TABLES FOR AUTOMATIC COMPUTATION

HERBERT S. WILF
Nuclear Development Corporation of America
White Plains, New York

I. *Introduction:*

Tables in use today generally give, for each of a set of values of a variable x, the corresponding value of the tabulated function f(x). It is historically clear why tables are made in this manner, but the demands of high speed digital computation led the present writer to inquire whether this is indeed the useful way to make a table for an automatic computer.

The normal use of a table of values in digital computation is to store the tabular values, at equally spaced arguments, to compute the value of x in some manner, extract the two table entries which surround x and interpolate linearly for f(x). Since the final product is this interpolated value one is led to ask whether the number $f(x_0 + nh)$ is the "best" number to insert in the n^{th} tabular position if we wish to optimize the accuracy, on the average, of the final interpolated function values. It is immediately clear that it is not, as may be seen most simply by considering a function which always lies above the chord connecting any two points on its graph.

Figures 1 and 2 below illustrate the function f*(x) which is actually used in the computation in the case of first, a function tabulated conventionally and second, tabulated by one of the methods to be described below.

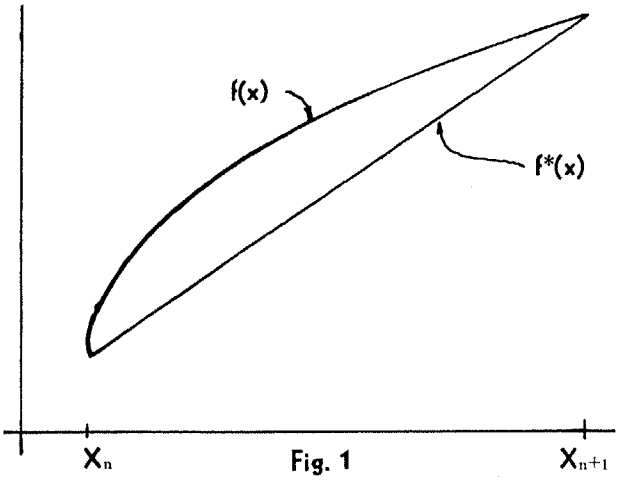

Fig. 1

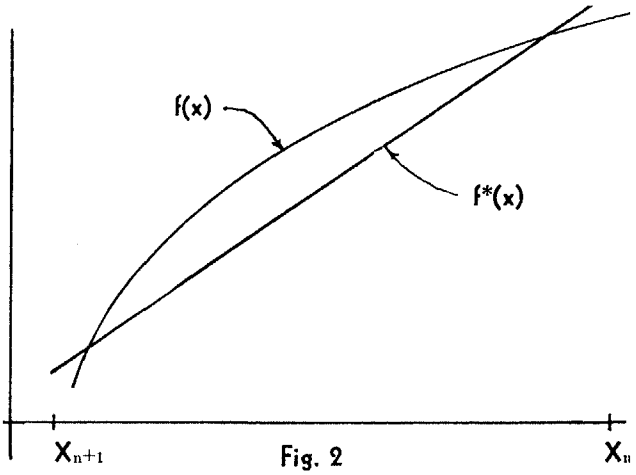

Fig. 2

It is clear that in many cases, one can do considerably better, accuracy-wise, if one drops the constraint, which may be unnecessary in digital computation, that the n^{th} tabular entry be exactly $f(x_0 + nh)$.

II. *Statement of the Problem:*

Formally, we are given a function f(x) and a set of points x_n, where $x_n = nh + x_0$ $(n = 0, 1, \cdots, N)$. Let $g_0, g_1, \cdots, g_N$ denote the set of entries to be determined.

Define $k_n(x)$ by

a) $k_n(x_n) = g_n$

b) $k_n(x_{n+1}) = g_{n+1}$

c) k_n is a straight line.

Finally, let f*(x) denote the function defined for $x_0 \leq x \leq x_N$ which takes the value $k_n(x)$ when x lies between x_n and x_{n+1}. Then f*(x) is the function which will ultimately be used in the computation.

Our problem now is to determine $g_0, g_1, \cdots, g_N$ in such a way that in some sense f*(x) lies as close as possible to f(x).

III. The Least Squares Relative (LSR) Modified Table:

The LSR results from minimizing

$$(1) \qquad \phi(g_0, g_1, \cdots g_N) = \int_{x_0}^{x_N} \left[\frac{f^*(x) - f(x)}{f(x)}\right]^2 dx,$$

where we will assume that $|f(x)| \geq c > 0$ on $[x_0, x_N]$. Writing

$$\phi(g_0, \cdots, g_N) = \sum_{n=0}^{N-1} \int_{x_n}^{x_{n+1}} \left[\frac{k_n(x)}{f(x)} - 1\right]^2 dx,$$

substituting

$$(2) \qquad k_h(x) = g_h + \frac{g_{h+1} - g_h}{h}(x - x_h),$$

and taking partial derivatives of ϕ with respect to each of the g_h, we get $N + 1$ simultaneous linear algebraic equations:

$$(3) \qquad [\alpha_{p-1} - \epsilon_{p-1}]g_{p-1} + [\beta_p + \epsilon_p - 2\alpha_p + \epsilon_{p-1}]g_p + [\alpha_p - \epsilon_p]g_{p+1} = \gamma_p - \delta_p + \delta_{p-1} \qquad (p = 0, 1, \ldots, N)$$

to solve for the g_p, where we have written

$$(4) \qquad \alpha_p = \frac{1}{h}\int_{x_p}^{x_{p+1}} \frac{x - x_p}{f^2(x)} dx \qquad (p = 0, \ldots, N-1)$$

$$(5) \qquad \beta_p = \int_{x_p}^{x_{p+1}} \frac{dx}{f^2(x)} \qquad (p = 0, \ldots, N-1)$$

$$(6) \qquad \gamma_p = \int_{x_p}^{x_{p+1}} \frac{dx}{f(x)} \qquad (p = 0, \ldots, N-1)$$

$$(7) \qquad \delta_p = \frac{1}{h}\int_{x_p}^{x_{p+1}} \frac{x - x_p}{f(x)} dx \qquad (p = 0, \ldots, N-1)$$

$$(8) \qquad \epsilon_p = \frac{1}{h^2}\int_{x_p}^{x_{p+1}} \frac{(x - x_p)^2}{f^2(x)} dx \qquad (p = 0, \ldots, N-1)$$

$$(9) \qquad \alpha_{-1} = \epsilon_{-1} = \delta_{-1} = \beta_N = \epsilon_N = \alpha_N = \gamma_N = \delta_N = 0.$$

Because of the form of Eq. 3 the solution may be obtained readily by writing

$$(10) \qquad \begin{cases} g_p = \lambda_p g_{p+1} + \mu_p \\ g_{N+1} = 0 \end{cases}$$

from which the recurrence relations

$$(11)\qquad \begin{cases} (a)\ \lambda_{-1} = 0 \\ (b)\ \lambda_p = -\,u_p\{u_{p-1}\,\lambda_{p-1} + V_p\}^{-1} \end{cases}$$

$$(12)\qquad \begin{cases} (a)\ \mu_{-1} = 0 \\ (b)\ \mu_p = (C_p - u_{p-1}\,\mu_{p-1})\,(u_{p-1}\,\lambda_{p-1} + V_p)^{-1} \end{cases}$$

follow. Here we have written

$$(13)\qquad u_n = \alpha_n - \epsilon_n$$

$$(14)\qquad V_n = \beta_n + \epsilon_n - 2\alpha_n + \epsilon_{n-1}$$

$$(15)\qquad C_n = \gamma_n - \delta_n + \delta_{n-1}$$

IV. The Least Squares Absolute (LSA) Modified Table:

Here we minimize

$$(16)\qquad \phi(g_0, g_1, g_2, \ldots, g_N) = \int_{x_0}^{x_N} [f^*(x) - f(x)]^2\,dx$$

which leads to the equations

$$(17)\qquad 2g_p + \tfrac{1}{2}g_{p+1} + \tfrac{1}{2}g_{p-1} = \alpha_p - \beta_p + \beta_{p-1}$$

$$(18)\qquad \beta_{-1} = g_{-1} = g_{N+1} = 0 = \alpha_N = \beta_N$$

$$(19)\qquad \alpha_p = \frac{3}{h}\int_{x_p}^{x_{p+1}} f(x)dx$$

$$(20)\qquad \beta_p = \frac{3}{h}\int_{x_p}^{x_{p+1}} f(x)\left(\frac{x - x_p}{h}\right)dx.$$

These can be readily solved by the device previously used.

V. Conclusions:

The criterion of least squares integrated error minimization is not the natural one in numerical approximations. Much preferable would be MMR and MMA for minimax relative and absolute error modified tables. An elegant and useful algorithm for the construction of such tables has so far eluded the author.

Finally, since these tables are almost always the appropriate kind to use in automatic computation, it is to be hoped that they will be prepared for all the standard mathematical functions and with a variety of mesh widths for each.

VI. An Example:

We give below a complete LSR modified table for $f(x) = \sqrt{x}$, $x_n = n + 1$ $(n = 0, \ldots, 9)$.

The tabular values are denoted by $\phi(x)$, and the actual values of $f(x)$ are tabulated for comparison. It will be noted that $\phi(x_n) > f(x_n)$, and that the line $k_n(x)$ cuts $f(x)$ in two points inside of every interval. The maximum relative error from using $\phi(x)$ with linear interpolation is about 1.28%, compared with 1.49% with $f(x)$ and linear interpolation.

Table I

x	$\phi(x)$	f(x)	x	$\phi(x)$	f(x)
1	1.012704	1.000000	6	2.450921	2.449490
2	1.423418	1.414214	7	2.646876	2.645751
3	1.735359	1.732051	8	2.829339	2.828427
4	2.002788	2.000000	9	3.000835	3.000000
5	2.237870	2.236068	10	3.162847	3.162278

A PROGRAMMED BINARY COUNTER FOR THE IBM TYPE 650 CALCULATER

B. C. KENNY and J. A. HUNTER
Westinghouse Electric Corporation, Baltimore, Maryland

In a recent computational job on the IBM 650 (with 653) the authors found it necessary to perform a calculation which depended upon a sequence of n yes-no decisions. It was required to repeat the procedure for all 2^n possible sequences. Since n was less than eleven, it was convenient to express each yes-no sequence by an integer composed of eights and nines. The "Branch on Distributor 8" instruction was then used by the program to interpret the current sequence and thereby control the calculation.

Regarding the sequences as binary numbers with eights representing zeros and nines denoting ones, a binary counter was programmed which enabled the computer to step through all sequences in turn. The pertinent logic of the main program is illustrated in the accompanying block diagram.

Following the diagram a specific coding for the counter is given. It is assumed that this program is stored on the magnetic drum in the cells immediately following the location of the current "binary" number, b. It is also understood that the drum copy of the binary counter routine is read into high speed storage afresh before each advance of b. In this way the contents of cells 9001 and 9004 are preset automatically.

It should be noted that the coding given below carries the implication that $n < 10$. For $n = 10$, the program must be modified because the simple test indicated at cell 9015 becomes impossible and because the scheme for altering the branch instruction in cell 9004 breaks down.

A Specific Coding of the Binary Counter

Core Location	*Number or Instruction*	*Remarks*
9000	bb bbbb bbbb	Binary number, b, to be advanced. 8's and 9's.
9001	00 0000 0001	Adjustable constant for altering b.
9002	01 0000 0000	Constant for altering instruction in 9004.
9003	65 9000 9004	*ENTER* Put b into D and L.
9004	91 9013 9005	Test digit of b. 8 or 9?
9005	16 9001 9006	If 9: Change to 8.
9006	20 9000 9007	Put altered b into 9000.
9007	65 9001 9008	Put constant in 9001 into L.
9008	35 0001 9009	Left shift one place.
9009	20 9001 9010	Replace adjusted constant in 9001.
9010	65 9004 9011	Put test instruction into L.
9011	15 9002 9012	Alter it to test next digit of b.
9012	20 9004 9003	Replace altered test instruction in 9004.
9013	15 9001 9014	If 8: Change to 9.
9014	20 DRUM 9015	Store altered b on drum and in D.
9015	9X PROG PNCH	*EXIT* If $b < 2^n$, return to program. If $b = 2^n$, go to output routine. $X \equiv n + 1 \pmod{10}$.