definition is completely arbitrary, the successive intervals scale being determined only within a linear transformation. A t-scale meeting such requirements may be obtained as follows. If v_{gl} is a set of k numbers to be used as possible estimates of t_g , then

$$t_{gl} = \frac{v_{gl} - \bar{v}_1}{\sqrt{\frac{1}{W} \sum_g^k (v_{gl} - \bar{v}_1)^2 \sum_i^n w_{ig}}} , \qquad (10)$$

where

$$\bar{v}_1 = \frac{1}{W} \sum_g^k v_{gl} \sum_i^n w_{ig}.$$

A set of equally spaced numbers, such as the integers from 1 to k converted according to equation (10), would be a convenient first estimate of the category boundaries. The rate of convergence may possibly be increased by doubling or tripling the difference between successive t-estimates, i.e., instead of using $t_{g(\alpha + 1)}$ in the $(\alpha + 1)$ iterative cycle use $(t'_{g(\alpha + 1)} = t_g + 2(t_{g(\alpha + 1)} - t_g)$.

The weights, w_{ig} , appearing in the above equations may be chosen in any fashion as long as $w_{ig} = 0$ and $w_{ig} z_{ig} = 0$ when $p = 0$ or $p = 1$. It would seem reasonable, however, to select these weights so as to take account of differences in the reliability of various proportions. See reference 2 for a more detailed discussion of the choice of weights.

TRANSFORMATIONS TO SIMPLIFY THE PUNCHED CARD PROCEDURE

Before presenting the punched card routine, the introduction of some linear transformations will greatly simplify the procedure. Because of the conversion of equation (10), the $t_{g\alpha}$ values in the above solution are both positive and negative, as are the normal deviate values, z_{ig}. Since it would be advantageous to have all these quantities converted to positive values, the above solution was modified according to the following functions:

$$Z_{ig} = e z_{ig} + f \qquad (11)$$

$$T_{g\alpha} = e t_{g\alpha} + f, \qquad (12)$$

where e and f are transformation constants chosen to make z_{ig} and $t_{g\alpha}$ all positive and to expand the range.

The introduction of these conversions does not affect the formula for estimates of the discriminal dispersions at all. Its form is merely changed to

$$s_{i\alpha} = A_i \sum_g^k w_{ig} T_{g\alpha} Z_{ig} - B_i \sum_g^k w_{ig} T_{g\alpha}, \qquad (13)$$

where A_i and B_i equal a_i and b_i, respectively, with Z_{ig} substituted for z_{ig}.

However, the transformations do produce some changes in the formula for estimating scale values. The $m_{i\alpha}$ values may now be obtained in terms of the transformed scores as follows:

$$m_{i\alpha} = \frac{1}{e}(P_{i\alpha} + fs_{i\alpha} - f), \qquad (14)$$

where $$P_{i\alpha} = C_i \sum_g^k w_{ig} T_{g\alpha} - B_i \sum_g^k w_{ig} T_{g\alpha} Z_{ig} \qquad (15)$$

Again, B_i and C_i equal b_i and c_i , respectively, with Z_{ig} substituted for z_{ig}.

In summary, if the transformations given in equations (11) and (12) are used, equation (15) may be applied to obtain some values, $P_{i\alpha}$, which can be converted by equation (14) into the scale values, m_i . New estimates of the $t_{g\alpha}$ scale can then be obtained as follows:

$$V_{g(\alpha+1)} = e\,v_{g(\alpha+1)} + f = \frac{\sum_i^n w_{ig} P_{i\alpha} + \sum_i^n w_{ig} s_{i\alpha} Z_{ig}}{\sum_i^n w_{ig}} \quad (16)$$

$$t_{g(\alpha+1)} = \frac{V_{g(\alpha+1)} - f}{\sqrt{\frac{1}{W} \sum_g^k \left[V_{g(\alpha+1)} - f \right]^2 \sum_i^n w_{ig}}} \quad (17)$$

THE PUNCHED CARD PROCEDURE

1. The starting point for successive intervals analysis is an n × k table of the number of times, F_{ig} , that the ith stimulus was placed below the gth category boundary, t_g . Since the data are rarely available in this form, individual scores must be tallied. If a card is made up for each of N individuals, with n columns representing n stimuli, the category (1 to k + 1) into which each stimulus was placed by an individual can be punched in the columns appropriate for that stimulus. It may be necessary to use more than one column to designate each stimulus; and if there are too many stimuli for one card per individual, two or more cards might be punched for each person. Then the cumulated frequencies, F_{ig}, with which each stimulus was placed below each category boundary may be directly tallied on the IBM 101 Electronic Statistical Machine. These operations may also be performed on an accounting machine.

The cards used for tallying the initial scores will not be needed in the rest of the analysis. The cumulated frequencies, F_{ig}, are placed in detail cards - there being one such card for each entry in a stimulus-by-category table. There are actually $k + 1$ categories; however, since F_{ig} would always equal N for the last category, it need not be punched. Also, there will usually be missing entries in the F_{ig} table, so the total number of detail cards will generally be less than $n \times k$. The detail cards will be operated upon from this point on. In addition to F_{ig}, each card should also contain punches for stimulus and category designations.

2. A deck of conversion master cards must now be set up which will convert cumulated frequencies, F_{ig}, into corresponding transformed normal deviates, Z_{ig}. Accordingly, each master card should contain F_{ig} and the corresponding Z_{ig} value. It will be advantageous for later operations if the conversion cards also contain for each F_{ig} the corresponding weight, w_{ig}, and the products $w_{ig}Z_{ig}$ and $w_{ig}Z_{ig}^2$. The conversion deck should also be set up so that cumulated frequencies for which the corresponding w_{ig} is zero would be converted into X punches in the Z_{ig} field. The X punch is used to indicate a conversion to zero, so that Z values with zero weights may be easily sorted out at a later stage of the procedure. This single conversion deck, then, includes the weighting system, the conversion to normal deviates, and the linear transformation of equation (11).
3. The detail cards are now sorted into ascending order of F_{ig}, the conversion master cards are merged in front controlling on F_{ig}, and the quantities Z_{ig}, w_{ig}, $w_{ig}Z_{ig}$, and $w_{ig}Z_{ig}^2$ are transferred from the master cards by

gang punching. Each detail card now contains entries of F_{ig}, Z_{ig} (where the Z_{ig} value may be an X punch), w_{ig}, $w_{ig}Z_{ig}$, $w_{ig}Z^2_{ig}$, and punches identifying stimulus, i, and category, g.

4. Some of the detail cards will not be used in the analysis, since they contain little or no information. At this point, all cards with an X punched in the Z_{ig} field can be removed. The X punch indicates that the information in the card was based upon a frequency so unreliable that it had been weighted zero. In case there is only one Z_{ig} for any stimulus i, the card for this Z_{ig} is to be removed and the stimulus dropped from the study.
5. The following steps may now be performed on a desk calculator:
   (a) Using the integers 1 to k as v_{gl}, solve equation (10) for initial estimates of the category boundaries, t_{gl}. If the number of stimuli is so large that it is difficult to find the k values of $\sum_i^n w_{ig}$ by hand, the cards may be sorted by category and $\sum_i^n w_{ig}$ obtained for each g on an accounting machine.

   (b) Convert the t_{gl} values into positive scores, T_{gl}, by equation (12).
6. Gang punch T_{gl} values into detail cards with the corresponding category designation, g.
7. Sort the cards in order of stimuli and insert trailer cards.
8. Punching a trailer card for each stimulus, run off the sums $\sum_g^k w_{ig}$, $\sum_g^k w_{ig}Z_{ig}$, and $\sum_g^k w_{ig}Z^2_{ig}$ on an accounting machine.
9. Using a desk calculator, compute the coefficients A_i, B_i, and C_i, according to equations (3), (4), and (5), respectively, with Z_{ig} substituted for z_{ig}. For large numbers of stimuli these values may be more efficiently computed on the 602A Calculating Punch.

10. With the cards in order of stimuli and punching a trailer card for each i, compute on the 602A the values $\sum_g^k w_{ig} T_{g1}$ and $\sum_g^k w_{ig} Z_{ig} T_{g1}$.

11. Using a desk calculator, solve equations (13) and (15) for the n values of s_{i1} and P_{i1}, respectively. Again the 602A is probably more efficient for solving these equations when the number of stimuli involved is very large.

12. Punch into each detail card the corresponding s_{i1} and P_{i1} value.

13. Sort the cards in order of category, and for each g obtain $\sum_i^n w_{ig}$, $\sum_i^n w_{ig} Z_{ig} s_{i1}$, and $\sum_i^n w_{ig} P_{i1}$ on the 602 A.

14. Again at this point, hand computation is probably better for solving equations (16) and (17) for a new set of t_{g2} values.

15. Convert t_{g2} to positive scores according to equation (12) and gang punch the T_{g2} values into detail cards with corresponding category designations.

16. Beginning with step 10, repeat the above computations to obtain new estimates s_{i2}, P_{i2}, and t_{g3}. Steps 10 to 15 can then be iterated, replacing the subscript 1 with an α appropriate to the iterative cycle being performed, until two successive t-estimates are as similar as desired.

17. When convergence has been obtained, the final scale values, m_i, can be computed from equation (14).

REFERENCES

1. F. Attneave, "A Method of Graded Dichotomies for the Scaling of Judgments," Psychological Review, LVI (1949), 334-340.
2. G. W. Diederich, S. J. Messick, and L. R. Tucker, "A General Least Squares Solution for Successive Intervals," Educational Testing Service, Research Bulletin 55-24.
3. W. R. Garner, and H. W. Hake, "The Amount of Information in Absolute Judgments," Psychological Review, LVIII (1951), 446-459.
4. H. Gulliksen, "A Least Squares Solution to Successive Intervals Assuming Unequal Standard Deviations," Psychometrika, IXX (1954), 117-139.
5. M. Saffir, "A Comparative Study of Scales Constructed by Three Psychophysical Methods," Psychometrika, II (1937), 179-198.

A SIMPLIFIED METHOD FOR THE COMPUTATION OF BISERIAL CORRELATION COEFFICIENTS ON THE 604 ELECTRONIC CALCULATING PUNCH[a]

K. Warner Schaie[b], Allan Katcher, S. Frank Miyamoto and Laura I. Crowell
University of Washington

A number of schemes for obtaining biserial correlation coefficients by means of punched card computing methods have been reported in the psychological and computing literatures. All of these procedures (a representative example is the one proposed by Johnson[4]) yield only the components necessary for computing the final coefficient and require additional manual computation on a desk calculator.

The greatest difficulty in programming a straightforward procedure lies in the fact that it is necessary to obtain values from a table of the normal probability curve in the course of the computation. While this problem can be readily taken care of on the IBM 650 or 704, the social scientist usually has access only to lower order equipment, which for his purposes is generally less expensive.

In this article we will describe a one-board procedure for the IBM 604 Electronic Calculating Punch, which will automatically perform the required table look-up and compute the final r_{bis}. We selected the 604 because this instrument is rapidly becoming standard equipment for most basic IBM installations and is thus readily available. In presenting our procedure, familiarity with the 604 and with general IBM procedures is assumed, and the terminology used agrees with the instruction manual for the 604[3].

[a] The authors gratefully acknowledge the research support provided by a grant from the Agnes H. Anderson fund of the University of Washington.

[b] Now with the Department of Psychiatry and Neurology, Washington University School of Medicine, St. Louis, Missouri.

Given the continuous and dichotomized scores, the mean, and the standard deviation for the total group on the continuous variable, the proposed method will summary punch, in one machine run, one biserial correlation coefficient at a time. No prior information is required on the magnitute of p or q, and it will be shown that categorical data can be used for purposes of dichotomization without having to be repunched. This method is thus also useful in cases where the investigator originally intended to compute Pearsonian r's but, after inspection of his data, concluded that biserial correlations were more appropriate.

The method to be described here was developed in connection with a research project focusing on self-concepts of communicative skills[1]. Two specifically constructed questionnaires were administered to several hundred students who participated in three different studies. As one phase of the study, we were interested in examining relationships between biographical information and questionnaire scores. The computational example given in the final part of this paper is derived from this study.

COMPUTATIONAL FORMULA

For our purposes, the most economical formula is one suggested by Garrett[2], since it requires a minimum number of constants to be carried during the calculation. We write the formula for the biserial correlation coefficient as follows:

$$r_{bis} = \frac{M_p - M_t}{\sigma t} \cdot \frac{p}{y} ,$$

where M_p = the mean of the continuous variable for the larger proportion of scores, regardless of whether this group represents 0 or 1 scores.

M_t = the mean of the total group on the continuous variable.

σ_t = the standard deviation of the total group on the continuous variable.

p = the proportion of the group having the largest number of entries.

y = the ordinate of the normal curve corresponding to p.

Of all the required components, a priori knowledge is needed only of M_t and σ_t, since all other components, with the exception of y, will be obtained during the computation. The values of y, for a sufficient range of p's, were obtained from a conventional table of the normal curve. These values were then punched on a special deck of cards, described below. Values for M_t and σ_t may be computed by a number of machine routines. We used a convenient procedure to obtain these values on the 604, which Lunneborg, Wright, and Ax recently reported.[5]

PREPARATION OF PUNCHED CARDS

Any set of detail cards containing scores for the continuous and dichotomous variables may be utilized, but a standard format will be given to correspond to the wiring diagrams shown in this paper.

Columns 1 to 4 are assigned the subject identification code. Column 6 receives an X punch, which identifies the card as a detail card and impulses the programs steps required for this card. The continuous scores are punched in columns 8, 9, and 10. In our design, three-digit scores are used for the continuous variable; two-digit scores would have to be punched in the form xx0 to conform to this layout. A zero or one is punched for each dichotomized variable, one column per variable, beginning with column 37. Where scores may have more than two categories, numerical scores are punched; and after a decision has been made as to where to dichotomize, allowance can be made in the punch panel wiring.

Four types of function cards are required. The first is a card which precedes the deck during calculation and clears the storage units. This card receives an X punch in column 78.

The second function card reads in the required constants and impulses part of the computation. This card receives the reciprocal of the total number of cases (1/N) in columns 1 to 5. The mean for the total number of cases M_t is punched in columns 8 to 11, and the standard deviation for the total group σ_t is assigned to columns 12 to 15. Both mean and standard deviation are taken to two decimal places, while the reciprocal is taken to five decimals. An X is placed in column 79 to impulse the program steps wired to occur when this card passes through the calculator.

Next we prepare the card which receives the final result. It requires a 12 punch in column 77. (A 12 rather than an X punch must be used to permit immediate transfer to avoid normal punch suppression. See diagrams.) This card is also prepunched with the variable identification number to ensure easier recording of the results.

Finally we prepare the table look-up deck of 40 cards, each of which receives an X punch in column 76. Each of the X76 cards further receives a numerical identification in columns 1 and 2. The values for p are punched in columns 21 and 22, and the corresponding values for y (staggered one card to permit delayed pickup) are punched to three decimals in columns 23 to 25. Table 1 gives the required values for punching this deck, which covers splits from 50-50 to 90-10. This should be adequate for most empirically derived distributions.

COMPUTATIONAL PROCEDURE

In developing the present procedure, advantage was taken of the limited decision-making capacity of the 604 and also of the possible use of this machine as a miniature card-programmed calculator. Table 2 lists all the steps required in the computation. These will be summarized in the following paragraphs.

The cards are arranged in the following sequence: X78, detail cards (X6), X79, table look-up deck (X76), 12-77. When the X78 card is read, the storage units are cleared; but no calculation is otherwise permitted to occur on this card.

As the first X6 (detail) card is read, a 1 is emitted into the counter. The dichotomized score is read through a digit selector and, depending upon whether the score is zero or unity, the 1 in the counter is assigned to different storage compartments. Depending upon this information also, the numerical score is read into different storage compartments. Both the count as well as

TABLE 1

Values of p = y Punched for the Table-Look-Up Operation on the 604[a]

Column	1-2	21-22	23-25
	1	52	399
	2	53	398
	3	54	398
	4	55	397
	5	56	396
	6	57	394
	7	58	393
	8	59	391
	9	60	389
	10	61	386
	11	62	384
	12	63	381
	13	64	378
	14	65	374
	15	66	370
	16	67	366
	17	68	362
	18	69	358
	19	70	353
	20	71	348
	21	72	342
	22	73	337
	23	74	331
	24	75	324
	25	76	318
	26	77	311
	27	78	304
	28	79	296
	29	80	288
	30	81	280
	31	82	271
	32	83	262
	33	84	253
	34	85	243
	35	86	233
	36	87	223
	37	88	212
	38	89	200
	39	90	188
	40	91	176

[a]Staggered to permit delayed pickup

TABLE 2

Program Chart for the Biserial Correlation Computing Procedure

Card No.	Prog. Step	Read Out	Read In	Function	Units		Suppression
					Into	Out of	
6	Read		Mult Quot				
79	Read		Mult Quot, FS2, FS4				
76	Read		FS1, GS2				
78	1	Counter	FS1, FS3	Positive			NX78
78	2	Counter	GS1-2, GS3-4	Positive			NX78
6	3		Counter	Emit 1 Positive			NX6
6	4	FS1 or FS3[a]	Counter	Positive			NX6
6	5	& Reset Counter	FS1 or FS3[a]				NX6
6	6	Mult Quot	Counter	Positive			NX6
6	7	GS1-2 or GS3-4[a]	Counter	Positive			NX6
6	8	Counter	GS1-2 or GS3-4[a]				NX6
79	9	FS1	Counter	Positive			NX79
79	10	FS3	Counter	Minus, Balance test for step suppression			NX79
79	11	& Reset Counter					NX79
79	12	FS1		Multiply Positive			NX79 & minus
79	13	FS3		Multiply Positive			NX79 & plus
79	14			1/2 adjust	3		NX79
79	15	FS1	FS3				NX79 & minus
79	16	& Reset Counter	FS1			4	NX79
79	17	GS1-2	Counter	Positive	3		NX79 & minus
79	18	GS3-4	Counter	Positive	3		NX79 & plus
79	19	FS3		Divide			NX79
79	20	& Reset Counter					NX79
79	21	Mult Quot	Counter	Positive			NX79
79	22	FS2	Counter	Negative			NX79
79	23			1/2 Adjust			NX79
79	24	& Reset Counter	GS1-2	Positive		2	NX79
79	25	GS1-2	Counter	Positive	5		NX79
79	26	FS4		Divide			NX79
79	27	& Reset Counter					NX79
79	28	Mult Quot	GS3-4				NX79
79	29	FS1	FS3				NX79
79	30	GS3-4	Counter	Negative			NX79 & minus
79	31	& Reset Counter	GS3-4				NX79 & minus
76	32	FS3	Counter	Positive			NX76
76	33	FS1	Counter	Negative			NX76
76	34			Zero Check			NX76
76	35	GS2	FS4				NX76[b]
76	36	& Reset Counter					NX76
77	37	FS3	Counter	Positive	5		NX77
77	38	FS4		Divide			NX77
77	39	& Reset Counter					NX77
77	40	GS4		Multiply Positive			NX77
77	41			1/2 Adjust	2		NX77
77	42	& Reset Counter	GS3-4			3	NX77
77	43	GS3-4	Counter	Positive			NX77
77	Punch	& Reset Counter					

[a]The dichotomized score read in through a digit selector determines the position of calculator selector 1. If a zero or low-order digit is read, FS1 and GS1-2 are used; if a unity or high-order digit is read, FS3 and GS3-4 are used (see text).

[b]Step 35 is suppressed also whenever a zero check impulse transfers calculator selector 2, through which this program is wired.

the numerical score is then added to the totals accumulating in the correct storage units. This selection is performed by impulsing calculator selector 1 to transfer whenever a zero is read. If categorical scores have been punched in the detail card, it is necessary only to wire the digits which are to be considered zero through bus hubs to the calculator selector pickup to achieve the same result as if a zero had actually been punched in the card.

The X79 reads in the information on M_t, σ_t, and 1/N, as well as impulsing a number of logical and arithmetical steps. The total counts for zero and unity entries are compared. The larger count is then divided by its N to obtain M_p, from which M_t is next subtracted. The remainder is then divided by σ_t and stored. Depending on the outcome of the balance test, an adjustment is made to give this remainder the correct sign.

On each X76 card the p entry stored in the calculator is compared with the one read from the individual X76 card. A zero check impulse is emitted whenever the two do not compare, thus transferring calculator selector 2 to inhibit reading in the y value. At the time when the two amounts do agree, the corresponding y value is permitted to read into calculator storage from the X76 card. Since the zero check impulse is emitted one cycle later, the y entries are staggered by one card on the table look-up deck.

The X77 card permits completion of the calculation. p is divided by y and the result is multiplied by the stored product of the first part of the equation which was computed while the X79 card passed through the calculator.

The result punches on this card in columns 71 to 74, the coefficient being punched to four decimals.

Figure 1 shows the punch panel wiring for this operation. It should be noted that all wiring is permanent except for the wire from the common hub of pilot selector 2. This wire is moved after each run if more than one dichotomous score is punched on the detail cards, as will ordinarily be the case. Figure 2 gives the calculator panel wiring, except for the necessary program suppressions. These are shown separately in figure 3 for greater clarity.

The time required to compute one coefficient is determined by the number of detail cards (N) plus the required 43 function cards. It can be determined by the equation

$$\text{Machine time for one } r_{bis} = \frac{N + 43}{100} \text{ minutes}$$

Because three-digit storage units are used to record the unit count, the upper limit for N is equal to p = 999 or, in general, to approximately 1200 cases, depending upon the extremity of the splits. The method will prove most useful where there are at least several hundred subjects and where there are few continuous and many dichotomous variables.

COMPUTATIONAL EXAMPLE

As already mentioned, the above method was developed for a study of self-concepts of communication skills. As a simple illustration we shall take the scores for six subjects on the communication scale and the dichotomized score on "number of siblings." The communication scale required the subject to

PUNCH PANEL

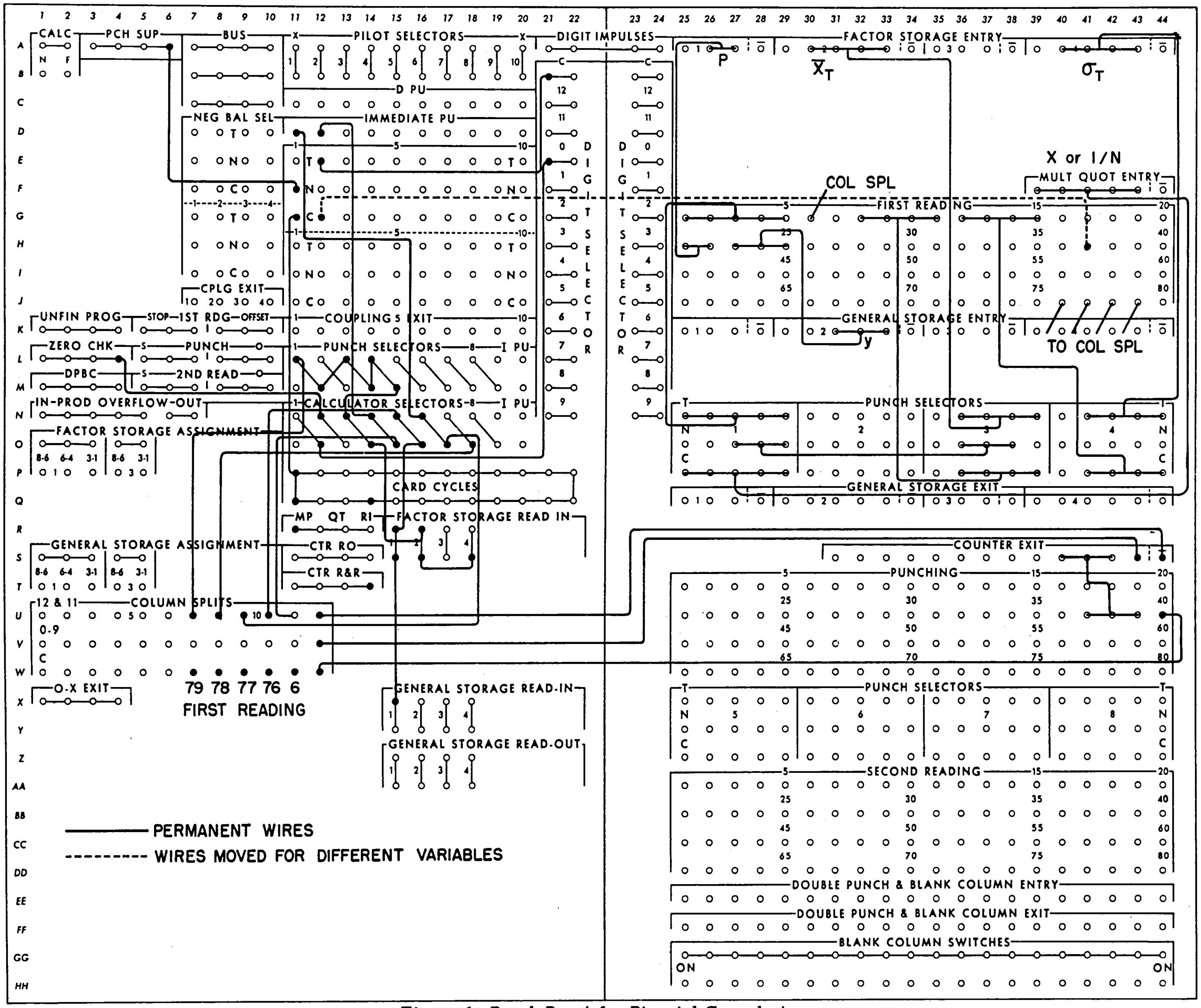
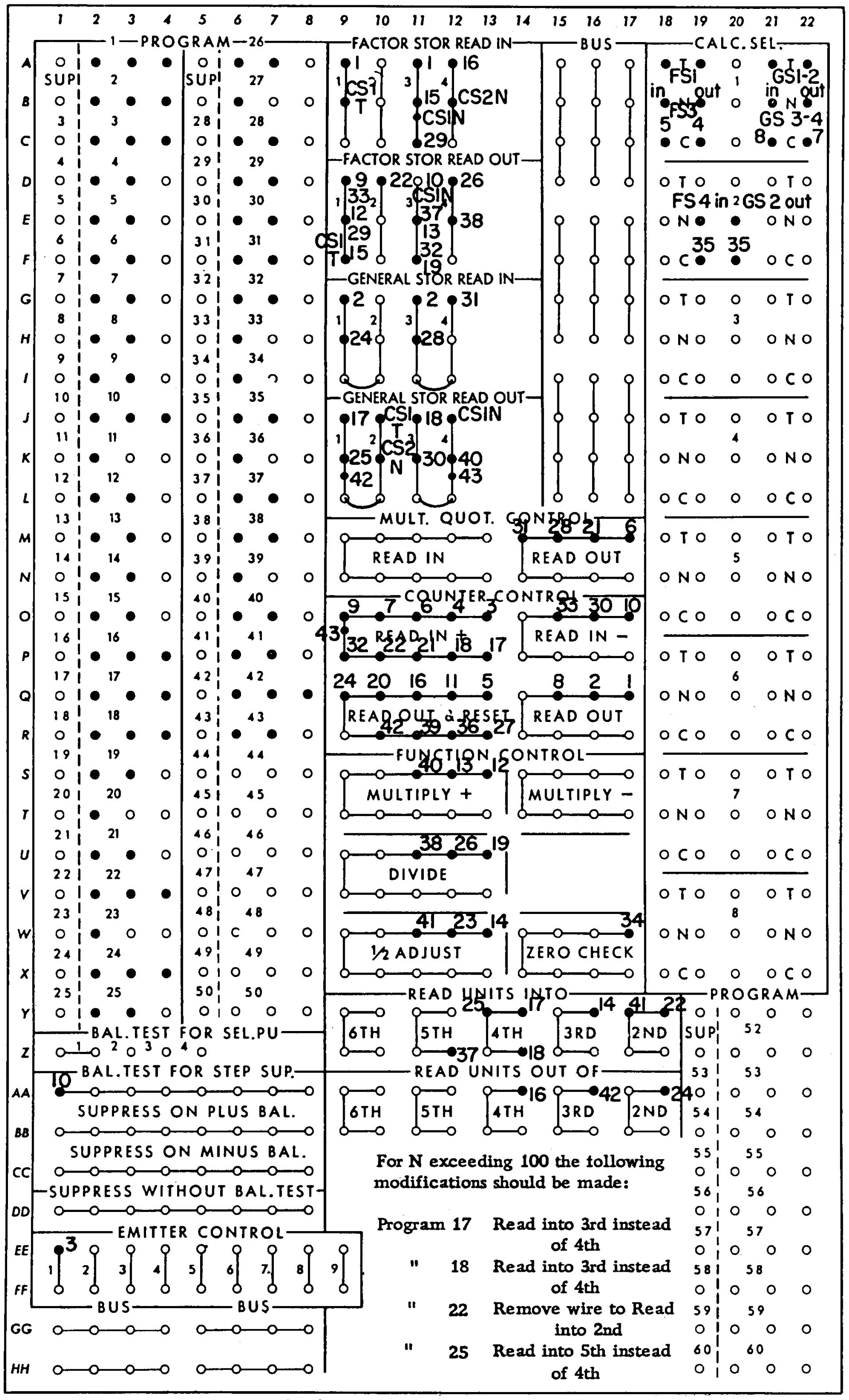


Figure 1. Punch Panel for Biserial Correlation



For N exceeding 100 the following modifications should be made:

Program	17	Read into 3rd instead of 4th
"	18	Read into 3rd instead of 4th
"	22	Remove wire to Read into 2nd
"	25	Read into 5th instead of 4th

Figure 2. Calculator Panel for Biserial Correlation

CALCULATOR PANEL

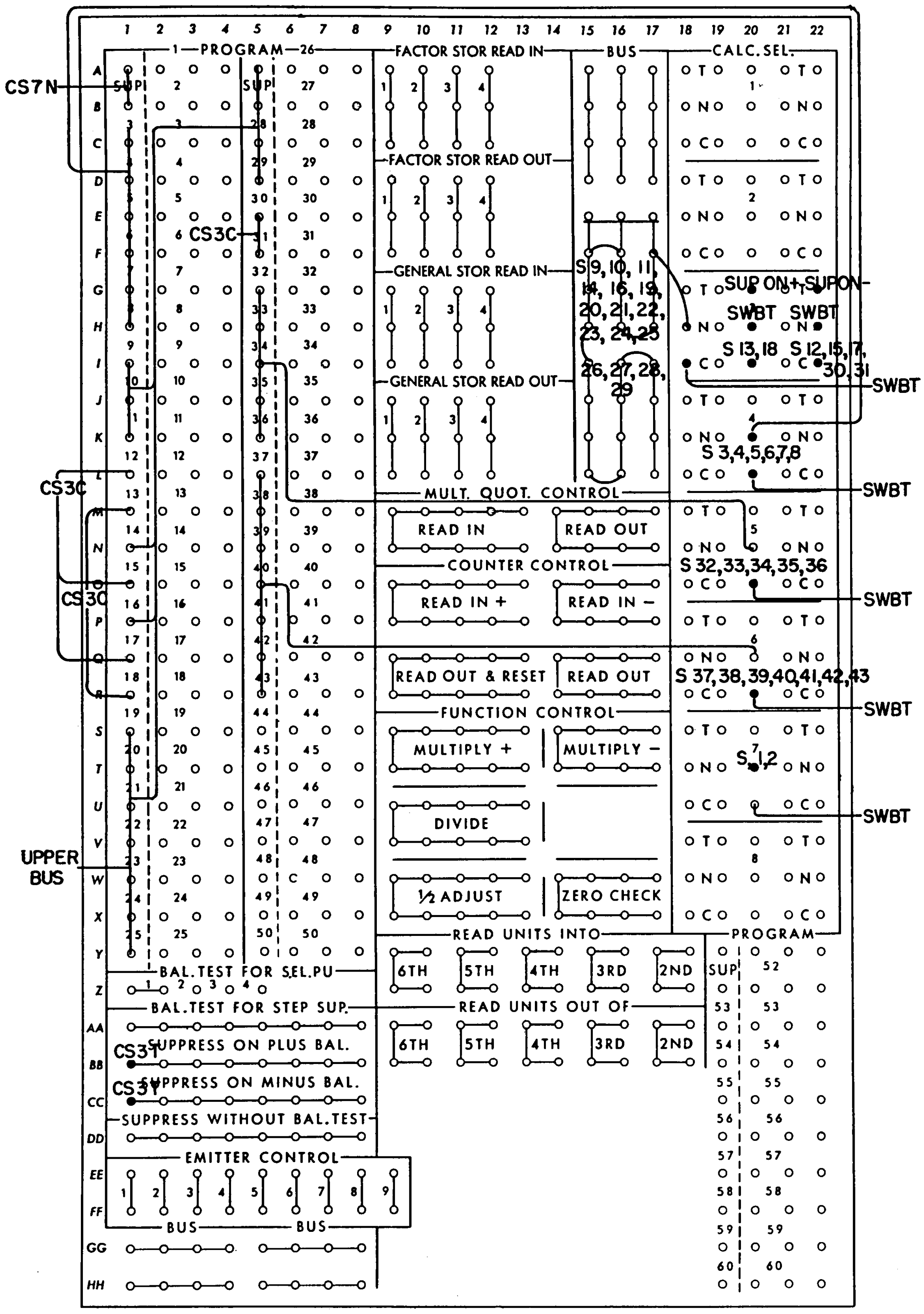


Figure 3. Program Suppression for Biserial Correlation

appraise his performance as a communicator in a wide variety of situations. A scoring scheme was employed so that total scores on the scale reflected the extent to which a subject felt his over-all performance was high or low. Table 3 gives the information for these subjects punched into the detail cards and the X79 card.

TABLE 3

Detail Cards for the Computational Example

	Subject Identif.	Communication Score	No. of Siblings
Column	1-4	8-10	37
	0001	075	1
	0002	085	1
	0003	110	1
	0004	080	1
	0005	090	2
	0006	125	3

$\overline{X} = 94.17$
$\sigma = 17.65$
$1/N = .16667$

The categorical variable (number of siblings) was dichotomized into subjects with one sibling and subjects with more than one sibling. Since a punch of 1 in the dichotomized variable column was the low-order digit, the 1 on the digit emitter was wired instead of zero. In all other respects the panels remained as described. The following paragraphs describe in detail the program steps of the required computations.

Only program steps 3 through 8 are permitted to operate on the detail cards. On program 3, a 1 is emitted into the counter. Since a low-order digit is read on card 0001, the 1 in the counter is transferred to FS1 on program 5. (Nothing is added to the counter on program 4, since FS1 has been cleared by the X78 card preceding the first detail card.) Similarly, the continuous score (75) is transferred on program 8 to GS1-2. On the second detail card (since again a low-order digit is read), program 4 adds the stored count of 1 to the counter and program 5 returns the augmented count of 2 to FS1. Similarly, program 7 adds the accumulated sum of the continuous scores to the one read in on card 0002 and program 8 returns the augmented sum to storage. The same process is repeated for cards 0003 and 0004, as both have low-order digits on the dichotomized variable. A high-order digit is read on card 0005, and therefore no impulse is available to transfer calculator selector 1. As a result, the unit count is stored in FS3 instead of FS1, and the sum of continuous scores is accumulated in GS3-4 instead of GS1-2. The information from card 0006 is treated in the same manner.

Table 4 gives the values which appear in the storage units and counter after completion of each program step, beginning with step 8. These program steps will be discussed in detail below.

The X79 card has the values given in table 3. Of these, M_t is stored in FS2, σ_t in FS4, and 1/N in MQ.

On program 9 the sum of the low-order digit count enters into the counter from FS1. On program 10 the sum of the high-order digit count enters negatively from FS3 and a program test for balance suppression is performed.

TABLE 4

Computational Example

Step	Factor Storage				M Q	Counter	General Storage			
	1	2	3	4			1	2	3	4
Computations During the X79 Card										
8	4		2			4		350		215
9		94.17		17.65	.16667	4				
10						2				
11						0				
12						.66668				
13										
14						.67168				
15			4							
16	67					0				
17						350000				
18										
19					87.500	0				
20										
21						87.500				
22						-6.670				
23						-6.675				
24						0		-6.67		
25						-6.67000				
26					-.376	Rem.				
27						0				
28										-.376
29			67							
30						.376				
31						0				.376
Computations During the 17th X76 Card										
32	68					67		.362		
33						-.01				
34						-.02				
35					.362					
36						0				
Computations During the 77 Card										
37	91					.670000		.176		
38					1.851	·Rem.				
39						0				
40						.696776				
41						.696826				
42						0				.6968
43						.6968				

All values are shown when they first enter the counter or storage units, remaining there until a change is indicated.

The result of this test determines which of several alternate program steps will be suppressed (see table 2). In our example, a plus balance results and program levels 13 and 18 are suppressed. Program 11 clears the counter; and on 12 the count for the larger sub-group is multiplied by the reciprocal of the total number of cases to yield p, the proportion of the larger group. This result is rounded off on program 14 and, on program 16, stored in FS2 to two decimal places. Program 15 serves to shift the N for the larger sub-group from FS1 to FS3 and is required to make available additional storage facilities. On program 17 the sum of the continuous scores for p is read into the counter and is divided, on program 19, by the N for this group to yield the mean for the continuous scores for the larger sub-group. After the result has been transferred back into the counter, M_t is subtracted by negative entry from FS2. The result of this operation is rounded off in the last place and then stored to two decimal places in GS1-2. On program 26 it is read out again into the counter and divided by σ_t. The quotient is stored in GS3-4; but its sign is reversed on programs 30 and 31, since the low-order digit group is the larger, to assure that the final coefficient will have the correct sign. The program levels passed over in the description serve to clear and shift the required storage units.

The next step in the calculation consists of reading in the corresponding value of y for the value of p which has been computed while the X79 card passed through the calculator. The deck of X76 cards constitutes the table required for this purpose. Successive trial values of p are read from the X76 cards and are compared with the computed value of p stored in the calculator. Whenever the two values balance, the corresponding value for y is stored in the calculator. On step 32 the computed value of p is read into the counter from FS3. Then on 33 the read-in trial value is subtracted. On 34 a zero check test is performed.

We found the value of p for our example to be .67. Therefore, no changes occur in the calculator until card # 16 of the X76 deck passes. No zero check impulse is emitted because observed and trial values agree; and on the next card the y value is entered into FS4, since step 35 is permitted to be active. Table 4 shows the values found after calculation is complete on X76 #17.

Since a zero check impulse will be emitted on all subsequent X76 cards, no further changes occur until the 77 card enters the calculator and impulses the remaining computations. On step 37 p is read into the counter into fifth place and is divided by y on step 38. After resetting the counter this result is multiplied on step 40 by the product of the preceding calculations stored in GS3-4. This final product is then rounded in the second position on step 41. The last two steps reduce the coefficient to four decimals, which are punched on the 77 card to give the biserial correlation coefficient of .6968 for our example.

SUMMARY

This paper describes a method for computing biserial correlation coefficients on the 604 Electronic Calculating Punch in a single-step procedure, provided the mean and standard deviation for the total sample are known for the continuous score. The method is also adaptable to the case where categorical scores must first be dichotomized before biserial r can be computed. A computational example as well as all required wiring diagrams and program charts are given.

REFERENCES

1. L. Crowell, A. Katcher, and S. F. Miyamoto, "Self-Concepts of Communication Skills and Performance in Small Group Decisions," Speech Monographs, 22 (1955), pp. 20-27.
2. H. E. Garrett, Statistics in Psychology and Education (New York: Longmans, Green & Co., 1946)
3. IBM Type 604 Electronic Calculating Punch Manual of Operation (seventh revision).
4. W. Johnson, "A Simplified Method for Machine Calculation of the Components Used in the Formulas for Biserial and Point Biserial Correlation Coefficients," Report # 001 058.25.02, U. S. Naval School of Aviation Medicine, Pensacola, Florida (1953).
5. C. E. Lunneborg, C. E. Wright, and A. F. Ax, "A Set of Statistical Boards for the 604 Electronic Calculating Punch," Mimeographed Report, University of Washington School of Medicine (1955).

A METHOD FOR THE PACKAGED PROCESSING OF A STATISTICAL ANALYSIS ON THE IBM 650

Leon H. Somerall and Nicholas A. Habibe

Air Weather Service
Asheville, North Carolina

The primary purpose of this paper is the presentation of a logical program design for the "one package" processing or continuous solution of a standard statistical problem on the IBM 650 Magnetic Drum Data Processing Machine.

The one package concept embodies all the functions (except the preliminary sorting or assembly and the printing of the output) formerly attained through "step" or discontinuous procedures requiring the use of diverse IBM equipment such as sorters, summary punches, collators, accounting machines, and/or calculating equipment such as calculating punches, card-programmed calculators. Proper program logic enables the 650 to simulate functions featured in component-type punched-card equipment.

The standard statistical problem to be discussed consists basically of two parts: (1) performance of a bivariate frequency distribution by a table look-up method and (2) evaluation of some statistics pertaining to the distribution.

The printed result, figure 1, shows two families of imposed class intervals X, Y, within which the single-valued variables or observations x, y will fall uniquely and respectively, and the family of groups Z, called the parameter of the distribution. Each member of the family Z contains all the members of X and Y.[a]

Part 1 seeks to build, by groups of data, a drum table of frequencies which will reflect the fitting of each x, each y, and the pair (x, y) into the appropriate class interval member X_i, Y_j, and (X_i, Y_j), respectively, to add a count each in the three drum cells containing the corresponding frequencies.

[a] The terminology and notation at the end of the paper will be found helpful.

		X_1	X_2	. . .	X_m		
	Y_1	f_{111}	f_{211}	. . .	f_{m11}	$g_{11} = \sum_{i=1}^{m} f_{i11}$	$\bar{x}_1$
	Y_2	f_{121}	f_{221}	. . .	f_{m21}	$g_{21} = \sum_{i=1}^{m} f_{i21}$	$\bar{y}_1$
Z_1	. . .	. . .	. . .	. . .	. . .	. . .	σ_{x1}
	Y_n	f_{1n1}	f_{2n1}	. . .	f_{mn1}	$g_{n1} = \sum_{i=1}^{m} f_{in1}$	σ_{y1}
		$f_{11} = \sum_{j=1}^{n} f_{1j1}$	$f_{21} = \sum_{j=1}^{n} f_{2j1}$	. . .	$f_{m1} = \sum_{j=1}^{n} f_{mj1}$	N_1	r_1
	Y_1	f_{112}	f_{212}	. . .	f_{m12}	$g_{12} = \sum_{i=1}^{m} f_{i12}$	$\bar{x}_2$
	Y_2	f_{122}	f_{222}	. . .	f_{m22}	$g_{22} = \sum_{i=1}^{m} f_{i22}$	$\bar{y}_2$
Z_2	. . .	. . .	. . .	. . .	. . .	. . .	σ_{x2}
	Y_n	f_{1n2}	f_{2n2}	. . .	f_{mn2}	$g_{n2} = \sum_{i=1}^{m} f_{in2}$	σ_{y2}
		$f_{12} = \sum_{j=1}^{n} f_{1j2}$	$f_{22} = \sum_{j=1}^{n} f_{2j2}$	. . .	$f_{m2} = \sum_{j=1}^{n} f_{mj2}$	N_2	r_2
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	Y_1	f_{11p}	f_{21p}	. . .	f_{m1p}	$g_{1p} = \sum_{i=1}^{m} f_{i1p}$	$\bar{x}_p$
	Y_2	f_{12p}	f_{22p}	. . .	f_{m2p}	$g_{2p} = \sum_{i=1}^{m} f_{i2p}$	$\bar{y}_p$
Z_p	. . .	. . .	. . .	. . .	. . .	. . .	σ_{xp}
	Y_n	f_{1np}	f_{2np}	. . .	f_{mnp}	$g_{np} = \sum_{i=1}^{m} f_{inp}$	σ_{yp}
		$f_{ip} = \sum_{j=1}^{n} f_{1jp}$	$f_{2p} = \sum_{j=1}^{n} f_{2jp}$	. . .	$f_{mp} = \sum_{j=1}^{n} f_{mjp}$	N_p	r_p

Figure 1

A count is also added into a cell reserved to keep track of how many valid observations have been processed. Thus, the table will furnish a running record of how many x's, how many y's, and how many (x, y)'s fall respectively in each class X_i for all values of Y, in each class Y_j for all values of X, and in each pair (X_i, Y_j). The members of X and Y may be in random sequence and not necessarily equi-spaced. If the class intervals are equi-spaced and continuous, the distribution will not necessarily require table look-up.

Each group Z_k in the printed output, figure 1, represents a corresponding group Z_k of input cards, shown in figure 2 arranged in groups $Z_1, Z_2, \ldots, Z_p$. Prior to processing in the 650, the cards are assembled in groups of the parameter Z by means of a sorting operation. Each card contains three fields: group Z_k to which the card belongs, the observation x, and the observation y.

Part 2 is concerned with the determination of means, standard deviations, and correlation coefficients. Of course, any other statistics may be included by appropriate programming.

The printed output, a modified replica of the drum table, is obtained by processing in the IBM 407 Accounting Machine the punched output of the IBM 650. The headings $X_1, X_2, \ldots, X_m$ in figure 1 are not present in the drum table, figure 4. They are printed by 407 control panel wiring. The drum area assigned to the table of frequencies and statistics must contain (m+3) (n+1) cells to house the frequencies, the identification, and the statistics. On the printed output, X runs horizontally and Y runs vertically. On the drum table, however, the X and Y directions are interchanged to facilitate 650 punching and 407 printing of the output one line per card.

The program design is based on the general scheme of effecting the following data analysis on each valid card of any group Z_k:

(a) Validity check on x, y

(b) Relating each member X_i of X to the family Y by means of the frequency f_{ik}, which is the sum of the frequencies f_{i1k}, f_{i2k}, . . . , f_{ink} in any one class X_i for all values of Y: $f_{ik} = \sum_{j=1}^{n} f_{ijk}$.

(c) Relating each member Y_j of Y to the family X by means of the frequency g_{jk}, which is the sum of the frequencies f_{1jk}, f_{2jk}, . . . , f_{mjk} in any one class Y_j for all values of X: $g_{jk} = \sum_{i=1}^{m} f_{ijk}$.

(d) Relating each member X_i of X to each member Y_j of Y by means of the frequency f_{ijk} of the simultaneous pair (X_i, Y_j).

Each pair (x, y) is considered valid if x and y comply with some imposed conditions of restraint. An invalid card is disregarded. The frequencies or counts f_{ik}, g_{jk}, and f_{ijk} are properly recorded on the drum table as a result of an interesting table look-up technique to be explained below. Thus, table look-up is the principal data processing link between the information contained in the input cards and the corresponding counts to be recorded in the drum table of frequencies.

(e) Progressive recording of $\sum x$, $\sum y$, $\sum x^2$, $\sum y^2$, $\sum xy$, N for the evaluation, by groups, of the mean $(\bar{x}_k)$ of all $\bar{x}$'s, the mean $(\bar{y}_k)$ of all y's, the standard deviation (σ_{xk}) of all x's, the standard deviation (σ_{yk}) of all y's, and the correlation coefficient r_k between all the x's and all the y's in each group.

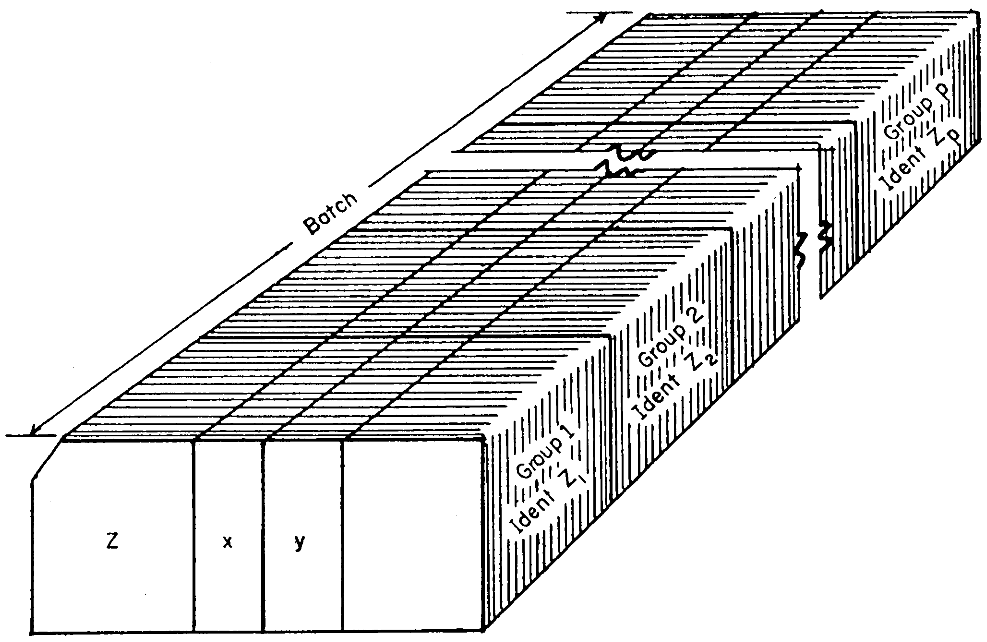


Figure 2. Batch or Deck-Sorted or Assembled in Groups Z_1, Z_2, . . ., Z_p

PROGRAM LOGIC

An overall procedure for the logical processing of data is shown in the block diagram of figure 3, which conforms to the block diagramming conventions outlined in the manual "IBM 650 Problem-Planning Aids." It is generally applicable in situations where it is desired to cause the production of a set of outputs whenever a difference exists between the Z value of two adjacent cards. This output will provide the result of the analysis of a Z group.

The following discussion ties in the block diagram, figure 3, and the drum display, figure 4.

BLOCKS

01. Early during the conception of the flow chart, a distinct fact is recognized: separate reading instructions must be provided for the first card only; the reading of any other card but the first must be accomplished with a

different set of instructions, as provided by block 05. Exclusive reading instructions for the first cards is dictated by (a) the uniqueness of the position of this card in the deck, (b) the necessity of clearing (block 02) the drum area assigned to the table of frequencies before the data analysis is to begin, and (c) the prevention of an output routine at the start of the data process. An instruction in block 01 reads the first card of the first group onto the drum, i. e., the observations x, y and the identification Z enter the read input area where the Z is then called Z_r.

02. Storage clearing is a prerequisite before the analysis of each group Z_k starts. Instructions in this block clear to zero all the memory locations assigned to cumulative data (frequencies or counts and other summations). It is generally faster to clear storage during the output routine. If the amount of time devoted to output computations of statistics in block 07 is less than the time available for maximum punch output rate, then storage clearing can be incorporated in the output routine to take advantage of the remaining available time. In this case, block 02 is operative for the first card of the first group only instead of the first card of each group. In other words, when storage clearing takes place at output time, block 02 is still operative for the very first card because there is no convenient way of divorcing storage clear in the output routine to have it apply to the very first card.

03. This set of instructions stores the first Z_r, i.e., the Z of the first card read in each group, in another location $L(Z_s)$ where it is renamed Z_s. The purpose of the transfer is two-fold: (1) to compare the Z of the first card of every group with the Z's of subsequent cards in order to detect a change in Z or transition to a new group, and (2) to identify the output. The use of the punch output area for the storage of Z_s permits most conveniently

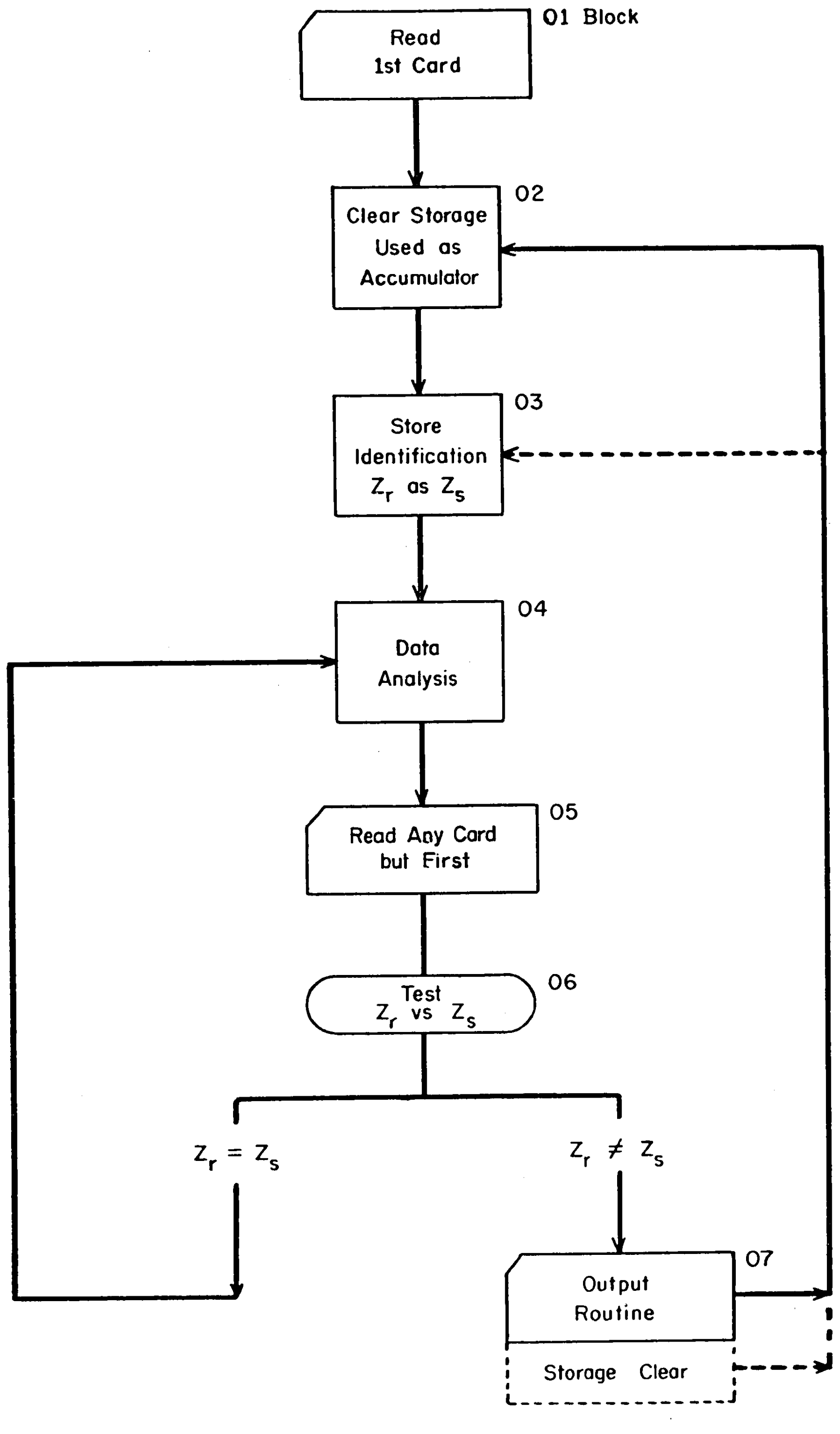


Figure 3. Block Diagram

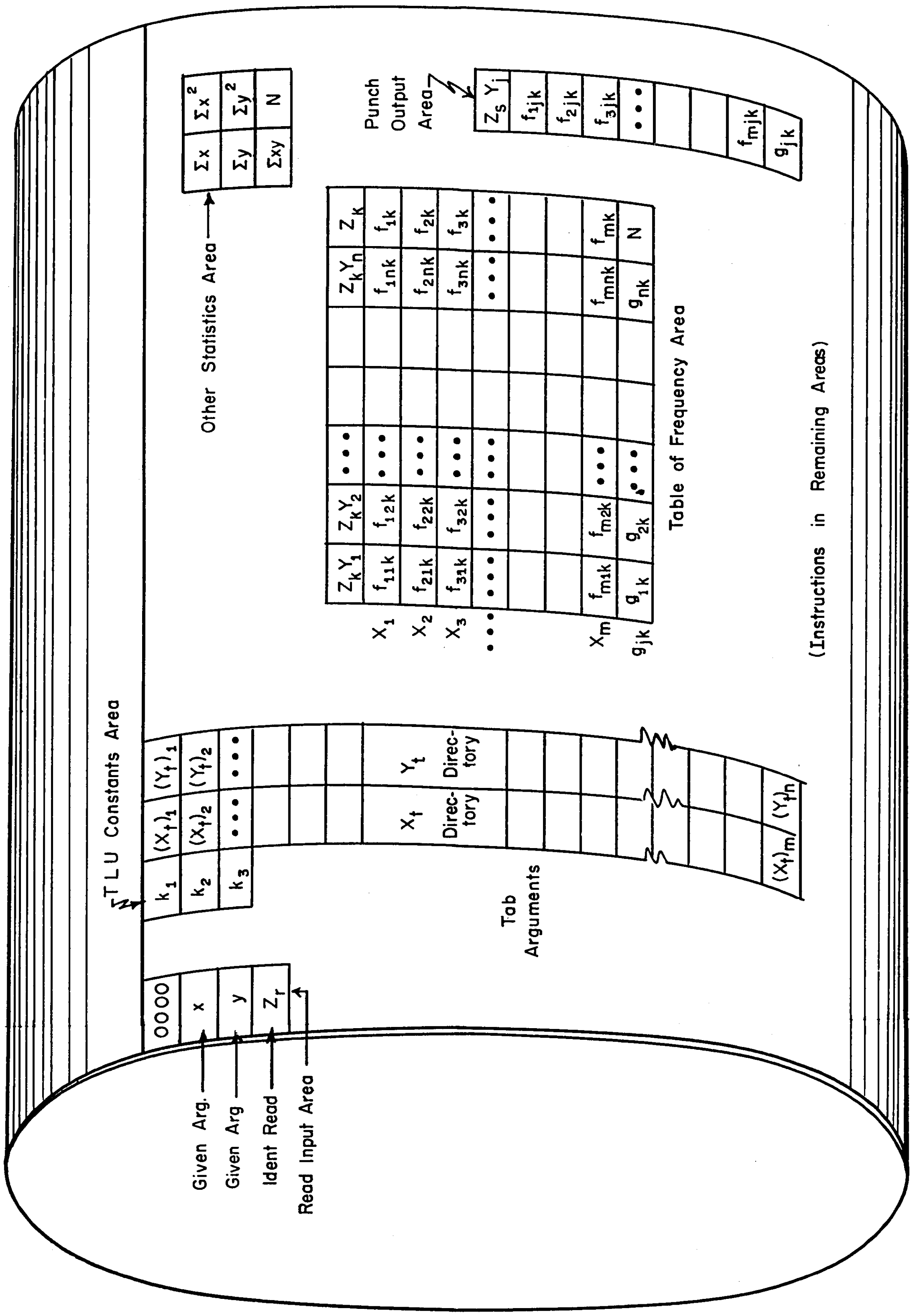


Figure 4. Drum Display

the performance of the two-fold function of Z_s.

04. In this particular problem, data analysis represents the following collection of operations applied to each card: (a) validity check, (b) progressive recording of cumulative frequencies by a table look-up method, (c) progressive recording of summations for the evaluation of statistics. Other operations, based on the individual values of the read elements, could be performed during this data analysis phase.

A validity check, a test to determine if the read variables x, y fulfill certain criteria or conditions of restraint, precedes data analysis. When the imposed conditions are not met, the card is disregarded and the program logic proceeds to read another card.

Progressive recording of cumulative frequencies for any valid card is accomplished through the following routine: (a) at the beginning of each group the drum area assigned to cumulative data is cleared to zero, as explained above; (b) a table look-up operation selects the appropriate terminal value $(X_t)_i$ of the class interval X_i in which the read x belongs, to yield the address $L(f_{ik})$ of f_{ik}, that is, one of the m locations assigned for the accumulation of the frequencies in the class X_i for all values of Y; the address being known, f_{ik} is sent to the accumulator to be increased by a count of one and then returned to its original location; (c) a similar routine for the read y yields the address of g_{jk}, that is, one of the n locations assigned for the accumulation of the frequencies in the class Y_j for all values of X; the contents of this address are then increased by a count of one as explained above; (d) the address of f_{ijk}, that is, one of the mn locations for the accumulation of the frequencies in the pair (X_i, Y_j), is attained via the addresses of f_{ik} and g_{jk}; f_{ijk} is then increased by a count of one.

Progressive recording of $\sum x$, $\sum y$, $\sum x^2$, $\sum y^2$, $\sum xy$, and N does not involve table look-up because these summations have a fixed reference (or data address) throughout the data process. The search for addresses mentioned in block 04 (b), (c), and (d) requires a variable address scheme capable of giving all the locations of the table of frequencies as a function of the read variables. The instructions to store the counts or frequencies are automatically modified for each card in accordance with the individual values of x, y. A number of schemes could be devised to accomplish address searching, some of which may or may not involve table look-up and/or computation. In the present problem, TLU and computation are featured in the determination of variable addresses because of the flexibility afforded by such a scheme.

In the conventional sense, TLU is the process of best matching a given argument to one of a collection of tabulated arguments and reading off the tabulated function associated with the tabulated argument. In the 650 sense, however, the standard TLU operation yields the address or location of one out of many tab arguments equal to or next larger (if no equal exists) than the given argument. The tab function, stored with or a constant number of locations away from its corresponding tab argument, is then extracted by making use of the found tab argument address. The tab function could be any function, such as a function of the argument, an instruction, an address, a constant, etc.

The standard TLU operation is susceptible to many refinements to accomplish address searching more efficiently. In the TLU technique presented in this paper, the tab function can be computed as a function of the location of the corresponding tab argument. Then, because no extraction takes place, drum area for the storage of tab functions is unnecessary. Since in this problem there are two given arguments x and y, two families of tab arguments will be required. The families of terminal values X_t and Y_t of the class intervals X and Y, which are the required tab arguments, are referred to here as the X_t directory and the Y_t directory. The location of the frequencies in the drum table of frequencies constitute the "tab functions."

The following drum areas are required for the execution of the table look-up techniques:

Area Name	Contents	Role
Read input	A value of x	Given x argument
	A value of y	Given y argument
X_t directory	Terminal values $(X_t)_1, (X_t)_2, \ldots, (X_t)_m$	Tab X_t arguments
Y_t directory	Terminal values $(Y_t)_1, (Y_t)_2, \ldots, (Y_t)_n$	Tab Y_t arguments
TLU constants	k_1, k_2, k_3	Computation of $L(f_{ik}), L(g_{jk}), L(f_{ijk})$

In order to store counts for the frequency in each class X_i for all values of Y, in each class Y_j for all values of X, and in each pair (X_i, Y_j), the locations $L(f_{ik})$, $L(g_{jk})$, and $L(f_{ijk})$ must be computed for every card with valid x and y using the equations

$$L(f_{ik}) = L(X_t)_i + k_1 \quad (1)$$

$$L(g_{jk}) = 50\, L(Y_t)_j + k_2 \quad (2)$$

$$L(f_{ijk}) = L(f_{ik}) + L(g_{jk}) + k_3 \quad (3)$$

The numerical values of k_1, k_2, k_3 are dependent on the relative position of the drum areas assigned to the directories and to the table of frequencies.

The sequence of events for obtaining the unknown addresses in equations (1), (2), (3) and for storing the counts in each address follows:

Step	Operation	Result
1. TLU	Read input x vs. X_t directory	$L(X_t)_i$
2. TLU	Read input y vs. Y_t directory	$L(Y_t)_j$
3. Computation	Equation (1)	$L(f_{ik})$
4. Computation	Equation (2)	$L(g_{jk})$
5. Computation	Equation (3)	$L(f_{ijk})$
6. Store count	One unit added to contents of $L(f_{ik})$	f_{ik}
7. Store count	One unit added to contents of $L(g_{jk})$	g_{jk}
8. Store count	One unit added to contents of $L(f_{ijk})$	f_{ijk}

To illustrate the table look-up technique presented, suppose

(a) a card in the first group ($k = 1$) undergoing data analysis contains the following: $x = x_5$, $y = y_6$, $Z_k = Z_1$, x and y valid.

(b) x_5 falls in the class interval X_5 ($i = 5$), whose terminal value is $(X_t)_5$; y_6 falls in the class interval Y_6 ($j = 6$), whose terminal value is $(Y_t)_6$.

(c) the family X contains nine class intervals, the family Y contains seven, and the drum area layout is the one shown in figure 5.

The problem is

(a) find the drum location $L(f_{51})$ of the frequency f_{51}, i.e., the location (0768) of the frequencies of occurrence in the class X_5, group Z_1, regardless of Y.

(b) find the location $L(g_{61})$ of the frequency g_{61}, i. e., the location (0673) of the frequencies of occurrence in the class Y_6, group Z_1, regardless of X.

(c) find the location of $L(f_{561})$ of the frequency f_{561}, i. e., the location

	Constants
k_1	364
k_2	-22,077
k_3	-773

X_t directory

Y_t directory

0400 $(X_t)_1$	0450 $(Y_t)_1$
0401 $(X_t)_2$	0451 $(Y_t)_2$
0402 $(X_t)_3$	0452 $(Y_t)_3$
0403 $(X_t)_4$	0453 $(Y_t)_4$
0404 $(X_t)_5$	0454 $(Y_t)_5$
0405 $(X_t)_6$	0455 $(Y_t)_6$
0406 $(X_t)_7$	0456 $(Y_t)_7$
0407 $(X_t)_8$	
0408 $(X_t)_9$	

$Y \longrightarrow$

$X \downarrow$

Z_1, Y_1	Z_1, Y_2	Z_1, Y_3	Z_1, Y_4	Z_1, Y_5	Z_1, Y_6	Z_1, Y_7	Z_1	Z_1
0414 f_{111}	0464 f_{121}	0514 f_{131}	0564 f_{141}	0614 f_{151}	0664 f_{161}	0714 f_{171}	0764 f_{11}	$\overline{x}_1$
f_{211}	f_{221}	f_{231}	...	...	...	...	f_{21}	$\overline{y}_1$
f_{311}	...	...					f_{31}	S_1
...							f_{41}	σ_1
					0668 f_{561}		0768 f_{51}	r_1
							f_{61}	
							f_{71}	
							f_{81}	
							f_{91}	
0423 g_{11}	0473 g_{21}	0523 g_{31}	0573 g_{41}	0623 g_{51}	0673 g_{61}	0723 g_{71}	N	

Figure 5. Drum Layout for the Illustrative Example

(0668) of the frequency of occurrence in the pair of classes (X_5Y_6).

(d) increase by one count each, the contents of the locations $L(f_{51})$, $L(g_{61})$, and $L(f_{561})$.

The solution is:

Step 1. TLU of given argument x_5 versus the X_t directory yields 0404, the location of the tab argument $(X_t)_5$ equal to or next higher than x_5.

Step 2. TLU of given argument y_6 versus the Y_t directory yields 0455, the location of the tab argument $(Y_t)_6$ equal to or next higher than y_6.

Step 3. Equation (1) gives:

$$L(f_{51}) = 0404 + 364 = 0768$$

Step 4. Equation (2) gives:

$$L(g_{61}) = 50(0455) - 22,077 = 0673$$

Step 5. Equation (3) gives:

$$L(f_{561}) = 0768 + 0673 - 773 = 0668$$

Steps 6 through 8. The contents of locations 0768, 0673, and 0668 are increased by one unit.

05. The program logic shown by the block diagram requires two blocks of instructions for the 650 to execute card reading. Block 05 reads all cards except the first card of the first group, which is read by block 01.

06. This test compares the identification Z_s of the first card of a group with the identification of each card within the group. If the equal condition exists, the card read (just before the test) is part of the group currently under process, in which case the data analysis and card reading will not be disturbed until an unequal condition is detected. If the test yields an unequal result, the card read (just before the test) is the first of a new group, implying that the

data collected and analyzed on an individual card basis is complete for the group; card reading is then temporarily interrupted until the output routine takes place.

07. The output routine incorporates two subroutines: (a) computation or further data analysis on a group basis and (b) assembly of the data in the punch output area. Means, standard deviations, correlation coefficients, etc., are computed in subroutine (a). The data to be punched is transferred from the table of frequencies to a punch output area by subroutine (b). It will generally require the computation of the data address (especially if a "loop" is used) of the instruction to "go and get" the appropriate data items for transfer to the output area. The data address of this instruction can very conveniently be inserted in a "clear to zero" instruction so that a "write and erase" subroutine can be developed. Upon completion of the output routine the data process starts all over again, this time with block 02 or 03.

The generalized program described above can be easily modified to do a variety of accounting and statistical problems.

TERMINOLOGY AND NOTATION

Conditions of Restraint	Criteria for acceptance of data or restrictions imposed in the course of a process.
Batch or Deck	A collection of groups of cards.
Group	A collection of items or cards having a common characteristic (such as identification).
Valid	Data which meet the criteria for acceptance or fulfill the imposed restrictions.
Table Look-up	The process of best matching a given argument with one of a collection of tabulated arguments to read off the tabulated function associated

	with the tabulated argument.
Given Arg	Given argument, a value of the variable which, by comparison with a tabulated argument, is used to search for the tabulated function.
Tab Arg	Tabulated argument, a value which best matches the given argument.
Tab Function	The value associated with the tab argument.
Directory	A relatively small table or drum area consisting of tab arg, tab functions, constants, etc., sometimes necessary for access to a larger drum area or table.
Extraction	The process of obtaining the tab function by means of 650 coded steps.
Class Interval	A range of values comprising and contained within an initial value and a terminal value.
x, y	Observations or variables to be distributed in class intervals. They have a finite though variable number of digits and belong in one class interval only, provided the class intervals do not overlap.
X	Family of class intervals in random sequence and not necessarily equi-spaced, representing all the members $X_1, X_2, \ldots, X_m$.

X_i	Any one member of the family X, one class interval whose range is from $(X_0)_i$ to $(X_t)_i$ $(i=1, 2, \ldots, m)$.
X_0	Family of initial values of the family X, representing all the members $(X_0)_i$, $(X_0)_2$, $(X_0)_m$.
$(X_0)_i$	Any one member of the family X_0, the initial value of the class interval X_i.
X_t	Family of terminal values of the family X, representing all the members $(X_t)_1$, $(X_t)_2$, $\ldots$, $(X_t)_m$.
$(X_t)_i$	Any one member of the family X_t, the terminal value of the class interval X_i.
Y	Family of class intervals in random sequence and not necessarily equi-spaced, representing all the members Y_1, Y_2, $\ldots$, Y_n.
Y_j	Any one member of the family Y, one class interval whose range is from $(Y_0)_j$ to $(Y_t)_j$ $(j=1, 2, \ldots, n)$.
Y_0	Family of initial values of the family Y, representing all the members $(Y_0)_1$, $(Y_0)_2$, $\ldots$, $(Y_0)_n$.
$(Y_0)_j$	Any one member of the family Y_0, the initial value of the class interval Y_j.
Y_t	Family of terminal values of the family Y, representing all the members $(Y_t)_1$, $(Y_t)_2$, $\ldots$, $(Y_t)_n$.

$(Y_t)_j$	Any one member of the family Y_t, the terminal value of the class interval Y_j.
Z	Family of group identifications representing all the members $Z_1, Z_2, \ldots, Z_p$.
Z_k	Any one member of the family Z ($k = 1, 2, \ldots, p$).
f_{ijk}	A frequency or number of times the pair of observations (x, y) falls within the pair of class intervals (X_i, Y_j) in a group Z_k.
f_{ik}	$= \sum_{j=1}^{n} f_{ijk}$, any one member $f_{1k} = \sum_{j=1}^{n} f_{1jk}$, $f_{2k} = \sum_{j=1}^{n} f_{2jk}$, $\ldots$, $f_{mk} = \sum_{j=1}^{n} f_{mjk}$ of a family f of summations of frequencies f_{ijk} falling within any one class interval X_i for all values of Y in a group Z_k.
g_{jk}	$= \sum_{i=1}^{m} f_{ijk}$, any one member $g_{1k} = \sum_{i=1}^{m} f_{i1k}$, $g_{2k} = \sum_{i=1}^{m} f_{i2k}$, $\ldots$, $g_{nk} = \sum_{i=1}^{m} f_{ink}$ of a family g of summations of frequencies f_{ijk} falling within any one class interval Y_j for all values of X in a group Z_k.
$\bar{x}_k$	Arithmetic mean of all x's in group k.
$\bar{y}_k$	Arithmetic mean of all y's in a group k.
σ xk	Standard deviation of all x's in a group k.

σ_{yk}	Standard deviation of all y's in group k.
r_k	Correlation coefficient of y vs. x in a group k.
Z_r	Group identification of any card just read.
Z_s	Group identification of any group's first card just stored.
L(U)	Drum address or location of U, where U is any quantity.
i	$= 1, 2, \ldots, m$.
j	$= 1, 2, \ldots, n$.
k	$= 1, 2, \ldots, p$.
o	Initial.
r	Read (past tense).
s	Stored.
t	Terminal.

AN INTERPRETIVE SUBROUTINE FOR THE SOLUTION OF SYSTEMS OF FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS ON THE 650

Franz Edelman
RCA Laboratories

INTRODUCTION

The problem of solving ordinary differential equations occurs very frequently in technical computations. The writing and subsequent checking of such programs is extremely wasteful of both machine and programming time. Because of this, a general purpose program has been written[a] which will automatically perform this function.

To use this subroutine, the programmer need specify only initial conditions, the equations to be solved and their number, as well as the precision required in the solution.

The programmer has a choice of two distinct methods for solution, those of Runge-Kutta-Gill and Milne. The program for the latter provides automatic choice of optimum intervals. A choice for a fixed or floating point precision criterion is also provided.

The maximum number of equations which may be solved simultaneously is limited only by the memory capacity of the machine and is in the neighborhood

[a]The program is written for the Interpretive System described by V. M. Wolontis of the Bell Telephone Laboratories, Murray Hill, New Jersey, in IBM Applied Science Division Technical Newsletter No.11.

of thirty.

The basic logic of this subroutine is in no way restricted to be applied solely to a 650 program. Therefore it is felt that, because of the flexibility and compactness of the logic, this program will lend itself well to transcription into other machine or pseudo-languages. Complete flow diagrams are furnished for this purpose at the end of the report.

In designing this general purpose program an effort has been made to anticipate as many of the likely requirements and needs as possible, particularly with regard to generality, flexibility, and convenience of use.

There are several phases of a typical differential equation problem where flexibility is nothing less than essential:

The Method of Solution. In general, integration methods which keep track of truncation error, such as that of Milne[1], are preferable to those which do not, for obvious reasons. There are, however, two disadvantages to the former: first, a starting procedure is required before the method can be applied; and, second, under some conditions this method may become unstable. The first shortcoming is only an inconvenience, but the second renders the method useless. For these reasons a second method is available, which is that of Runge-Kutta-Gill[2]. This method also doubles as the starting procedure for the Milne method.

Output. At each step of the integration the type and amount of the output as well as the format are completely within the control of the user of the program. All the information required by the user as output is available from a given set of locations arranged in such a manner as not to tax his memory unnecessarily.

The user may, if he wishes, change the nature and format of his output during a run depending on some predetermined criterion.

The Criterion for Terminating a Solution. Here again is something which varies from one problem to the next. Therefore the program is designed in such a way that the user may program this criterion, which may depend on any or all of the variables, to suit his purpose.

Precision. This consideration applies only to the Milne method. The manner in which the precision is computed is important, since its proper choice may in some cases reduce machine time by more than 50 percent. This program offers two choices. The increasing or decreasing of the integration interval is governed by a comparison of the predicted and corrected values of the dependent variables using a predetermined number of either (1) decimal places or (2) significant figures. In either case this number is specified by the user as a floating point integer. If the solutions oscillate about zero with an amplitude of order unity, for example, then criterion (1) is considerably faster. Thus if at some point the solution is very near zero, criterion (1) would remain undisturbed while criterion (2), noting the poor agreement between predicted and corrected values (in terms of significant figures), would proceed to reduce the integration interval unnecessarily. On the other hand, if the solutions run uniformly to very large or very small numbers, criterion (2) is faster and more realistic. In other words, criterion (2) is always safe, even though sometimes excessively time-consuming. The solutions to $y'' + y = 0$

and $y'' - y = 0$, for example, illustrate this point.

Reducing the Integration Interval. In general, the variations of the dependent variables and their derivatives are not violent enough to warrant such a drastic reduction of the integration interval as an order of magnitude, for example. Nevertheless it may happen that the customary reduction by the factor 0.5 is insufficient. Therefore this factor has been left to the discretion of the user. Unless otherwise instructed, however, the program will use the factor 0.5.

Increasing the Integration Interval. When the precision of the computations becomes too large, implying that the integration interval has become too small, the subroutine will automatically double the interval until the precision reaches its prescribed range.

The above-mentioned features are believed to make this program flexible enough to cover a wide range of problems.

For a better understanding of what is to follow, a list of definitions is given below:

y_0 is the independent variable; y_{0j} is its $(j + 1)^{th}$ value.

y_i, $i = 1, 2, \ldots, N$, are the dependent variables; y_{ij} are their values at y_{0j}.

$f_{ij} = (dy_i/dy_0)_j$, $i = 0, 1, 2, \ldots, N$, are the derivatives of y_i with respect to y_0 at y_{0j}. Thus $f_{0j} = 1$ for all j.

k_{ij}, r_{ij}, q_{ij} are defined analytically in the following section.

h is a positive or negative integration increment.

N is the number of equations to be solved.

T is the required precision.

H is the fraction by which h is to be reduced.

D is the tolerance factor for doubling the integration interval.

E_{ij} is the deviation between the predicted and corrected values of y_i at y_{0j}. E_j is the maximum E_{ij} at y_{0j}.

A table summarizing the functions of all special locations, which may be referred to by the user of this program, is shown at the end of this report.

MATHEMATICAL BACKGROUND

Any system of ordinary differential equations may be reduced, by suitable changes of variables, to an equivalent system of first order ordinary differential equations:

$$y_i' = f_i(y_0, y_1, y_2, \ldots, y_N), \quad (i = 1, 2, \ldots, N), \qquad (1)$$

subject to the initial conditions

$$y_i(y_{00}) = y_{i0}, \quad (i = 1, 2, \ldots, N). \qquad (2)$$

The program described here will automatically solve such a system, requiring of the user only the specification of the functions f_i and the initial conditions.

For the sake of completeness the equations of both methods employed are given below.

The equations of Runge-Kutta-Gill are

$$\left.\begin{aligned}
k_{ij} &= hf_{ij} && (3)\\
r_{i,\,j+1} &= a_j k_{ij} - b_j q_{ij} && (4)\\
y_{i,\,j+1} &= y_{ij} + r_{i,\,j+1} && (5)\\
q_{i,\,j+1} &= q_{ij} + 3r_{i,\,j+1} - c_j k_{ij} && (6)
\end{aligned}\right\}\begin{array}{l} j = 0,\ 1,\ 2,\ 3\\ i = 0,\ 1,\ 2,\ \ldots,\ N\end{array}$$

The initial values q_{i0} are taken to be zero at the first point and equal to q_{i4} of the previous point, subsequently. a_j, b_j and c_j are sets of four constants, each given by

$$a_j = \left(\tfrac{1}{2},\ 1-\sqrt{\tfrac{1}{2}},\ 1+\sqrt{\tfrac{1}{2}},\ \tfrac{1}{6}\right) \tag{7}$$

$$b_j = \left(1,\ 1-\sqrt{\tfrac{1}{2}},\ 1+\sqrt{\tfrac{1}{2}},\ \tfrac{1}{3}\right) \tag{8}$$

$$c_j = \left(\tfrac{1}{2},\ 1-\sqrt{\tfrac{1}{2}},\ 1+\sqrt{\tfrac{1}{2}},\ \tfrac{1}{2}\right) \tag{9}$$

The equations of Milne are the "predictor":

$$y^{(p)}_{i,\,j+1} = y_{i,\,j-3} + \frac{4h}{3}\left(2f_{ij} - f_{i,\,j-1} + 2f_{i,\,j-2}\right) \tag{10}$$

and the "corrector":

$$y^{(c)}_{i,\,j+1} = y_{i,\,j-1} + \frac{h}{3}\left(f^{(p)}_{i,\,j+1} + 4f_{ij} + f_{i,\,j-1}\right) \tag{11}$$

$$\begin{array}{l} j \quad 3,\ 4,\ 5\ \ldots\\ i \quad 1,\ 2,\ \ldots,\ N\end{array}$$

Superscripts p and c indicate predicted and corrected values, respectively.

An RKG solution is obtained by successive applications of equations (3) through (6) for each j from 0 to 3, together with four applications of equation (1). y_{i4} are the values of the solutions at the mesh points determined by h.

A Milne solution is obtained by using (10) followed by (1) to evaluate $f^{(p)}_{i,\ j+1}$. Then (11) and another application of (1) determine $y^{(c)}_{i,j+1}$, and $f^{(c)}_{i,\ j+1}$ respectively.

The maximum deviation (over i) between predicted and corrected values divided by 29 (see reference 1) is used to judge the size of the integration interval. The procedure for changing this interval will be described in detail below.

Both of the above methods yield fourth order precision, i. e., have truncation errors of order h^5.

THE PROGRAM

Every subroutine, if it is to be complete and self-contained, requires of the user a set of instructions which enable the subroutine to perform its intended function and which supply to the subroutine the required return address(es) to the main program. Such a set of instructions is termed the "calling sequence."

This subroutine is designed in such a manner that the calling sequence, which is written as part of the user's program, has two distinct portions.

The first portion is that in which the functions f_i are to be programmed and stored in the appropriate locations. The data required for the computation of f_i are available from given locations.

The second portion of the calling sequence is concerned with preparing the output and testing for continuation of the solution. As mentioned previously, both of these operations are entirely within the control of the user. The two parts of the calling sequence are separated by a TRSUB instruction.

This separation was necessary because we do not wish to execute the output program every time control is transferred to the first part of the calling sequence. For example, when using the Milne method, $y^{(p)}$, $y'^{(p)}$, $y^{(c)}$, $y'^{(c)}$ are computed in that order. The second and fourth of the above quantities are computed in the first part of the calling sequence. The second part of the calling sequence, however, is to be executed only after computation of $y'^{(c)}$.

Three program parameters must be supplied by the user before execution of the program.

1. $|N|$ is entered into location 997
2. h is entered into location 998
3. T is entered into location 999

The initial conditions y_{00}, y_{10}, y_{20}, $\ldots$, y_{N0} are entered into locations 500 to 500+ $|N|$. All input is in normalized floating point form. After the above data are stored, the calling sequence is as follows:

```
loc. p :  TRSUB   + 0    204    q    600
loc. q :  Start computation of f_i, storing them in 550 +  i, i = 1, 2, ..., N.
loc. r :  TRSUB   +0     204    s    620
```

loc. s : Select output from locs. 500 to 500+|N| and 551 to 550+|N|

for punching.

Punch output.

Select from the same locations the criterion for terminating

the solution.

Test this criterion.

If solution is to be continued, transfer to loc. 640.

The y_{ij} values for computing the functions f_i between locations q and r are available from locations 500 to 500+|N| .

If the parameter N is stored as a positive number, the program will compute the first three points (not counting the initial point) with the RKG method. Then it will switch to the Milne method until the integration interval is to be decreased. If N is stored with a negative sign, the program will use the RKG method throughout.

(If it should be desirable to compute m RKG points ($m > 3$) before switching to the Milne method, this can be achieved by placing behind the program deck the instruction

```
loc. 625: Set B   + 0    050    655    m + 1.)
```

The subroutine may be used any number of times within the same program. The only restriction here is that |N| is not changed. If |N| is to be changed, the program must first be reloaded.

It might be pointed out here that the system of equations to be solved need not be simultaneous. Several unrelated differential equations may be solved at the same time.

DOUBLING THE INTERVAL

When using the Milne method, the routine examines at each point the sum of the absolute values of the maximum deviations for the last three points. When this sum is exceeded by the quantity $29x10^{-T}/D$, the interval is doubled. The value of D is largely a matter of experimentation. Obviously, there has to be some separation between the regions in which the interval is decreased or increased. The region in which the interval is left unchanged is given by

$$\frac{29x10^{-T}}{3D} \leq E_j < 29x10^{-T}.$$

In fact, the ratio of these two bounds must be at least 30, since the error is multiplied by approximately that when the interval is doubled. It was found here in one specific instance that D = 200 was near a machine time optimum. However, D, which is located in 788, may be changed should the user so desire.

After doubling of the interval has taken place, the routine is prevented from doubling again until 3 more RKG points, followed by one Milne point, have been computed.

It has been found useful to indicate on the output any modification of h which may have taken place. Therefore the program will punch a card with a two in the second word when the doubling routine is called in.

DECREASING THE INTERVAL

Before going into this process it is necessary to specify the two precision criteria which were mentioned previously.

If the entire program deck is used, the quantity

$$E_j = \max_i \left| y_{ij}^{(p)} - y_{ij}^{(c)} \right|$$

will be compared to 29×10^{-T}. If the last card is removed (Card 70 Deck 1), then the quantity

$$E_j = \max_i \left| 1 - \frac{y_{ij}^{(p)}}{y_{ij}^{(c)}} \right|$$

will be compared to 29×10^{-T}. These are the fixed and floating point precision criteria, respectively.

Three different routines for decreasing the interval are built into this program:

1. If the interval is to be decreased at point p_1 in the middle of a run (and if h has remained unchanged at the previous point p_0), the program will discard that point, back up to the previous point p_0, re-enter the RKG method and compute three points with the decreased interval. It will then switch again to the Milne method. An indicator card with an H will be punched.

2. If, after having just decreased the interval, the program finds upon leaving the RKG procedure that it is to decrease the interval again, it will again back up to the original point p_0 and proceed from there as before. The program will repeat this procedure as often as necessary. The results at point p_0 are not repunched during this operation.

3. If the point p_0 happens to be the initial point of the solution, i. e., if

the initial h has been chosen too large, the program will, upon leaving the RKG procedure, punch a spacer card, repunch the initial point and start again with the reduced interval. It will repeat this procedure until the proper initial h has been found.

If, in this last case, the programmed switch is set to STOP, a conditional stop will occur, displaying the value of h which caused it. This was done in order to give the user the opportunity to modify the starting procedure. If, when the stop occurs, the storage entry switch is set to plus and the start button is pressed, routine c will be executed. If the storage entry switch is set to minus, routine a will be executed.

There is one more special situation which may arise. Suppose we are solving a single differential equation of large order N. Then it is conceivable that we may not wish all derivatives up to order N-1 to be examined for precision. Suppose we wish to examine only the first n ($n < N$) functions, i. e., $y, y', y'', \ldots, y^{(n-1)}$. This can be achieved by placing behind the subroutine a card loading into location 723 the instruction

$$+\ 0 \quad 109 \quad B \quad 717,$$

where $B = 1000 - N + n$.

The fraction H is stored in location 789 and may be changed by the user, if he so desires.

STORAGE

Permanent. The program occupies locations 600-999. This breaks down as follows : The interpretive program (Deck 1), which corresponds to the entire solution of the problem, occupies 180 locations. This interpretive program contains some forty-odd instructions which are functions of the program parameter $|N|$. Those instructions are compiled with two basic language programs (Deck 2) totalling 190 instructions. The basic language programs, which contain no loops, are executed only once. Numerical constants (Deck 3) take up 20 locations, and parameters 10 locations.

Erasable. The buffer between the user's program and the subroutine, locs. 500 to $500+|N|$ and 551 to $550+|N|$, is erasable storage before and after execution of the subroutine. Location 550 cannot be used during execution of the subroutine. The remainder of these two bands is not used.

If the RKG method only is used (N negative), the erasable storage requirement is locations $500-2(|N| + 1)$ to 499.

If the Milne method is used (N positive), the erasable storage requirement is locations $500-13|N|$ to 499. The appropriate block of erasable storage which is used to carry along the function values at previous points may not be used during execution of the subroutine, but is available before and after. For the sake of compactness the erasable storage was so arranged as to leave free as much of lower memory as possible. For all practical purposes N is unlimited in size if the RKG method is used. In the case of the Milne method, which requires much more storage, $N = 30$ leaves about 150 locations for the main program. The overall limit on N, dictated by the size of the buffer, is 49.

TIMING

The routine was so designed that no operations are performed unnecessarily or repeated needlessly. N is tested only once, and the basic control operations are executed only once. It is felt that this routine is nearly as efficient as a custom-made program. If the RKG method is to be used, then no instructions pertaining to the Milne method are executed, with the exception of the one-time execution of a single SET C instruction at the beginning of the program.

The execution time per point, excluding the time required for the computation of f, is about $(6 + 3 \, |N|)$ seconds for the RKG method and $(2.5 + 1.5 \, |N|)$ seconds for the Milne method.

A detailed time study for a particular problem involving a single second order equation showed that the custom-made program required 14 seconds per point whereas the subroutine required 16 seconds per point.

MISCELLANEOUS REMARKS

The loop box and COUNT register are available anywhere for the user's own program. A word of caution is appropriate at this point. It is to be noted that the program in which the functions f_i are computed by the user is executed four times for each RKG point and twice for each Milne point. This is important if the coefficients of the equation are given in tabular form. It would then be necessary to count the number of executions before giving the loop instruction.

Two unconditional stops may occur during execution of the subroutine :

1. A stop at location 635 indicates $|N| < 1$ or $|N| \geq 100$.
2. A DIV-CHECK at location 717 indicates $y^{(c)}_{i,\ j+1} = 0$

   for some i, if the floating point precision criterion is used.

When tracing, the program is so arranged as to trace through each of the two sections of the user's program only once. Therefore it is advisable, when checking out this portion of the program, to enter the routine with some non-trivial initial conditions.

The program may be translated with the standard Bell translation program. Deck numbers distinguish the three types of cards used.

The error term E_j is stored in location 659 so that the skeptic may include it in his output if he so desires.

EXAMPLE

The program for a specific sample problem is shown in Figure 1. The equation to be solved is

$$y^{(4)} + y' y'' y''' + y y' = \sin x,$$

subject to the initial condition

$$y(0) = 1, \quad y'(0) = 0, \quad y''(0) = 1, \quad y'''(0) = 0.$$

To reduce this equation to a system of first order equations, we define

$$y' = u$$

$$u' = v$$

$$v' = w.$$

Then $w' = \sin x - uvw - yu$.

The initial conditions become

$$y(0) = 1, \quad u(0) = 0, \quad v(0) = 1, \quad w(0) = 0.$$

Four significant figures are required, and the initial h is taken to be 0.1.

The output is to consist of the quantities x, y, u, v, w, and w' punched on one card.

The solution is to be terminated when x exceeds unity or y becomes negative, whichever happens first.

The entire program requires 17 instructions.

FLOW DIAGRAMS

Complete flow diagrams for this subroutine are shown in figures 2 and 3. A numeral or letter outside an instruction box denotes the location of the instruction represented by that box.

Locations α and β contain the transfers to the main program.

SUMMARY TABLE

In order to aid in the application of this program a summary table is given below :

Item	Purpose	Location
$\pm N$	Input	997
h	Input	998
T	Input	999
y_0	Input - Output	500
y_i	Input - Output	501 to 500+$\|N\|$
f_i	Output	551 to 550+$\|N\|$
1st address	TR to SUBR, initial.	600
2nd address	TR to SUBR, after computation of f_i	620
3rd address	TR to SUBR, for continuation of solution	640
H	Parameter	789
D	Parameter	788
E_j	Error term	659
+ 0050655 (m + 1)	To compute m initial RKG points	625
+0109B717	To test only up to $y^{(n-1)}$ $B = 1000 - N + n$	723
Storage	Locations used by program	600 to 999
Storage	Erasable storage for RKG	500-2($\|N\|$ + 1) to 499
Storage	Erasable storage for Milne	500-13$\|N\|$ to 499
STOP TRACE	Tracing control	601, 621, and 641

FIGURE 1. PROGRAM FOR THE SOLUTION OF

$$y^{(4)}+y^{I}y^{II}y^{III}+yy^{I}=\sin x$$

LOC.	ALPHA OPERATIONS	±	0_1	A OR 0_2	B	C	
100	TRSUB	+	0	204	101	600	ENTER SUBROUTINE
101	MPY	+	3	502	503	000	UV
102	MPY	+	3	000	504	200	UVW
103	MPY	+	3	501	502	201	YU
104	SIN	+	0	303	500	000	SIN X
105	SUB	+	2	000	200	000	SIN X−UVW
106	SUB	+	2	000	201	554	W′= SIN X−UVW−YU
107	MOVE	+	9	003	502	551	Y′=U,U′=V,V′=W
108	TRSUB	+	0	204	109	620	RE ENTER SUBROUTINE
109	MOVE	+	9	005	500	250	MOVE OUTPUT
110	MOVE	+	9	000	554	255	//
111	PCH	+	0	410	250	255	PUNCH
112	SUB	+	2	500	505	000	X−1
113	TRS	+	0	201	116	114	TEST X−1
114	MOV	+	9	000	501	000	Y
115	TRS	+	0	201	640	116	TEST Y
116	STOP	+	0	000	117	000	STOP
117	DISPL	+	9	999	999	999	
500		+	0			00	LOAD X_0
501		+	1			50	// Y_0
502		+	0			00	// U_0
503		+	1			50	// V_0
504		+	0			00	// W_0
505		+	1			50	// I
997		+	4			50	//+N
998		+	1			49	// h
999		+	4			50	// T

FIGURE 2. FLOW CHART FOR RUNGE–KUTTA–GILL METHOD AND STARTING PROCEDURE FOR MILNE METHOD

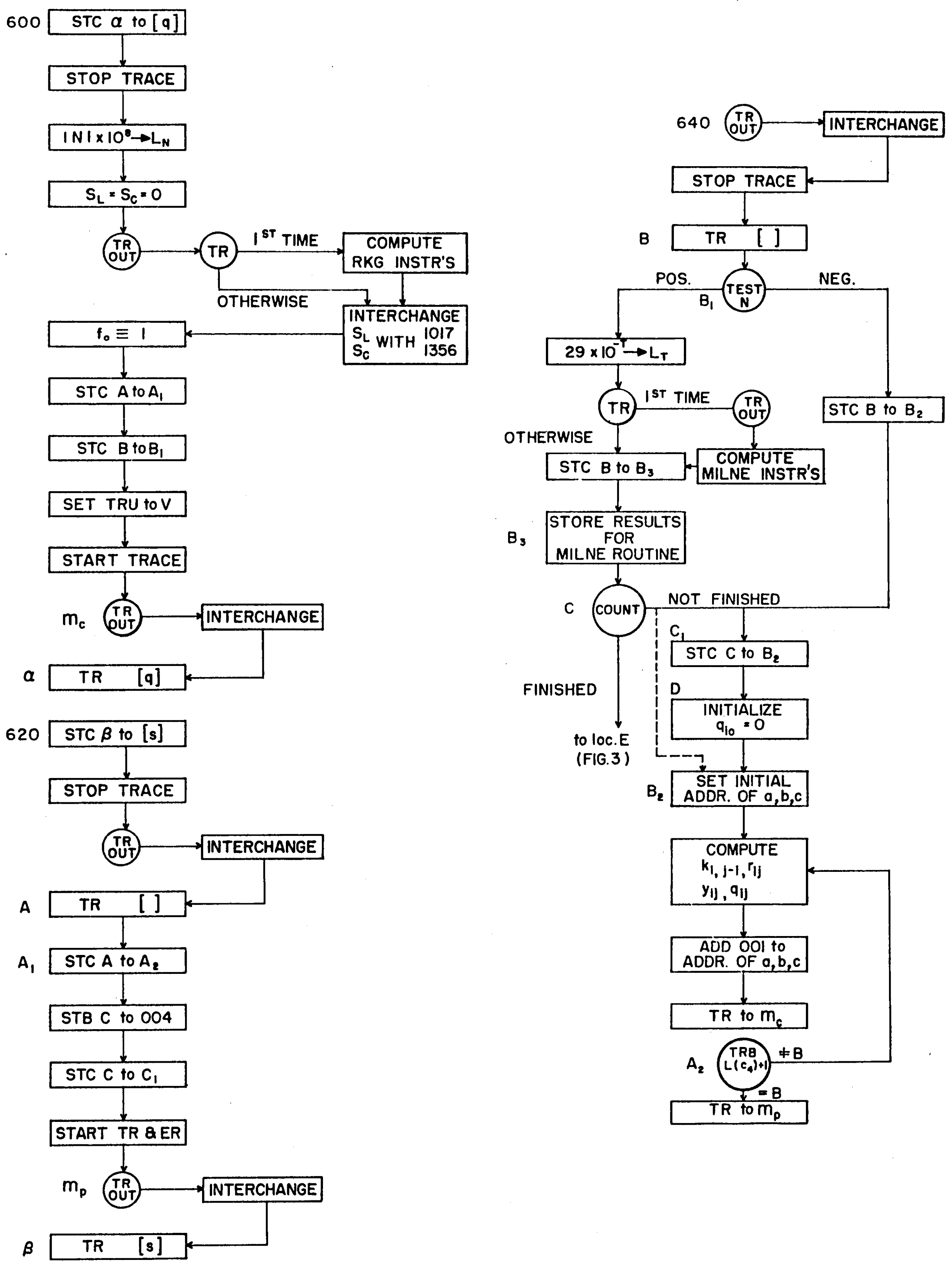
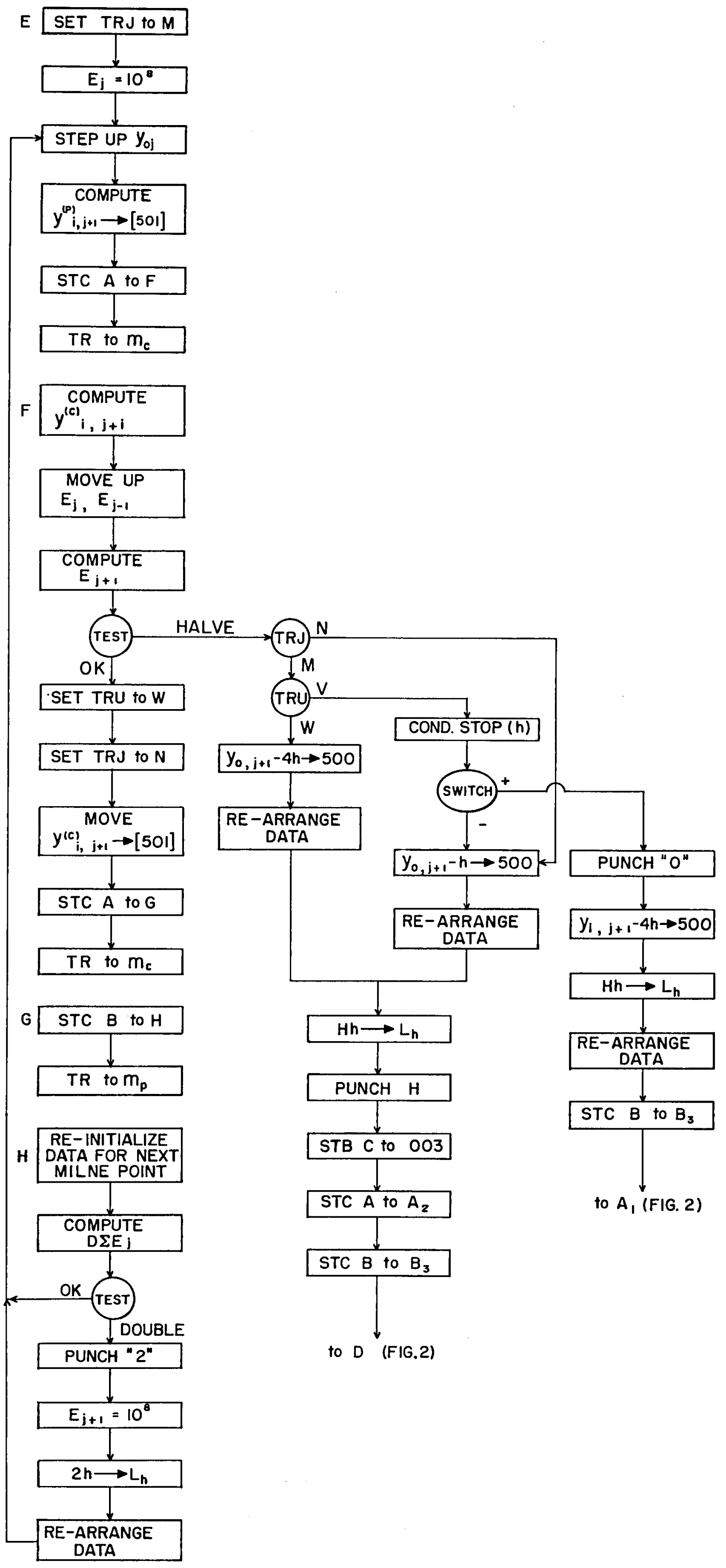


FIGURE 3. FLOW CHART FOR MILNE METHOD



REFERENCES

1. W. E. Milne, Numerical Calculus (Princeton University Press, 1949), p. 134.

2. S. Gill, "A Process for the Step-by-Step Integration of Differential Equations in an Automatic Digital Computing Machine," Camb. Phil. Soc., Proc., 47 (1951), p. 96.

DOUBLE TABLE LOOK–UP ON THE IBM 650

Robert H. Goerss[a]
Dodco, Inc.

INTRODUCTION

One of the most difficult types of computation in a computing laboratory using digital equipment is double table look-up of location of data presented in a three-dimensional form. Specifically, Y is presented as a function of two variables X and Z.

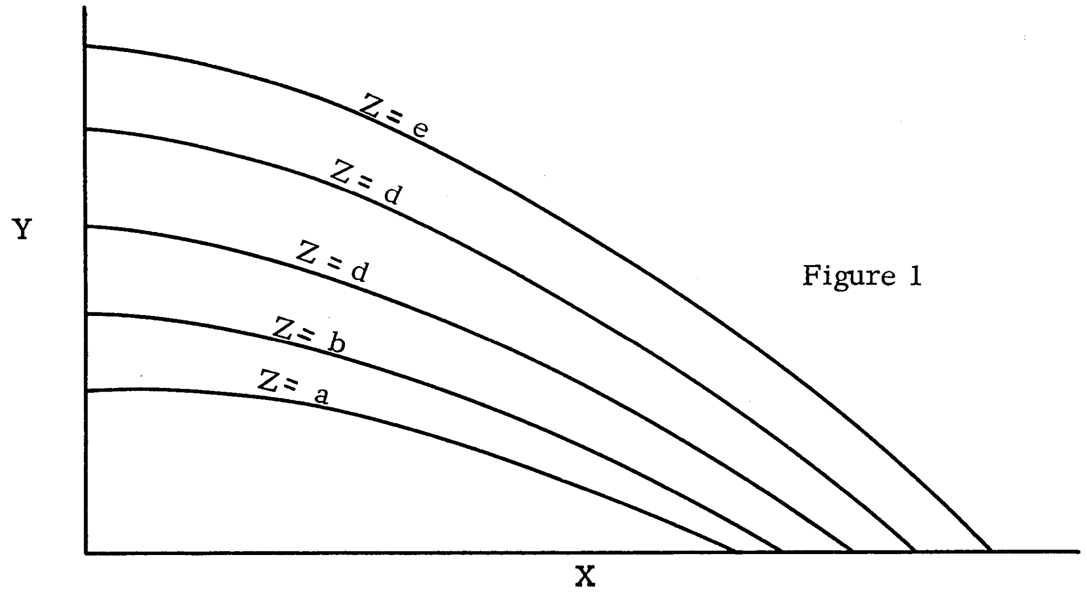


Figure 1

For example, the presentation of drag of both aircraft and missiles is frequently in this form where drag is a function of velocity and altitude. One method of approach to this problem is to reduce the data to a function of one variable or a fit in either one or both variables. Unfortunately, this laborious procedure must be repeated for each set of data.

[a] Now associated with Electronic Associates, Inc.

This paper presents a general method of double table look-up of Y as a function of two variables X and Z. The procedure outlined has the advantage that tables of various sizes can be handled without changes in the code, subject to storage limitations of the equipment used. While the basic concept can be used on any medium or large-scale stored-program computer, this paper presents the storage limitations, code, and an example of a 5 x 5 table for the IBM 650.

METHOD

On the 650 the storage limitations are that the number of points in both X and Z be less than 48 and the number of values of Y be less than 1800 minus the storage requirements of the problem using this subroutine.

The subroutine will find the necessary values for linear interpolation for Y as a function of any X and Z in the range given by the table. The only restriction on the tabulated data is that Y must be given for the same points of X for all values of Z and vice versa. To interpolate linearly for Y as a function of X_0, Z_Z, where $X_I < X_0 \leq X_{II}$ and $Z_A < Z_Z \leq Z_B$, we need the following numbers: X_I, X_{II}, Z_A, Z_B, $Y_{(X_I Z_A)}$, $Y_{(X_{II} Z_A)}$, $Y_{(X_I Z_B)}$, $Y_{(X_{II} Z_B)}$

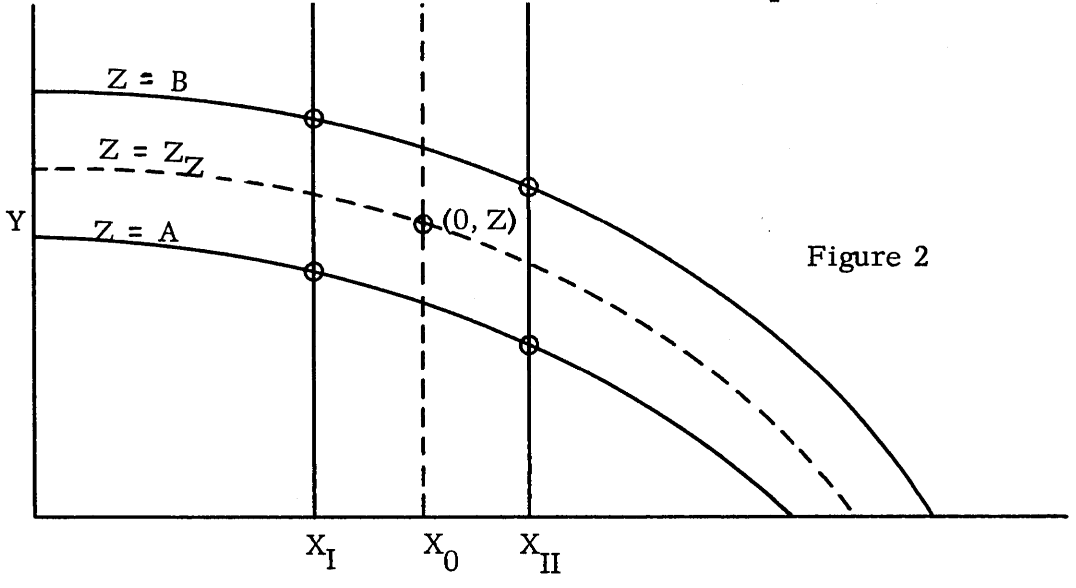


Figure 2

To make the explanation of the presentation of the data and the code easily understood, we define X as the primary argument and Z as the secondary argument. The arguments and data are fed into the 650 as follows: The primary arguments are loaded in ascending order starting in storage location 0050; the secondary arguments are loaded in ascending order starting in location 0100; and the table of Y's are loaded in ascending order first with respect to the primary argument, then with respect to the secondary argument. Tables 1, 2 and 3 show the storage of primary and secondary arguments and the functions for a 5 x 5 table.

TABLE 1

Location of Primary Argument

0050	X_1
0051	X_2
0052	X_3
0053	X_4
0054	X_5

TABLE 2

Location of Secondary Argument

0100	Z = a
0101	Z = b
0102	Z = c
0103	Z = d
0104	Z = e

TABLE 3

Storage of Function Y(XZ)

x = 1-5 z = a-e

Location	Contents	Location	Contents	Location	Contents
0200	Y(1a)	0210	Y(1c)	0220	Y(1e)
0201	Y(2a)	0211	Y(2c)	0221	Y(2e)
0202	Y(3a)	0212	Y(3c)	0222	Y(3e)
0203	Y(4a)	0213	Y(4c)	0223	Y(4e)
0204	Y(5a)	0214	Y(5c)	0224	Y(5e)
0205	Y(1b)	0215	Y(1d)		
0206	Y(2b)	0216	Y(2d)		
0207	Y(3b)	0217	Y(3d)		
0208	Y(4b)	0218	Y(4d)		
0209	Y(5b)	0219	Y(5d)		

Inspection of Table 3 indicates that it could be considered a table of tables; that is, tables of Y as a function of X are arranged as in ascending order of Z. Associated with both arguments is a table of secondary modifiers. The secondary modifiers serve as an index to which table of Y as a function of the primary argument should be used. Specifically, adding the data address of the secondary modifier to the location of X_I yields the location of Y_{IA}.

TABLE 4

Storage of Secondary Modifiers

Location	Contents	
0150	00 0150 0002	z = a
0151	00 0155 0002	z = b
0152	00 0160 0002	z = c
0153	00 0165 0002	z = d
0154	00 0170 0002	z = e

Associated with the primary argument is a number called the jump modifier. Adding the data portion of the jump modifier to the data address of Y_{IIA} yields the data address of Y_{IB}. It is important to note that for a table of any size the only changes that need to be made in the routine other than the data and arguments are the secondary and jump modifiers.

The code itself consists of 31 instructions, which logically break down into eight steps. In each of these steps a Store Distributor command with a variable data address is generated in the lower accumulator. This operation is then executed, and the contents of the distributor are stored in a desired storage position. The arguments X_0 and Z_Z are assumed stored in locations 0030 and 0031. The following operations are performed:

1. Find the location of Z_B by 650 table look-up; store Z_B in 0035.
2. Find the location of Z_A by subtracting 1 from the data address of Z_B; store Z_A in 0034.

3. Find the location of the secondary modifier by adding 50 to the address of Z_A; store the secondary modifier in 0046.
4. Find the location of X_{II} by 650 table look-up; store X_{II} in 0033.
5. Find the location of X_I by subtracting 1 from the location of X_{II}; store X_I in 0032.
6. Find the location of Y_{IA} by adding the secondary modifier to the location of X_I; store Y_{IA} in 0036.
7. Find the location of Y_{IIA} by adding 1 to the location of Y_{IA}; store Y_{IIA} in 0037.
8. Find the location of Y_{IB} by adding the jump modifier to the location of Y_{IIA}; store Y_{IA} in 0038.
9. Find the location of Y_{IIB} by adding 1 to the location of Y_{IB}; store Y_{IIB} in 0039.

The complete code and a list of storage positions follows:

TABLE 5
Storage of Input and Output

Location	Contents
0030	X_0
0031	Z_Z
0032	X_I
0033	X_{II}
0034	Z_A
0035	Z_B
0036	Y_{IA}
0037	Y_{IIA}
0038	Y_{IB}
0039	Y_{IIB}

Jump Modifier

0040 00 0004 0002

TABLE 6

Other Storage

Location	Contents			Function
0041	00	0000	9998	Address Modifier
0042	00	0050	0002	Address Modifier
0043	00	0001	0002	Address Modifier
0044	69	0000	0004	Load Distributor Command
0045	69	0000	0012	Load Distributor
0046				Temporary Storage

This system was used by the author as a subroutine in a problem coded with the floating decimal interpretive system described by V. M. Wolontis in IBM Applied Science Division Technical Newsletter No. 11. For that reason the data address of the last command is 1095, a location for a return to the Wolontis system. In this connection it should be pointed out that when using any floating coded decimal system, or when using a floating decimal arithmetic unit, care must be taken that the argument is only one order of magnitude. This can be accomplished by using a modified argument equal to the argument increased by the quantity $1 \times 10^{\alpha}$, where α is such that all modified arguments have the same exponent.

If there are n values of X and m values of Z, the secondary modifiers and jump modifier can be expressed in terms of n and m as follows:

	Command	Data	Instruction
Jump modifier	00	n - 1	0002
Secondary modifiers	00	0150	0002
	00	0150 + n	0002
	00	0150 + 2n	0002
	00	0150 + 3n	0002
		.	
		.	
		.	
	00	0150 + (m-1)n	0002

With slight modifications, this system can be expanded for higher order interpolation or to handle two related functions of the same two arguments.

ACKNOWLEDGMENT

The author wishes to express thanks to D. O. Dommash, President, Dodco, Inc., for his helpful encouragement and for suggesting the useful notation of primary and secondary arguments. The author also wishes to express his thanks to Sherwood Skillman of the RCA Laboratories for his help in checking the code.

IBM Trade Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-1
PRINTED IN U.S.A.

PROBLEM: DOUBLE TABLE LOOK-UP WRITTEN BY: R. H. GOERSS

INSTR NO.	LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
		ABBRV.	CODE	DATA	INSTRUCTION	
	0001	RAL	65	0044	0002	Reset lower accumulator basic load
						distributor command.
	0002	LD	69	0031	0003	Load Z_Z (secondary argument) into the
						distributor.
	0003	TLU	84	0100	8002	Find the location of Z_B.
	8002	LD	69	L (Z_B)	0004	Load Z_B into distributor.
	0004	STD	24	0035	0005	Store Z_B 0035.
	0005	SL	16	0041	8002	Find the location of Z_A by subtracting
						1 from the location of Z_B, add 2 to
						the instruction address.
	8002	LD	69	L (Z_A)	0006	Load Z_A into the distributor.
	0006	STD	24	0034	0007	Store Z_A in 0034.
	0007	AL	15	0042	8002	Find the location of the secondary
						modifier and increase instruction
						address by 2.
	8002	LD	69	L (SM)	0008	Load secondary modifier into
						distributor.
	0008	STD	24	0046	0009	Store secondary modifier in temporary
						storage.
	0009	RAL	65	0045	0010	Reset lower accumulator load basic
						load distributor command.
	0010	LD	69	0030	0011	Load primary argument X_0 into the
						distributor.
	0011	TLU	84	0050	8002	Find the location of X_{II}.
	8002	LD	69	L (X_{II})	0012	Load X_{II} into distributor.
	0012	STD	24	0033	0013	Store X_{II} in 0033.
	0013	SL	16	0041	8002	Find location of X_I by subtracting 1
						from the location of X_{II}.
	8002	LD	69	L (X_I)	0014	Load X_I into distributor.
	0014	STD	24	0032	0015	Store X_I in 0032.
	0015	AL	15	0046	8002	Find the location of Y_{IA} by adding
						the secondary modifier to the location
						of X_I.
	8002	LD	69	L (Y_{IA})	0016	Load Y_{IA} into the distributor.
	0016	STD	24	0036	0017	Store Y_{IA} in 0036.
	0017	AL	15	0043	8002	Find the location of Y_{IIA} by adding
						1 to the location of Y_{IA}.
	8002	LD	69	L (Y_{IIA})	0018	Load Y_{IIA} in distributor.
	0018	STD	24	0037	0019	Store Y_{IIA}.
	0019	AL	15	0040	8002	Find the location of Y_{IB} by adding
						jump modifier to the location of Y_{IIA}.
	8002	LD	69	L (Y_{IB})	0020	Load Y_{IB} into distributor.
	0020	STD	24	0038	0021	Store Y_{IB} in 0038.
	0021	AL	15	0043	8002	Find location of Y_{IIB}.
	8002	LD	69	L (Y_{IIB})	0022	Load Y_{IIB} in distributor.
	0022	STD	24	0039	1095	Store Y_{IIB}.

PAGE ______ OF ______

Form No. 32-7710

IBM® DATA PROCESSING

INTERNATIONAL BUSINESS MACHINES CORP.
590 MADISON AVENUE, NEW YORK 22, N. Y.

Litho. in U. S. A.