The method of successive grids for reduction of function storage requirements*

By Roger L. Boyell

A method is analysed for reducing storage requirements of arbitrary but relatively well-behaved functions of any number of variables. The method consists of storing successive digits of the words representing the values of the function according to grids of successively finer meshes. The advantage is illustrated as dependent upon the size of the function table to be stored, being well suited to computers operating with variable word length.

It is frequently required to maintain a large table of values of an arbitrary function, this function being of one or more variables. When operating with computing devices, it is frequently necessary that the size of the bulk store be as small as possible—both in number of entries and in size of each entry (number of digits carried) —even at the expense of computational effort. It may be desirable to maintain the table in two portions, consisting respectively of those relatively few points which lie on a coarse grid, and then of those points that must be interpolated (only differential values stored) by use of a fine grid. If the function to be represented is slowly varying, it may be found that the total size of the table (number of entries times average number of digits per entry) would be further reduced by having a three-level grid—coarse, medium, and fine.

Extending this technique to its limit, one may conceive of a computer, operating under a radix r, in which each succeeding grid is r times as fine as its predecessor. There would be a different mesh grid for each digit in the table entries. Specifically, the coarsest grid would give the first digit, the next finer grid would give the next lower-order digit, etc., until the value of the function at the selected point was built digit by digit. (Note that a point on any given grid is also on any finer grid, and thus all lower order digits will be stored there also.) Obviously this represents a saving in the number of digits stored, compared with storing all the necessary digits for all points on the finest mesh grid, but it requires more computation to accumulate the digits from different grids in order to obtain the value of the function from the table. (The analogy to digit-by-digit analog/digital converters may be apparent.)

Example

To make the process more concrete, take a simple example. Fig. 1 illustrates an arbitrary but monotonic function, plotted on coordinates expressed in a binary number system. The requirement that the function be slowly varying is more explicitly a limitation that the

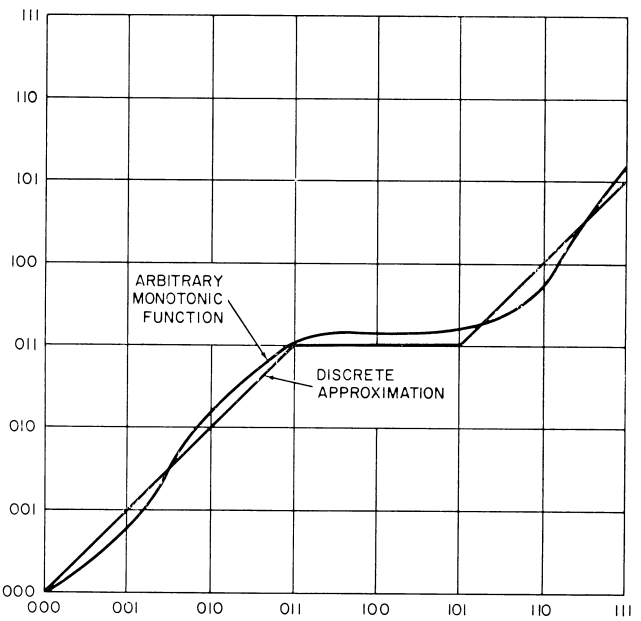

Fig. 1.—Example for radix $r = 2$, digits per word $n = 3$

slope of the function, positive or negative at any point, *cannot exceed one unit of ordinate per unit abscissa*, after appropriate scaling factors have been applied. Indeed, this is a general limitation on the method described herein. (The instantaneous slope must be allowed to equal unity, or else the ordinate could never change as the abscissa advanced away from the origin.) At the halfway point along the abscissa (100 in binary), the first digit of the ordinate is given by the value of the curve at that point. For all tabulated points on this curve of greater abscissa, the first digit can increase by no more than one unit, since the value of the curve can change by no more than three units (011 in binary) and this curve is monotonic increasing. Thus, for all points of greater abscissa, only differential values need be stored with respect to the value at the halfway point, thus reducing the number of digits to be stored for the table.

It is convenient to set up the relations involving the number of digits in a form such as Table 1. Referring

* This work was supported in part by Contract N61339-1025 with the U.S. Naval Training Device Center, and in part by Pennsylvania Research Associates Inc.

Table 1

Derivation of general case; $i = 1, 2, \ldots, n$

	(1) LEVEL OF BREAKDOWN	(2) NUMBER OF POINTS ADDED	(3) LIMIT OF ORDINATE CHANGE	(4) ORDINATE DIGITS PER POINT	(5) TOTAL ORDINATE DIGITS
CASE A	1 2 3	1 2 4 7	4 2 1	3 2 1	3 4 4 11
CASE B	i	2^{i-1}	2^{n-i}	$n+1-i$	$2^{i-1}(n+1-i)$
CASE C	i	2^{i-1}	$\pm 2^{n-1}$	$n+2-i$	$2^{i-1}(n+2-i)$
CASE D	i	$(r-1)r^{i-1}$	$\pm(r-1)r^{n-i}$	$n+2-i$	$(r-1)r^{i-1}(n+2-i)$

radix 2 / radix r — monotonic / not — 3-digit abscissa / n-digit abscissa

first to Case A, column (1) shows the level of breakdown —halfway, quarterway, etc.—along the abscissa of the original curve. In the example of Fig. 1 with eight points along the abscissa, including the origin, there are three levels necessary (corresponding to the three binary digits required to represent the eight-valued abscissa). Column (2) shows the number of points added at each breakdown, or the number of points which can be reached by going to each successive level. Column (3) shows the limit that can be reached on the change of value of the ordinate (in decimal notation) upon going to each level. Column (4) shows the number of digits required to represent the possible change in ordinate for each level. Column (5) is simply the product of Columns (2) and (4), giving the total number of digits required.

In the simple example chosen, it can be seen that eleven digits (in this case, bits) are required to represent the value of the curve at the seven points other than the origin. Using three-digit accuracy in the conventional way, it would of course require twenty-one digit positions to store the same data, representing almost a factor of a half in storage size.

The other cases in Table 1 are successive generalizations, in all of which the index $i = 1, 2, \ldots, n$ where n is the number of digits of precision with which the data must be stored. Case B retains the restriction of monotonic functions, but Case C removes this restriction by adding a digit signifying the algebraic sign of each successive low-order portion of the word representing the value of the ordinate. Finally, Case D illustrates the most general case of an arbitrary radix r. (An entire digit position is assumed to be required for the algebraic sign.)

It is apparent that, with an n-digit word to represent the abscissa, $r^n - 1$ points other than the origin may be represented using n different grids, each of mesh r times that of the preceding. The total number of digits using this "method of successive grids" is the sum of column (5) in Table 1, or for Case D

$$(r-1)\sum_{i=1}^{n} r^{i-1}(n+2-i) = \frac{2r-1}{r-1}(r^n-1) - n$$

On the other hand, storing all ordinates independently (including a digit for the sign) would require $(n+1)(r^n-1)$ digits in storage altogether. The ratio of storage space required is always less than unity. For a binary computer ($r = 2$) it reduces to less than $3/(n+1)$; storing 32,768 entries, ($n = 15$) this upper bound becomes 3/16, extremely close to the actual storage reduction.

For computers operating in the binary system, not only is the method of successive grids rather efficient, but also the selection of points at successive levels is particularly easy. Referring to Fig. 1, it can be seen that the first level of breakdown is represented by the halfway abscissa (100 in binary). The next level of breakdown is represented by the abscissa X10, where X is 0 or 1 respectively for the 1/4 and 3/4 points along the abscissa. In general then, for breakdown level i there are 2^{i-1} points on that level represented by that number of combinations of $i-1$ bits at the beginning of the abscissa word, with the rest of abscissa word being of the form 1000.... For computers operating in number systems other than binary, a similarity exists. For each breakdown level i, the first $i-1$ digits are allowed to run through all possible values, but the next digit is restricted to non-zero values, thus producing the $(r-1)r^{i-1}$ points for that level.

Conclusion

The advantage of the method of successive grids, when it is applicable, is greater for tables of functions of more than one variable; a saving of a factor of two or three in each dimension can be quite useful. For a two-dimensional table, the computer must be programmed to add successive digits for the value of a function at any point from the "lower left" corners of the interstices, including the point of successively finer grids.