For roots other than the principal root the program was modified by replacing $m^k \pmod{p}$ by $\rho^\nu m^k \pmod{p}$, $(\nu = 1, 2, \cdots, k - 1)$, where ρ is a primitive root of p. In this case the primitive roots subroutine was incorporated into the routine and the k sums were computed abreast and added in 12 as a check.

EMMA LEHMER

Berkeley, California

1. G. B. MATHEWS, *Theory of Numbers*, Cambridge, 1892, p. 202–212.
2. *Ibid.*, p. 223–228.
3. J. VON NEUMANN & H. H. GOLDSTINE, "A numerical study of a conjecture by Kummer," *MTAC*, v. 7, 1953, p. 133–134.
4. G. BEYER, "Über eine Klasseneinteilung aller kubischen Restcharaktere," *Abh. Math. Seminar*, Univ. Hamburg, v. 19, 1954, p. 115–116.
5. H. HASSE, *Vorlesungen über Zahlentheorie*, Berlin, 1950, p. 457–466.
6. P. BACHMANN, *Die Lehre von der Kreistheilung und ihre Beziehungen zur Zahlentheorie*, Teubner, Leipzig, 1872, p. 228–230.
7. A. J. C. CUNNINGHAM, *Quadratic Partitions*, London, 1904.
8. EDMUND LANDAU, "Über die Verteilung der Primideale in den Idealklassen eines algebraischen Zahlkörpers," *Math. Annalen*, v. 63, 1906–7, p. 204.
9. EMMA LEHMER, "The quintic character of 2 and 3," *Duke Math. J.*, v. 18, 1951, p. 11–18.

A Method for Computing Certain Inverse Functions

A method will be demonstrated for computing the inverse of certain functions. The method is applicable to the computation of logarithms and inverse trigonometric functions. It makes use of the binary expansion of real numbers and is, therefore, particularly suitable for use in automatic digital computing machines which use the binary number system. It is not recommended for hand computing.

1. The Method. Let $f(x)$ be a function which satisfies the following conditions,

(i) $f(x)$ is continuous and monotone on an interval $(0, a]$ (including a, but not including 0),

(ii) $f(a/2)$ is known,

(iii) $f(2x)$ and $f(2x - a)$ can be computed when $f(x)$ is known.

$(0, a]$ is taken to mean $[a, 0)$ if a is negative. Also, the symbol $(f(0), f(a)]$ will be taken to mean $(f(0, +), f(a)]$ $(f(0, -), f(a)]$, $[f(a), f(0, +))$ or $[f(a), f(0, -))$, whichever is appropriate.

Examples are:

$$\text{(a)}\quad f(x) = 2^x \qquad a = -1$$
$$f(2x) = (f(x))^2 \qquad f(2x + 1) = 2(f(x))^2$$

$$\text{(b)}\quad f(x) = \cos x \qquad a = \pi$$
$$f(2x) = 2(f(x))^2 - 1 \qquad f(2x - \pi) = 1 - 2(f(x))^2$$

Let $y \in (f(0), f(a)]$, and let it be required to compute $f^{-1}(y)$, that is, to find x such that $f(x) = y$. The existence and uniqueness of such an x in $(0, a]$ are guaranteed by condition (i). Let $w = x/a$. Then w is in the interval $(0, 1]$. It follows that w has an expansion as a binary number,

$$w = \sum_{n=1}^{\infty} u_n \cdot 2^{-n}$$

where $u_n = 0$ or 1 for each n. Where two such expansions exist, choose the one in which there are infinitely many u_n's equal to 1. For each non-negative integer k, let

$$w_k = \sum_{n=1}^{\infty} u_{n+k} \cdot 2^{-n},$$

$$x_k = w_k a.$$

Then $x_0 = x$ and each x_k is in the interval $(0, a]$. Also, for each k,

$$x_k = \tfrac{1}{2} u_{k+1} a + \tfrac{1}{2} x_{k+1}.$$

Thus,

$$u_{k+1} = \begin{cases} 0 & \text{if} \quad x_k \in (0, a/2] \\ 1 & \text{if} \quad x_k \in (a/2, a]. \end{cases}$$

This is equivalent to

$$u_{k+1} = \begin{cases} 0 & \text{if} \quad f(x_k) \in (f(0), f(a/2)] \\ 1 & \text{if} \quad f(x_k) \in (f(a/2), f(a)]. \end{cases}$$

By condition (iii), it follows that $f(x_{k+1})$ can be computed when $f(x_k)$ is known. Since $f(x_0) = y$ is given, an inductive procedure is established for determining the u_k's. These determine w, and the required number x is wa. The procedure will be illustrated in three examples:

Example 1. Compute $\log_2 (.6)$. In this example,

$$\begin{aligned} f(x) &= 2^x & a &= -1 \\ f(2x) &= (f(x))^2 & f(2x+1) &= 2(f(x))^2 \\ (f(0), f(a/2)] &= (1, \sqrt{.5}] & (f(a/2), f(a)] &= (\sqrt{.5}, .5]. \end{aligned}$$

Since we have to square each $f(x_k)$ anyway, we can test whether $f(x_k)$ is in $(0, f(a/2)]$ or $(f(a/2), f(a)]$ by squaring and comparing with .5. Thus, each step consists of squaring $f(x_k)$, comparing with .5 and recording $u_{k+1} = 1$ if $(f(x_k))^2 < .5$, and $u_{k+1} = 0$ if $(f(x_k))^2 \geq .5$. Then we record $f(x_{k+1})$ which is either $(f(x_k))^2$ or $2(f(x_k))^2$, whichever is between .5 and 1.

k	$f(x_k)$	$(f(x_k))^2$	u_{k+1}
0	.6	.36	1
1	.72	.5184	0
2	.5184	.26873 856	1
3	.53747 712	.28888 16545 23494 4	1
4	.57776 33090 46988	$\cdots$	

Thus $w = .1011 \cdots$ (binary notation) and $\log_2 .6 = aw = -.1011 \cdots$ (binary) $= -(11/16 + \epsilon) \quad 0 < \epsilon < 1/16$.

Example 2. Compute arc cos 0.5. In this example,

$$\begin{aligned} f(x) &= \cos x & a &= \pi \\ f(2x) &= 2(f(x))^2 - 1 & f(2x-\pi) &= 1 - 2(f(x))^2 \\ (f(0), f(a/2)] &= (1, 0] & (f(a/2), f(a)] &= (0, -1] \end{aligned}$$

At each step we record $u_{k+1} = 0$ if $f(x_k) \geq 0$, $u_{k+1} = 1$ if $f(x_k) < 0$. Then $f(x_{k+1}) = \pm (2(f(x_k))^2 - 1)$, the sign $\pm$ is that of $f(x_k)$.

k	$f(x_k)$	u_{k+1}	$2(f(x_k))^2 - 1$
0	.5	0	−.5
1	−.5	1	−.5
2	.5	0	−.5
3	−.5	1	−.5

$$\begin{aligned} w &= .0101 \cdots \text{ (binary)}, \\ w &= 1/3 \\ \text{arc cos } 0.5 &= \pi/3. \end{aligned}$$

Example 3. Compute arc cos 0.2.

k	$f(x_k)$	u_{k+1}	$2(f(x_k))^2 - 1$
0	.2	0	−.92
1	−.92	1	.6928
2	−.6928	1	−.04005 632
3	.04005 632	0	$\cdots$

$$\begin{aligned} w &= .0110 \cdots \text{ (binary)} \\ w &= 3/8 + \epsilon, 0 < \epsilon < 1/16 \\ \text{arc cos } 0.2 &= w\pi = 3\pi/8 + \epsilon\pi, \quad 0 < \epsilon\pi < 0.1964. \end{aligned}$$

2. Error Analysis. Condition (iii) is seldom realized in practice, for required quantities are usually "computable" only approximately. Even so simple a process as squaring cannot be repeated indefinitely without eventually "rounding off." A more practical condition than (iii) is

(iiia) there exist small positive numbers δ and ϵ, such that when $f(x)$ is given, ($x \in (0, a]$) numbers can be computed which differ from $f(2x)$ and $f(2x - a)$, respectively, by less than ϵ, and such that for any pair x, $x' \in (0, a]$, $|f(x) - f(x')| < \epsilon$ implies $|x - x'| < \delta$.

Hereafter ϵ will designate the *maximum roundoff error.* Suppose that the calculation of x proceeds as indicated above, except that at each stage the true value of $f(x_n)$ is replaced by an approximation y_n, computed in accordance with (iiia). Then for each n,

$$|y_n - f(2f^{-1}(y_{n-1}) - au_n)| < \epsilon$$

and this implies

$$|f^{-1}(y_n) - 2f^{-1}(y_{n-1}) + au_n| < \delta.$$

Now, let

$$z_0 = x$$

and for each positive integer N, let

$$z_N = a \sum_{n=1}^{N} u_n \cdot 2^{-n} + 2^{-N} f^{-1}(y_N).$$

That is, z_N consists of the first N binary digits of w plus an error term. Then, for each n,

$$z_n - z_{n-1} = 2^{-n}(f^{-1}(y_n) - 2f^{-1}(y_{n-1}) + au_n)$$
$$|z_n - z_{n-1}| < 2^{-n}\delta.$$

Thus for every N

$$|z_N - x| = |z_N - z_0|$$
$$= \left|\sum_{n=1}^{N} (z_n - z_{n-1})\right| < \sum_{n=1}^{N} 2^{-n}\delta < \delta.$$

Thus each z_n differs from the required x by less than δ. If the process could be carried on indefinitely, therefore, roundoff errors would introduce a total error no greater than δ. Since δ is a function of the computation facilities available and independent of the number of steps executed, it will be designated as the *maximum residual error*. It is interesting to note that the maximum residual error is numerically equal to the maximum error that could result from rounding the input number, y.

If the process is terminated after N steps, the result is an underestimate and should, therefore, be "rounded up" by adding $2^{-N-1}a$. The computed value for x is, then,

$$a\left(\left(\sum_{n=1}^{N} u_n \cdot 2^{-n}\right) + 2^{-N-1}\right).$$

This differs from z_N by

$$2^{-N}(f^{-1}(y_N) - a/2)$$

which is not greater in absolute value than $2^{-N-1}a$. Therefore, $2^{-N-1}|a|$ is designated the *maximum termination error*. The total error is obviously less than the sum of the maximum residual error and the maximum termination error.

In Example 1 above ($\log_2 y$), the maximum residual error is $1/2(2^{\epsilon} - 1)$ where ϵ is the maximum roundoff error. It is less than

$$1/2 \frac{\epsilon \log_e 2}{1 - \epsilon \log_e 2}.$$

The maximum termination error is 2^{-N-1}.

In Examples 2 and 3 above, arc cos y, the maximum residual error is arc cos $(1 - \epsilon)$, which is approximately $\sqrt{2\epsilon}$, and the maximum termination error is $\pi/2^{N+1}$.

3. Programming for Machine Computation. A typical machine program for the computation of $f^{-1}(x)$ by the method of section 1 would be as follows:

(It is assumed that $f(x)$ is an *increasing* function of x. If it is a *decreasing* function, then the relation $>$ should be replaced by $<$ in Step 1.) The symbol C_m means "contents of cell m."

Cell Number	Initial Contents	Representing
1	y	y_n
2	$1/2$	2^{-n-1}
3	0	$\sum_{i=1}^{n} u_i \cdot 2^{-i}$
4	2^{-N}	2^{-N}
5	$f^{-1}(a/2)$	$f^{-1}(a/2)$

Program:

Step 1. If $C_1 > C_5$, proceed to Step 3. If not, proceed to Step 2.
Step 2. Compute $f(2f^{-1}(C_1))$ and store in cell 1. Proceed to Step 5.
Step 3. Compute $f(2f^{-1}(C_1) - a)$ and store in cell 1. Proceed to Step 4.
Step 4. Store $C_2 + C_3$ in cell 3. Proceed to Step 5.
Step 5. Store $1/2C_2$ in cell 2. Proceed to Step 6.
Step 6. If $C_2 < C_4$, proceed to Step 7. If not, proceed to Step 1.
Step 7. Store $C_2 + C_3$ in cell 3. Store $a \cdot C_3$ in cell 3.
The computed value of $f^{-1}(y)$ is C_3.

Of course, the program outlined above can be improved when special properties of $f(x)$ are known. If, for example, $f(x)$ is monotone in the interval $(0, 2a]$ then comparing of $f(x)$ and $f(a/2)$ is equivalent to comparing $f(2x)$ and $f(a)$. (This condition is satisfied by 2^x, but not by $\cos x$.) If in addition, $f(x - a)$ can be computed when $f(x)$ is known, the program outlined above can be modified by placing $f^{-1}(a)$ rather than $f^{-1}(a/2)$ in cell 5, and altering steps 1–3 to read:

Step 1. Compute $f(2f^{-1}(C_1))$ and store in cell 1.
Step 2. If $C_1 > C_5$, proceed to Step 3. If not, proceed to Step 5.
Step 3. Compute $f(f^{-1}(C_1) - a)$ and store in cell 1. Proceed to Step 4.

The modified program has several advantages. For one thing, Step 3 of the modified program is likely to require fewer orders than Step 3 of the original program. In the case of the function $f(x) = 2^x$, for example, Step 3 of the modified program is multiplication by 2, whereas in the original program, it is squaring and multiplying by 2. Another advantage in the case $f(x) = 2^x$ is that $f(a) = 1/2$ is more easily and accurately evaluated than $f(a/2) = \sqrt{1/2}$.

It may be possible to take advantage of special features of the particular machine to be used, by combining certain steps. A program for $\log_2 x$ has been developed for the CRC-102A Computer, which has only four commands in the repeating loop, and only three additional cells referred to in the repeating loop. The repetitive part of the program is shown below, coded in octal, with explanatory notes at the right.

Cell Number	(Initial) Contents				
2000	25	2007	2007	2007	Square y
2001	27	2006	2005	2006	Shift logical
2002	31	2006	2007	2006	Scale factor
2003	37	2006	2003	2000	Test overflow
2004	34	3000	2100	—*	Exit
2005	00	0000	0000	0001	Shift control
2006	37	7777	7777	7776	See text
2007				y	y_n

* Address of next command in main program.

The number in cell 2006 is

$$2^{n-35}\left(1 - \tfrac{1}{2}\sum_{i=1}^{n} u_i \cdot 2^{-i}\right) - 1.$$

4. Other Inverse Trigonometric Functions. Although the method given above is applicable directly to the computation of other inverse trigonometric functions, the functions $f(2f^{-1}(y) - u_n a)$ are in general more complicated than $\pm\,(2y^2 - 1)$. It is advantageous, therefore, to evaluate other inverse trigonometric functions by converting them to inverse cosines, as follows:

$$\text{arc sin}\, y = \pi/2 - \text{arc cos}\, y$$

$$\text{arc tan}\, y = \pm\, 1/2 \text{ arc cos}\left(\frac{1 - y^2}{1 + y^2}\right)$$

(sign is that of y)

$$\text{arc cot}\, y = \pi/2 \pm 1/2 \text{ arc cos}\left(\frac{1 - y^2}{1 + y^2}\right)$$

(sign opposite that of y)

$$\text{arc sec}\, y = \text{arc cos}\,(1/y)$$

$$\text{arc csc}\, y = \frac{\pi}{2} \text{arc cos}\,(1/y).$$

The application to arc sin y provides an interesting algorithm for computing arc sin y, which may be useful even in hand computing. The algorithm is as follows:

1. Iterate the function $2y^2 - 1$, starting with the number y whose arc sin is required.
2. Record the *signs* of the iterates in order.
3. Accumulate the signs; that is, record the "partial products" of the signs in order.
4. Write descending powers of 2 between the signs accumulated.
5. Multiply the series obtained by $\pi/2$.

Example: Compute arc sin $\sqrt{.75}$

1.	$\sqrt{.75}$,	.5,	$-.5$,	$-.5$,	$-.5$	$\cdots$	
2.	$+$	$+$	$-$	$-$	$-$	$\cdots$	
3.	$+$	$+$	$-$	$+$	$-$	$\cdots$	
4.	$+1/2$	$+1/4$	$-1/8$	$+1/16$	$-1/32$	$\cdots$	$= 2/3$

5. $\pi/2 \cdot (2/3) = \pi/3$

   arc sin $\sqrt{.75} = \pi/3$.

5. Comparison with Other Methods. The usefulness of the method described above as compared with other methods depends, of course, on the function to be evaluated and on the features of the machine to be used.

The fact that each iteration yields exactly one binary bit may be an advantage or a disadvantage; a method where error decreases faster than 2^{-n} will converge with fewer iterations than this one. On the other hand, one iteration of this method may consist of fewer commands than an iteration of another method. The logarithm program for the CRC-102A described above has 4 commands per iteration as compared with 14 commands per iteration in another program for logarithm on the same machine. The program for arcsin by the method described above has 9 commands per iteration as compared to 22 for another arcsin program. The fact that each iteration yields exactly one binary bit also simplifies error analysis, for the number of digits of accuracy is exactly one less than the number of digits computed.

D. R. Morrison

SANDIA Corp.
Albuquerque, New Mexico

A Method for Solving Algebraic Equations using an Automatic Computer

Introduction. Many methods have been developed for solving algebraic equations and several of these have been used with automatic computers [1, 2]. Those methods which are most suitable for use with automatic computers are ones which apply to a wide class of equations and which are relatively rapid when the degree of the equation is large. The method described here has been constructed with these considerations in mind and has been programmed for the Illiac computer at the University of Illinois.