

to  $x$ , that the process will not yield reliable values of  $J_n(x)$ . Indeed this is just the situation and this fact provides a means of determining a lower bound ( $\hat{n}_{min}$ ) for  $\hat{n}$  relative to  $x$ .

An upper bound ( $n_{max}$ ) for  $\hat{n}$  will be forced on the generation of  $J_n(x)$  by the restrictions on the size of numbers that may be used. In the experimental work carried out for the determination of these bounds, the IBM 650 with the automatic floating point device was used. Using this system, all numbers must fall within  $(10^{-51}, 10^{49})$ . It was found that the maximum number size was attained, not from generation of the  $Y_n(x)$ , but from calculation of the normalizing constant  $K$  in equation (3). In addition, of course,  $\hat{n}$  will be limited by the amount of storage space available in the particular machine, but this limitation is not considered here.

Proceeding as suggested above, experimental values have been determined for  $\hat{n}_{min}$  and  $\hat{n}_{max}$  relative to  $x$ . In presenting the results graphically it was found convenient to consider separately the two ranges  $10^1 \leq x < 500$  and  $10^{-51} \leq x < 10^1$ , which may be found in figures 1 and 2 respectively. A useful set of approximating curves for  $\hat{n}_{min}$  and  $\hat{n}_{max}$  are also illustrated in figures 1 and 2. The time necessary to calculate both  $J_n(x)$  and  $Y_n(x)$  for  $n=0, 1, \dots, \hat{n}$  is approximately linear with  $\hat{n}$ . For the IBM 650, time in sec  $\sim .13\hat{n}$ .

### Conclusion

1. It is possible to generate tables of  $J_n(x)$  and  $Y_n(x)$  by their recursive relations (1) and (6), and obtain reliable results. The generation of  $J_n(x)$  is carried out by a "backwards" process using (1), and the  $Y_n(x)$  are generated forwards using (6).

2. By using single precision floating point arithmetic (8 digit mantissa and 2 digit characteristic) it is possible to obtain results with a maximum error of 1 in the sixth significant digit. No attempt was made to determine what effect the use of double precision arithmetic would have on the number of correct significant digits obtainable.

3. In general, the accuracy is not improved by selecting an  $\hat{n}$  greater than  $\hat{n}_{min}$ .

4. The choice of  $\hat{n}_{max}$  for  $\hat{n}$  in generating  $J_n(x)$  was sufficient in all cases to guarantee that  $Y_n(x) < 10^{49}$  for  $n \leq \hat{n}$ .

5. This technique, though fast, is best used when  $J_n(x)$  and  $Y_n(x)$  are desired for an entire range of  $n$ , rather than for a specific  $n$ .

## A SUBROUTINE METHOD FOR CALCULATING LOGARITHMS

R. W. BEMER, I. B. M. Corporation, 590 Madison Avenue, New York 22, N. Y.

This note describes a method for calculating logarithms which is particularly suitable to computers with Variable Execution Time operations (VET). It was first used for the IBM 705 in the PRINT I and Autocoder systems and is currently being adapted for the IBM 709. The basic method is adaptable for computers which do not have VET operations but may not be competitive with other methods in these cases. The method is described for use with floating point arguments, but it may be adapted, with some preliminary modifications, for computation of logarithms of fixed point arguments.

It is desired to compute  $\log_b N$ , where the argument or number  $N$  is represented in floating point as:

$$N = M \cdot b^E \quad \text{where the exponent } E \text{ is an integer and}$$

(decimal)	$.1 \leq M < 1.$
(binary)	$.5 \leq M < 1.$

Then,  $\log_b N = E + \log_b M$

The basic principle of this method is cyclic multiplication of the mantissa  $M$  by one or a series of specially chosen multipliers, as necessary, until the resultant product falls in a specially chosen range starting at 1.0. A relaxed Taylor series for the  $\log(1+x)$  is then used, represented here as  $P_a(x)$ , where  $a$  represents the number of digits of accuracy in decimal. Let the final product in the range be  $m_k$ , so that:

$$1 \leq m_k \leq (1 + \Delta), \text{ where } m_i = M \prod_{i=1}^k A_i$$

$$\text{Then, } \log_b M = \log_b m_k - \sum_{i=1}^k \log_b A_i$$

Various considerations of the number base and machine characteristics dictate the value of  $\Delta$  and the multipliers chosen. The multiplier  $A_i$  is selected from a table, one entry for each interval of the value of  $m_i$ . It is chosen such that when multiplied by the largest value of  $m$  in each interval, the resultant product does not exceed  $1 + \Delta$ . For both binary and decimal methods outlined here, either 1, 2 or 3 multiplications may be required before the resultant product exceeds 1. The average number of times thru the loop is about 1.7 for both methods.

### Decimal Machines

The interval chosen here was on the basis of the first significant digit of the running product.  $\Delta$  was chosen as .1 ( $\Delta$  must not exceed the maximum interval of argument). Multipliers were chosen with a minimum number of digits for faster VET computation.

First Digit of Running Product	Range of Argument	Multiplier	$\log_{10}$ of Multiplier
.1	.1-.2	5.5	.74036 26894 94
.2	.2-.3	3.6	.55630 25007 67
.3	.3-.4	2.7	.43136 37641 59
.4	.4-.5	2.2	.34242 26808 22
.5	.5-.6	1.8	.25527 25051 03
.6	.6-.7	1.5	.17609 12590 56
.7	.7-.8	1.3	.11394 33523 07
.8	.8-.9	1.2	.07918 12460 48
.9	.9-1.0	1.1	.04139 26851 58

Approximating polynomials  $P_n(x)$  for the  $\log_{10}(1+x)$  are produced from the Taylor series by Tchebysheff relaxation of the higher order terms and further relaxation to minimize the number of digits in the highest order remaining coefficient so the first multiplication in the polynomial expansion is VET fast. In the range  $0 \leq x \leq .1$ :

For 8-digit accuracy (max error = 32 in 10th decimal position),

$$\log_{10}(1+x) \cong P_8(x) = .4342\ 9394\ x - .2170\ 981\ x^2 + .14327\ x^3 - .09\ x^4$$

For 10-digit accuracy (max error = 68 in 12th decimal position),

$$\log_{10}(1+x) \cong P_{10}(x) = .43429\ 44627\ x - .21714\ 4958\ x^2 + .14466\ 55\ x^3 - .1066\ x^4 + .0683\ x^5$$

Individual variations may be constructed on these lines. For example, a smaller range might be chosen for  $P_8(x)$  so that the 4th order term would be completely eliminated within permissible error limits. This would in turn dictate more and smaller increments in the table.

Example: Find  $\log_{10}(36)$   
 $= \log_{10}(.36 \cdot 10^2)$

$$\begin{array}{r} .36 \\ \underline{2.7} \quad (A_1) \\ .972 \\ \underline{1.1} \quad (A_2) \\ 1.0692 \end{array} \quad \begin{array}{l} -\log A_1 = -.43136\ 37642 \\ -\log A_2 = -.04139\ 26852 \\ P_8(.0692) = .02905\ 89481 \\ E = 2. \end{array}$$

Note: The values of  $A_i$  must be used as multipliers, not multiplicands, else the VET advantage will be lost.

$$\text{Summing, } \log_{10}(36) \cong 1.55630\ 24987$$

*Binary Machines*

The most feasible and general method of logarithm computation for binary computers seems to be to compute  $\log_2$  initially and then convert to either  $\log_e$  or  $\log_{10}$  by a single end multiplication in floating point. Since in binary floating point

$$.5_{\text{decimal}} \leq M = .1\boxed{\text{xxx}}\text{xxxx} \dots < 1.0_{\text{decimal}}, \quad \text{an 8-position table}$$

was chosen to represent the 3 enclosed bits above. Address formation may thus be done by extraction or some other direct method. The basic consideration in the determination of a value for  $\Delta$  was the elimination of the 4th order term by Tchebysheff relaxation, still maintaining an error less than 1 in the 27th bit for the  $\log_e$ . (This was chosen for the 27 bit mantissa of a floating point number of the 709 and may be varied for other machines.)  $\Delta = .0390625_{\text{dec}}$  was chosen for the table below, being the largest 6 bit binary fraction less than the  $\Delta$  which meets the above specifications.

Extracted Bits	Range (octal)	Multiplier (octal)	Log <sub>2</sub> of Multiplier (octal)
000	.40-.44	1.66	.703 723 450 561
001	.44-.50	1.52	.564 543 765 447
010	.50-.54	1.40	.453 400 321 640
011	.54-.60	1.30	.353 165 174 015
100	.60-.64	1.20	.244 647 411 363
101	.64-.70	1.14	.176 740 553 440
110	.70-.74	1.06	.102 142 610 633
111	.74-1.0	1.02	.026 565 575 654

In the range  $0 \leq x \leq .0390625$ , for 27 bit accuracy (max error = 1 in 27th bit for  $\log_e$ , .5 in 27th bit for  $\log_{10}$ ):

$$\log_2(1+x) \cong P_8(x) = 1.342\ 520\ 342\ x - .560\ 7625\ x^2 + .35\ x^3 \text{ (octal coeff.)}$$

Example: Find  $\log_e(110.1101)$

In octal,

$$= \log_e(.1101101 \cdot 2^3) \quad \begin{array}{r} .664 \\ 1.14 \\ \hline 1.0056 \end{array} \quad (A_1) \quad \begin{array}{r} -\log A_1 = -.176\ 740\ 553\ 440 \\ P_8(.0056) = .010\ 177\ 475\ 366 \\ \hline E = 3. \end{array}$$

To verify,  $\log_2(110.1101) = 2.76818\ 43186_{\text{dec}}$   
 $\log_e(110.1101) = 1.91875\ 916$

$$\log_2(110.1101) = 2.611\ 236\ 721\ 726$$

**GENERAL PURPOSE PROGRAMMING SYSTEMS\***

ANATOL W. HOLT, UNIVAC Applications Research Center, Philadelphia, Pa.

In these remarks, I do not propose to describe an automatic coding system. My aim is, rather, to present an approach to automatic coding or, if you prefer, to present a motivation which has guided us in the development of the three actual systems for Remington Rand computers. The earliest of these three systems, developed for the UNIVAC I, has been known by the name Generalized Programming. (This system was recently renamed FLEXIMATIC by the Remington Rand Sales Office.) In operation for approximately two years, GP is currently employed at a considerable number of commercial installations. The second system, Generalized Programming Extended (GPX), is a new version of FLEXIMATIC for the UNIVAC II; it has considerably increased powers in comparison with its ancestor.

\* Based on a talk delivered at the ACM conference in Houston, Texas, June 19-21, 1957.