J. E. Meggitt

Digit-by-Digit Methods for Polynomials

Abstract: This paper presents a general system configuration for an arithmetic unit of a computer, which is used to solve polynomial problems efficiently. The technique is based on a digit-by-digit computation of the coefficients of the given polynomial, after the origin has been displaced systematically. Compared with standard techniques, the new scheme, closely allied with Horner's method, is similar in efficiency for polynomial evaluation and is superior for locating roots. The fact that the computed coefficients are related to the derivatives permits the systematic location of all real roots of a real polynomial.

Introduction

This paper describes some methods for manipulating polynomials and suggests some computer hardware which is very useful for this purpose. The technique echoes two current trends in computer design. First, the user is provided with a powerful instruction set; and second, the designer wishes to exploit a small high-speed memory in his design, this being now technically feasible. Therefore, there is a need for simple, efficient algorithms that perform powerful operations and make use of a small, high-speed store.

Hitherto the most complicated macro operations implemented in hardware have been (with certain exceptions) multiplication and division. To continue to use these as the basic building blocks for more complex operations, however, does not take advantage of modern computer technology.

Some thought has already been given to constructing novel algorithms for generating the elementary functions.[1] The present paper concerns methods for solving polynomial problems, the principal one being the extraction of roots. For the future there is work to be done finding algorithms for matrix problems and for linear programming.

The ultimate objective is to find a family of algorithms for solving a whole range of mathematical problems. These algorithms will be more complicated than the basic ones currently implemented in hardware but also more efficient.

The first part of this paper shows how a function $f(x)$ may be evaluated for given x, where $f(x)$ is a polynomial specified by its coefficients. This is done in a digit-by-digit way and is suitable for arithmetic performed in any radix R. This means that if

$$x = \sum_{i=0} q_i R^{-i}, \tag{1}$$

$f(x)$ is evaluated successively for $x = 1, 2, \cdots q_0$; $x = q_0 + R^{-1}, q_0 + 2R^{-1}, \cdots, q_0 + q_1 R^{-1}; \cdots$. Each evaluation is made in a simple way from information that was generated at the previous step.

The entire calculation of $f(x)$ involves a set of additions with shifts, which, of course, is what is involved if $f(x)$ is evaluated conventionally using a multiplier. The only difference is the order in which these additions are carried out. Their number is the same. Hence the digit-by-digit method is as efficient as (but no more efficient than) the conventional one. Its significance lies in the fact that it may be reversed, successive digits of x being found in such a way that $f(x)$ is driven towards zero. In this way roots of $f(x)$ are found. If the polynomial is chosen to be $ax - b$, then the method described is no more than the long division of b by a; if the polynomial is $x^2 - b$, then this method is the conventional square root process for finding $\sqrt{b}$.

The method extends to the case where $f(x)$ and x are complex, so that complex polynomials can be evaluated for complex arguments. Furthermore the method may be reversed and complex roots found. The method also enables polynomials in more than one variable to be evaluated.

The virtue of the method, other than its inherent simplicity, is the high speed with which roots may be

found. The amount of computation involved in finding a root is approximately that needed for calculating $f(x)$ for given x, and this should be contrasted with the amount of work done, say, in iterating Newton's method.

The second part of the paper shows how, using information that is generated at each stage of the calculation for no additional cost, *all the real roots of a real polynomial may be found systematically.* This procedure is a virtue possessed by none of the conventional methods. It will transpire that the method of calculating $f(x)$ finds not only $f(x)$, but all the derivatives of $f(x)$ at each point x that is considered in the digit-by-digit generation. In particular the signs of $f(x)$ and of all its derivatives are available, and it is intuitively clear that this information suffices to specify where x lies in relation to the roots of $f(x)$. It is not surprising, therefore, that an algorithm exists whereby the digits of x can be generated according to the signs of $f(x)$ and its derivatives, in such a way that all the real roots of $f(x)$ are found successively.

It is plausible to expect this to extend to the extraction of complex roots. Indeed such an extension exists and this will be presented in a future paper.

The third section of the paper shows how $f(x)$ may be calculated with ease where $f(x)$ is a polynomial function specified not by its coefficients, but by its differences. The algorithm established is effective only in binary arithmetic. It is useful where it is required to fit a polynomial curve through a set of given points. It is another digit-by-digit method and may be reversed to find the roots of $f(x)$.

Evaluation of $f(x)$ for given x

It is supposed that $f(x)$ is a polynomial of degree n and that the $n+1$ numbers $A_r(x_0)$ are known, where

$$f(x) \equiv \sum_{r=0}^{n} A_r(x_0)(x - x_0)^r. \tag{2}$$

The first objective is to define a set of simple operations that produce the numbers $A_r(x_0 + a)$ in terms of the $A_r(x_0)$, where a is an arbitrary power of the radix R in which operations are performed; that is, it is required to find the new coefficients of the polynomial when the origin of x is moved a distance a. This, of course, is what is done when Horner's method is used, though the distance a is then not in general restricted to being a power of the radix. If this set of operations is obtained, it may be employed as follows. Suppose that the $n+1$ coefficients $A_r(0)$ are given initially and that

$$x = \sum_{i=0} q_i R^{-i}. \tag{3}$$

The origin may then be moved in successive steps from 0 to 1, 2, $\cdots$, q_0, and then to $q_0 + R^{-1}$, $q_0 + 2R^{-1}$, $\cdots$ $q_0 + q_1 R^{-1}$, $\cdots$ and ultimately to x.

When this is done, it is clear from (2) that

$$f(x) = A_0(x). \tag{4}$$

Moreover, at each point x_0, the numbers $A_r(x_0)$ are proportional to the derivatives of $f(x)$ at that point, and so the signs of the numbers $A_r(x_0)$ are exactly the signs of the derivatives. This fact will be exploited at a later stage.

$$\begin{aligned} \text{Now } f(x) &\equiv \sum_{r=0}^{n} A_r(x_0)[(x - x_0 - a) + a]^r \\ &\equiv \sum_{r=0}^{n} \sum_{s=0}^{r} A_r(x_0) C_s^r a^{r-s} (x - x_0 - a)^s \\ &\equiv \sum_{s=0}^{n} A_s(x_0 + a)(x - x_0 - a)^s \end{aligned}$$

by definition.

Thus, equating the coefficients of $(x - x_0 - a)^s$ gives

$$A_s(x_0 + a) = \sum_{r=s}^{n} C_s^r a^{r-s} A_r(x_0) \tag{5}$$

and this is the required rule.

The numbers $A_s(x_0 + a)$ can be generated in a very simple way from the numbers $A_r(x_0)$ by means of the binomial tree in Fig. 1a. The symbols $\oplus$ indicate word adders, and in the Figure it is assumed that every diagonal segment between two adders contains a multiplication by a, i.e., a shift, since a is chosen to be a power of the radix. The numbers $A_r(x_0)$ are fed in on the left, and the numbers $A_s(x_0 + a)$ appear on the right. Figure 1b lists the successive adder outputs for the case $n = 3$.

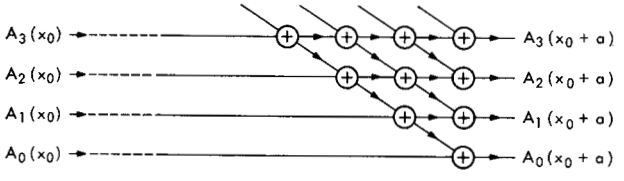

Figure 1a **Binomial adding tree.**

Figure 1b **Adder outputs.**

A_3	A_3	A_3	A_3
	$A_2 + aA_3$	$A_2 + 2aA_3$	$A_2 + 3aA_3$
		$A_1 + aA_2 + a^2A_3$	$A_1 + 2aA_2 + 3a^2A_3$
			$A_0 + aA_1 + a^2A_2 + a^3A_3$

A way of exploiting these results is to build in hardware a system consisting of $n + 1$ registers and a parallel adding tree (as shown in Fig. 1a) whose output is fed back into the registers. Initially the registers are loaded with the $n + 1$ given coefficients $A_r(0)$, and x is assumed to have the value in (3). The contents of the registers are cycled through the tree q_0 times with the shift set at ZERO. After this they are cycled q_1 times with the shift set at ONE (that is, $a = R^{-1}$) and so on. After the contents have been cycled Σq_i times, the registers will contain $f(x)$ and its derivatives $(1/r!)\, f^{(r)}(x)$, $r = 0, 1, \cdots, n$.

The tree may be a parallel tree as suggested, consisting of $n(n + 1)/2$ adders exactly as in Fig. 1a, or it may be serialized in any convenient way. This, for example, may be done by regarding the diagram in Fig. 1a as split into n separate columns and forming the sums in each column during one add time, there being n adders.

It may be inconvenient to perform shifts within the tree as indicated. If this is the case, these shifts may be avoided by replacing the numbers $A_r(x_0)$ by $a^r A_r(x_0)$ so that a relation like (5) is obtained but with the factor a^{r-s} omitted. It will then be necessary to shift $a^r A_r(x)$ r digit places right after each digit q_i of x has been processed, since when this happens the value of a is changed from R^{-i} to R^{-i-1}, which implies a multiplication by R^{-r}.

This procedure, therefore, shows how $f(x)$ may be calculated. It should be observed that the basis of the method is the simple linear relationship (5). Multiples of the numbers $A_r(x_0)$ and linear combinations of them will satisfy other linear relations. If any of them have a simplicity comparable to that in (5), they may be considered as the basis of a method for finding $f(x)$.

One such scheme is obtained by writing

$$A_r(x_0) = C_r^n B_{n-r}(x_0) \tag{6}$$

in (5), n being the degree of the polynomial.

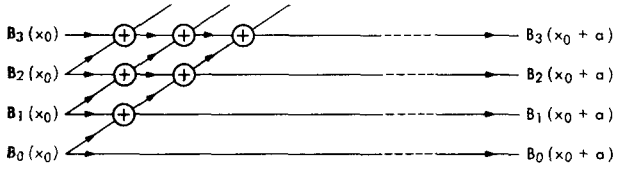

Figure 2a **Alternative binomial adding tree.**

Figure 2b **Adder outputs.**

This leads at once to

$$B_s(x_0 + a) = \sum_{r=0}^{s} C_r^s a^{s-r} B_r(x_0). \tag{7}$$

The numbers $B_s(x_0 + a)$ may therefore be obtained from the numbers $B_r(x_0)$ by means of the tree shown in Fig. 2a, which should be contrasted with that shown in Fig. 1a. Again a shift is assumed in each diagonal segment.

Figure 2b lists the successive adder outputs for the case $n = 3$.

From a practical point of view there seems little to choose between this scheme and the previous one. Of course in the second scheme, the registers must initially be loaded with $B_r(0)$. These are defined by

$$f(x) \equiv \sum_{r=0}^{n} B_r(0) C_r^n x^{n-r}, \tag{8}$$

and the numbers $(r!/n!) f^{(n-r)}(x)$ are produced. Whether it is convenient to work with the B's or the A's depends on how the problem is presented.

•*Scaling*

It is assumed that x has the form (3) with $q_0 \neq 0$ so that $1 \leq |x| < R$. The value x can always be brought to lie in this range by scaling it by a power of R, which implies scaling the coefficients of the polynomial by powers of R. When this is done, the coefficients should have roughly equal precisions if the problem is formulated properly, since $f(x)$ is formed by adding together the various coefficients with roughly equal weights. Therefore, when the problem is scaled in this way, it is sensible to hold the coefficients as fixed point numbers with their radix points aligned, in registers all of the same size.

If the problem were presented with the coefficients as floating point numbers, then there would be a preliminary scaling and fixing step.

• *Error analysis of process for forming f(x)*

For definiteness it will be assumed that the scheme shown in Fig. 1a is employed with the shifts occurring within the binomial tree. The number $f(x)$ will be obtained accurately if registers are sufficiently long to hold the shifted numbers. If the coefficients in $f(x)$ are given initially as integers, and if x is a p-digit number of the form (3), then clearly $A_r(x)$ will be calculated as a number with $r(p - 1)$ fractional digits, owing to the shifts. Normally such accuracy is

B_3	$B_3 + aB_2$	$B_3 + 2aB_2 + a^2B_1$	$B_3 + 3aB_0 + 3a^2B_1 + a^3B_0$
B_2	$B_2 + aB_1$	$B_2 + 2aB_1 + a^2B_0$	
B_1	$B_1 + aB_0$		
B_0			

neither justified nor required and so at each stage the fractional digits of A_r are dropped after a rounding. The magnitude of the resulting error in the calculated value of $f(x)$ must be found in order to justify allowing it.

The analogue of this occurs if $f(x)$ is evaluated by repeated multiplication, since products are in general truncated so that they may serve as factors in further products.

The dropping of fractional digits causes errors in the A_r to occur on each cycle through the binomial tree. On one such cycle the value of A_r is altered due to $r + 1$ rounded additions. Thus the dropping of fractional digits will cause a maximal error in A_r of $(r + 1)/2 (r = 0, 1, \cdots, n - 1)$, the factor $1/2$ occurring because of rounding. The consequence of each such set of errors is to produce a polynomial which is the sum of the desired polynomial and an additional polynomial bounded by

$$\sum_{r=0}^{n-1} (r + 1)x^{*r}/2, \tag{9}$$

where x^* is that part of x which remains to be processed when this set of errors is introduced.

Now when the digit q_i of x is processed, the worst case value of x^* is $(R - 1)R^{-i}$ on the first cycle; $(R - 2)R^{-i}$ on the second; $\cdots$ and R^{-i} on the last, assuming that q_i has the worst value, $R - 1$.

The entire maximal error E in the computed value of $f(x)$ may therefore be calculated by summing over all the maximal errors introduced at different stages. This gives

$$E = 1/2 \sum_{i=1}^{p-1} \sum_{t=0}^{R-1} \sum_{r=0}^{n-1} (r + 1)(tR^{-i})^r. \tag{10}$$

Now

$$\sum_{t=0}^{R-1} t^r < R^{r+1}/(r + 1)$$

so that

$$E < R/2 \sum_{i=1}^{p-1} \sum_{r=0}^{n-1} R^{(1-i)r}$$

Hence the bound

$$E < R/2 \sum_{k=1}^{p-2} \sum_{r=0}^{n-1} R^{-kr} \tag{11}$$

is obtained. This estimate for E may be evaluated exactly, but it is clear that E is of the order $(R(p + n)/2$. Such an error is entirely reasonable and shows how many additional digits should be carried in the calculation to ensure accuracy.

It should be observed that errors arise predominately from the repeated addition of the shifted A_1 to A_0. The errors in the other numbers A_r do not influence $f(x)$ appreciably since, for large p, the A_r are effectively shifted many places before being added to form $f(x)$.

Another way of stating this is to say that as the calculation progresses, the A_r, for large r, are kept to an accuracy that is more than that required. This suggests that it would be possible to ignore the digits of A_r that have small significance, or even to ignore the A_r altogether at some stage in the calculation, and still to maintain an acceptable accuracy. It is attractive to do this if the hardware is such that the time taken to cycle the adding tree depends on the number of additions that need to be done in it, because an increase in speed is then possible.

To be precise, suppose that A_r is set to ZERO immediately after q_i has been processed. This will cause an error in the computed value of $f(x)$ of

$$|A_r x^{*r}|, \tag{12}$$

where x^* is again that part of x which is left to be processed. Now if the numbers A_r are constrained to be integers lying like x in p-digit registers, then $|A_r| < R^p$. (The problem is assumed to be scaled so that this is so.) Furthermore the worst case value of x^* at the stage when q_i has been processed is R^{-i}. The dropping of A_r, therefore, causes a maximal error of R^{p-ir}. It is arranged, therefore, to omit A_r from the calculation as soon as

$$j \geq p/r \tag{13}$$

so that the resulting error is less than unity.

Errors arise from this cause as the A_r are dropped for $r = n, n - 1, \cdots, 2$, and clearly this gives rise to an additional maximal error of $n - 1$, which is again tolerable. The conclusion is that this truncated procedure also has an acceptable accuracy.

- *Register overflows*

The numbers A_r are contained in p-digit registers and overflow must be avoided. This may be ensured by a suitable initial scaling, for if an upper bound is placed on each coefficient of $f(x)$ and if a bound is placed on $|x|$, then it is possible to place bounds on the derivatives, and thus on the numbers $A_r(x)$, and thereby to ensure that they will fit into p-digit registers.

This scheme may result in the waste of register space because of its pessimism. An alternative one is to arrange for the contents of all the registers to float together, so that they move right whenever one of them overflows and move left whenever that is possible. In this way a normalized floating point calculation is performed.

- *Speed of calculation*

The speed of calculation depends on the system used for its implementation. The parallel scheme is clearly very fast. However, it is interesting to consider a completely serial system, in which there is just one adder that is shared, and to compare the calculation time with that for a conventional repeated-multiplication process.

If no truncation is employed, each cycle through the binomial tree takes $n(n + 1)/2$ add times, and on the average the tree will be cycled $Rp/2$ times. The total time is therefore

$$Rpn(n + 1)/4. \tag{14}$$

If the truncated scheme is used where, for comparison purposes, it is assumed

$$x = \sum_{j=1}^{p+1} q_j R^{-j}, \tag{15}$$

the r additions used to form A_{r-1} have to be made only until

$$j = p/r \tag{16}$$

(or for slightly longer if p is not divisible by r). This set of r additions is made on the average only $(p/r)(R/2)$ times. The total average calculation time is obtained by summing these times for all r, and this gives a total time

$$Rpn/2. \tag{17}$$

If $f(x)$ were evaluated by repeated truncated multiplication, there would be n consecutive multiplications, each one involving $Rp/2$ word additions on the average, which would give exactly (17) as the calculation time. Thus the truncated version of this method proposed for calculating $f(x)$ is as fast as the conventional one. This is not surprising since $f(x)$ is formed essentially in a series of additions, the number of which is determined solely by the accuracy to which $f(x)$ is required.

• *Example of method*

A typical calculation is shown in Table 1. In this example a cubic function

$$f(x) \equiv 3127x^3 - 3759x^2 \tag{18}$$

is evaluated at $x = 1.203$, decimal arithmetic being used. No truncation is employed. Each cycle of the tree, which is that in Fig. 1a with $n = 3$, is put within vertical lines. The outputs of the adders are shown explicitly as in Fig. 1b.

Table 1 **Typical calculation, showing evaluation of a cubic function.**

$f(x) \equiv 3127x^3 - 3759x^2$

x	0.000				1.000				1.100
$f'''(x)/6 = A_3$	3127	3127	3127	3127	3127	3127	3127	3127	3127
$f''(x)/2 = A_2$	−3759		−0632	2495	5622		5935	6249	6561
$f'(x) = A_1$	0000			−0632	1863			2457	3082
$f(x) = A_0$	0000				−0632				−0386
Shift used					0				1
					1.200				1.201
Continued		3127	3127	3127	3127	3127	3127	3127	3127
			6874	7187	7500		7503	7506	7509
				3769	4488			4496	4504
					−0009				−0005
					1				3
					1.202				1.203
Continued		3127	3127	3127	3127	3127	3127	3127	3127
			7512	7515	7518		7521	7524	7527
				4512	4520			4528	4536
					0000				+0005
					3				3

This calculation shows $f(1.203) = +0005$.

• *Complex polynomials; polynomials in more than one variable*

The method of evaluating a real polynomial function of one real variable generalizes very easily. For a complex polynomial in a complex variable $z = x + iy$, Eq. (2) becomes

$$f(z) = \sum_{r=0}^{n} A_r(z_0)(z - z_0)^r, \tag{19}$$

where the A_r are complex numbers.

The origin z_0 may be moved by a, which is now in general complex, and this gives rise to a set of numbers $A_r(z_0 + a)$. These may be calculated by means of the binomial tree shown in Fig. 1, where it is understood that all numbers are complex and that diagonal segments of the tree contain a multiplication by a.

To evaluate $f(z)$ for given x and y, the origin is moved to z by making increments which are powers of the radix in magnitude and which lie in the x and y directions. It does not matter if the incrementing in the x direction is interleaved with the incrementing in the y direction.

Complex numbers within the tree are represented by their real and imaginary parts, so that additions in the tree are achieved by adding together these real and imaginary parts separately.

For an increment in the x direction, a has the value R^{-k}, $k = 0, 1, 2 \cdots$, whereas for an increment in the y direction a has the value iR^{-k}. The multiplication by R^{-k} is achieved by shifting the real and imaginary parts k places. In the latter case the additional multiplication by i is made by interchanging real and imaginary parts and altering a sign.

With these generalizations, the complex theory is as straightforward as the real theory.

For a polynomial $f(x, y)$ in two variables, it is supposed that

$$f(x, y) \equiv \sum_{r,t=0}^{n} A_{rt}(x_0, y_0)(x - x_0)^r(y - y_0)^t. \tag{20}$$

The polynomial is defined by the $(n + 1)^2$ numbers $A_{rt}(0, 0)$. The origin is moved by increments of powers of the radix in the x and y directions independently. When it has been moved by amounts x and y,

$$f(x, y) = A_{00}(x, y) \tag{21}$$

Now

$$A_{st}(x_0 + a, y_0) = \sum_{r=s}^{n} C_s^r a^{r-s} A_{rt}(x_0, y_0). \tag{22}$$

This operation is exactly like (5) for each t, ($t = 0, 1, 2 \cdots$) and the updating is achieved by a set of adding trees in exactly the same way. The generalization is therefore trivial.

Roots of polynomials

The significance of the method described here for evaluating polynomials is that it may be reversed and used to find roots. This is the analogue of reversing conventional multiplication to obtain conventional division.

It will first be supposed that $f(x)$ is a real polynomial and that it is known there is a distinct real root near x_0, though this restriction will be dropped in the next section. It will be supposed for definiteness that $f'(x) > 0$ near x_0 and that $f(x_0) < 0$.

To find that x for which $f(x) = 0$, a search procedure is set up. Unit increments are made and $f(x)$ is evaluated for $x = x_0; x_0 + 1; x_0 + 2, \cdots$, until an x is found for which $f(x) > 0$. At this stage x is decreased by one, making $f(x) < 0$, and a further set of increments of magnitude R^{-1} is made. This again continues until $f(x)$ goes positive, when again there is restoration and scaling. In this way as many digits of x are found as are needed. Alternatively a non-restoring procedure may be used in which as soon as $f(x)$ goes positive, scaled decrements are made until $f(x)$ goes negative again.

There is, of course, nothing novel about the search procedure. The significant point is that each new value of f is found from information evaluated previously, in a very simple way, so that there is little computation. Indeed the amount of computation made in finding the root is exactly that for finding $f(x)$ for given x, save for the extra steps caused by the restoration. This mirrors the difference between conventional division and multiplication.

The example in Table 1 illustrates the method used if the procedure shown is regarded as being reversed. The value x_0 may be taken to be the point $x_0 = 1$. Then $f(x)$ is calculated at $x = 2$ (which step is not shown), where it is found that $f(x) > 0$. Next there is restoration back to the point $x = 1$, and increments of magnitude 0.1 are made. The function $f(x)$ is calculated successively at $x = 1.0$, 1.1, 1.2, 1.3 (which latter step is not shown in Table 1). At the point $x = 1.3$ it is again found that $f(x) > 0$, so there is restoration to $x = 1.2$ and a step is made to $x = 1.21$ (not shown). Again $f(x) > 0$, so $f(x)$ is evaluated at $x = 1.200$, 1.201, 1.202, which latter value is found to be a root since $f(x) = 0$.

It is interesting to observe that if $f(x)$ is close to the root, the digits of $x = x_0$ are obtained essentially by subtracting the numbers $A_1(x_0)$, with suitable shifts, from $A_0(x_0)$, and recording the number of subtractions. This operation is clearly the long division of $f(x_0)$ by $f_1'(x_0)$ and so Newton's method is reproduced automatically.

To find a complex root of a complex polynomial $f(z)$, the two dimensional analogue of this procedure is set up. Unit and scaled increments and decrements are made in both the x and y directions with the object of making both the real and imaginary parts of $f(z)$ zero.

At any point z near the root, the signs of the real and

imaginary parts of $f_2'(z)$ and $f_2'(z)$, which are known, will prescribe a unique direction (one of $+x$, $-x$, $+y$, $-y$), in which a small movement will drive both the real and imaginary parts of $f(z)$ simultaneously towards zero. Initially unit steps are made in the directions prescribed at each stage and this continues until one of the variables overshoots. This is indicated by that variable repeating one of its former values. When this happens the step length for that variable is scaled by the radix, and the process is continued. In this way the digits of x and y are generated successively.

It is clear that in this case, too, the root is found as rapidly as $f(z)$ could be calculated for given z, except for the overshooting.

Systematic extraction of all real roots of a real polynomial

The methods described will find roots efficiently if reasonable approximations to them have been obtained. For the real roots of a real polynomial it is possible to improve upon this and to find all the real roots in a systematic way.

This is so because the signs of $f(x)$ and of its derivatives combine to locate the position of x precisely in relation to the real roots of $f(x)$. Furthermore the process produces these signs at every stage, as has been remarked here, and so no additional calculation is necessary. The essential content of this relationship between roots and derivatives is expressed by Budan's Theorem.[2] However, since this theorem is used in a different form from that in which it is stated, all results will be proved.

Relationships also exist between the positions of complex roots and the signs of the real and imaginary parts of the derivatives of a complex function, but they are more complicated and so will be presented in a future paper.

The method to be described shows how the t^{th} largest real root may be found directly where t is specified. The "t^{th} largest" has the obvious meaning when all the roots are real and an extended one when some roots are complex. This selectivity may be exploited if it is known beforehand which root is required. All the real roots may be found by making a set of n calculations with t set successively at $1, 2, 3, \cdots$.

For the real roots of a real polynomial, consider first the case where $f(x)$ is of degree n with n real roots. This implies that $f^{(r)}(x)$ has $n - r$ real roots and that they are interleaved with those of $f^{(r+1)}(x)$ for $r = 0, 1, 2, \cdots$. Define $k_r(x)$ to be the "labelling function" of $f^{(r)}(x)$. This function is defined to take values $0, 1, 2, \cdots, n - r$ in the $n - r + 1$ regions into which the x-space is partitioned by the roots of $f^{(r)}(x)$. In other words $k_r(x)$ is the number of sign variations in $f^{(r)}(x)$ as x varies from $+\infty$ to x. The functions $k_r(x)$ and $k_{r+1}(x)$ are shown together symbolically in Fig. 3, the vertical lines indicating the roots of

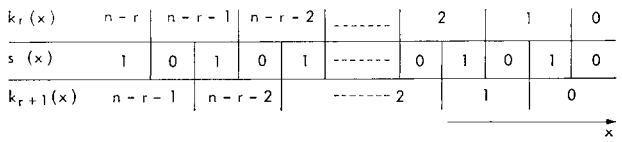

Figure 3 **Labelling functions for $f^{(r)}(x)$ and $f^{(r+1)}(x)$, where $f(x)$ has n real roots.**

$f^{(r)}(x)$ and $f^{(r+1)}(x)$. The function $s_r(x)$ is also shown in Fig. 3, where $s_r(x)$ is defined to be ZERO if the signs of $f^{(r)}(x)$ and $f^{(r+1)}(x)$ are the same, and ONE otherwise (it is assumed that the coefficient of x^n in $f(x)$ is positive). It is clear from Fig. 3 that

$$k_r(x) = k_{r+1}(x) + s_r(x) \tag{24}$$

Hence, since

$k_n(x) = 0$, then

$$k_0(x) = \sum_{r=0}^{n-1} s_r(x). \tag{25}$$

This is, therefore, the rule for calculating the "labelling" function for $f(x)$. *To find where x lies in relation to the roots of $f(x)$, it is merely necessary to count the number of sign variations in the sequence $f^{(n)}(x), f^{(n-1)}(x), \cdots, f(x)$.* For example, if $f(x)$ is a cubic and this sequence has the signs $+, -, +, +$, this shows that $k_0(x) = 2$, which implies that there are two roots to the right of this x and one to the left, as a sketch of $f(x)$ verifies.

In general the t^{th} root of $f(x)$ from the right is found by constraining x, to be decreased if $k_0(x) < t$ and to be increased otherwise. The function $f(x)$ may however have fewer real roots than n, and it is necessary to see what action this alogorithm has in that case, where the function $k_0(x)$ is still defined by (25).

If $f^{(r+1)}(x)$ has u_{r+1} real roots, $f^{(r)}(x)$ cannot have more than $u_{r+1} + 1$ real roots, and in general has a number $2v_r$ less than this. It may be said of $f(x)$ that v_r pairs of real roots are lost at the r^{th} derivative. This means that there exist v_r zeros of $f^{(r+1)}(x)$ at each of which $f^{(r)}(x)$ has the opposite sign from what it would have if $f^{(r)}(x)$ had the maximum number of roots, $u_{r+1} + 1$. These zeros are said to occur at the "critical" points. Altogether $f(x)$ will have Σv_r pairs of roots lost, and there will be Σv_r critical points. In order to account for all the roots of $f(x)$, $n - 2\Sigma v_r$ zeros and Σv_r critical points must be found.

It happens that the algorithm causes these critical points to be found automatically in those cases where pairs of real roots are lost, and so all the roots of $f(x)$ are accounted for.

First suppose that $f^{(r+1)}(x)$ has $n - r - 1$ real roots, but that one pair of roots is lost at the r^{th} derivative.

Figure 4 shows the labelling of the regions of $f^{(r+1)}(x)$, the function $s_r(x)$, and the consequent labelling of $f^{(r)}(x)$ in accordance with (24). (The vertical dotted line indicates merely the separation of regions.)

It is seen that the loss of the pair of real roots from $f^{(r)}(x)$ causes a region to be missing from $k_r(x)$ and an abrupt jump of 2 to occur in the labelling at the critical point.

If v_r pairs of roots are lost from $f^{(r)}(x)$ instead of one, then clearly there will be v_r abrupt jumps in the labelling of $f^{(r)}(x)$, and these will occur at the v_r critical points.

Now suppose that instead of $f^{(r+1)}(x)$ having $n - r - 1$ real roots, it has fewer, and that the labelling of the regions of $f^{(r+1)}(x)$ contains a number of abrupt jumps itself as a consequence. At any one such jump, the function $s_r(x)$ will not in general alter its value. Therefore since $k_r(x) = k_{r+1}(x) + s_r(x)$, there will be a similar abrupt jump of 2 in the labelling of $f^{(r)}(x)$.

It is therefore clear that the labelling of $f(x)$ will contain Σv_r abrupt jumps of 2 and that they will occur at the Σv_r critical points. The example in Fig. 5 shows the labelling for a typical quintic, where one pair of roots is lost at the third derivative and another at the zero derivative.

In general the search algorithm is applied with t set successively at $1, 2, \cdots, n$. The n values of x so found are either roots of $f(x)$ or critical points. The roots are distinguished because at these points $f(x) = 0$, whereas at the critical points $f^{(r)}(x) = 0$ for some $r > 0$. In the limiting case of repeated roots, critical points and roots may coincide but this causes no trouble.

In this way all the real roots of $f(x)$ are found systematically. The calculation of the critical points is not wasteful, since it is essential that these points should be found in order to account for all the roots. It should be observed that in this scheme each root is found using the original data.

If, instead, one is prepared to allow the errors that arise when factors are divided out, the factorization may be achieved trivially. This is because when one root x is found, the numbers A_r, $r = 1, 2, \cdots n$ are exactly the coefficients of the reduced polynomial referred to x as origin.

It is assumed that the problem is scaled so that the root that is being found lies in the range $1 \leq |x| < R$. This preliminary scaling can be made automatically, since if $|x|$ is not in this range as is shown by the initial stages of the search for the root, x can be put in this range by shifting the coefficients of the polynomial by appropriate amounts. If the coefficients are given as floating point numbers, they should be fixed and a search made to find the range of the root. The original floating coefficients should then be scaled by the appropriate amounts and fixed again.

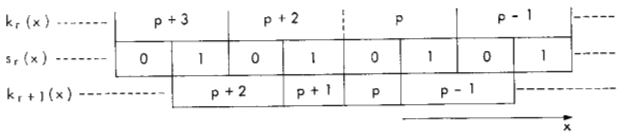

Figure 4 **Segment of labelling where a pair of roots is lost.**

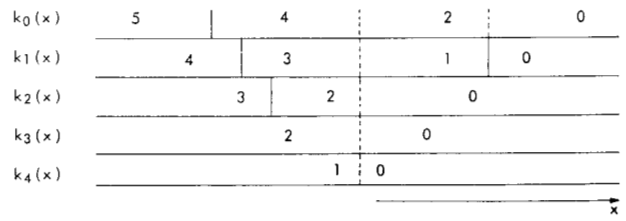

Figure 5 **Labelling for typical quintic.** (*See text*).

Polynominals defined by interpolation

It has been shown how $f(x)$ may be calculated for given x, where $f(x)$ is a polynomial defined by its coefficients. In some problems, such as interpolation, it is required to fit a polynomial through certain points, and then to find its value for a given x. It is possible to do this directly, where the differences of the function are manipulated in a manner analogous to the way derivatives were manipulated in the previous work. The hardware is very similar to that used before, but there is a restriction that binary arithmetic must be used.

To be precise, suppose E^{-1} is the unit backwards-displacement operator defined by

$$E^{-1}f(x) \equiv f(x - 1), \tag{26}$$

and suppose that $f(x)$ is an n^{th} degree polynomial defined by the $n + 1$ differences at the origin

$$(1 - E^{-1})^r f(0), \qquad r = 0, 1, \cdots, n \tag{27}$$

and that it is required to find $f(x)$ at

$$x = \sum_{i=0} q_i 2^{-i}. \tag{28}$$

It is clear that if the $(n + 1)$ differences for a difference interval of 2^{-i} are known at $x = \Sigma_{i=0}^{i-1} q_i 2^{-i}$, then the corresponding differences at $x + 2^{-i}$ can be calculated merely by adding successive differences together. The value q_i is therefore processed either by making these additions or not, depending on whether q_i is ONE or ZERO.

It is furthermore clear that if the $n + 1$ differences for an interval 2^{-i-1} at any point can be calculated from the corresponding differences for an interval of 2^{-i}, then q_{i+1} can be processed in the same way as q_i, and so all the

digits of x may be processed in a digit-by-digit manner.

The halving of the difference interval can be accomplished in an adding tree. Defining,

$$A_{rs}(x) \equiv 2^{ri+s}(1 - E^{-2^{-i-1}})^r(1 + E^{-2^{-i-1}})^{r-s}f(x), \tag{29}$$

where $r = 0, 1, \cdots n$, $s = 0, 1, \cdots r$.

For $s = 0$, the numbers $A_{r0}(x)$ are the $n + 1$ differences for an interval of 2^{-i}. (A factor 2^{ri} is included in the definition of the r^{th} difference, so that this difference tends to the derivative as the interval gets small.)

For $s = r$, the numbers $A_{rr}(x)$ are the differences for an interval of 2^{-i-1}. It is therefore necessary to find the numbers A_{rr} given the numbers A_{r0}.

The A_{rp} may be calculated for successive p, by means of an adding tree, since they satisfy the recurrence relationship

$$A_{rp}(x) = A_{rp-1}(x) + 2^{-i-2}\, A_{r+1p+1}(x)$$
$$p = 0, 1, \cdots r. \tag{30}$$

This tree is shown in the left part of Fig. 6, where the output of the p^{th} adder in the r^{th} row is $A_{rp}(x)$, it being supposed that there is a shift of $i + 2$ places in each diagonal segment.

The right part of Fig. 6 shows how the updated differences are used to find the corresponding differences after x has been incremented by 2^{-i-1}, which is necessary if q_{i+1} is one. In this case there is a shift of $i + 1$ places in each diagonal segment, owing to the scale factor in the definition of the differences.

For the case $n = 4$, with $a = 2^{-i-2}$, the tree makes

$$A_{44} = A_{40}$$

$$A_{33} = A_{30} + 3a_{40}$$

$$A_{22} = A_{20} + 2a\,A_{30} + 5a^2\,A_{40}$$

$$A_{11} = A_{10} + a\,A_{20} + 2a^2\,A_{30} + 5a^3\,A_{40}$$

and it may be verified directly that this is the correct relationship between the differences.

To evaluate $f(x)$, the tree in the left part of Fig. 6 is cycled once for each digit of x, with i set successively at 0, 1, 2, $\cdots$. The other additions shown on the right are performed for every digit of x that is ONE. The $n + 1$ numbers ultimately produced are $f(x)$ and its n derivatives.

Accuracy and speed analyses may be made as before. It is clear that other scaling schemes are possible and also that difference intervals may be doubled rather than halved.

To find a root of $f(x)$, the procedure is reversed, and a binary search is carried out under the control of the sign of $f(x)$. This provides a simple and rapid way of finding a zero of a function defined by interpolation.

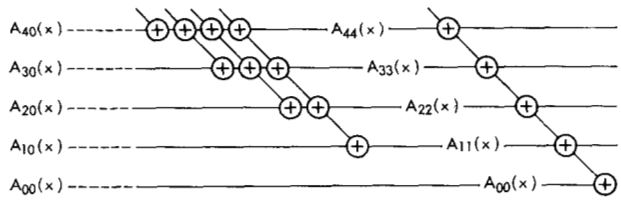

Figure 6 **Adding tree which implements Eq. (30).**

Conclusion

It should be observed that though this work is concerned with polynomials, any smooth function may be described by a series of "best fit" polynomials, different polynomials being used in different regions. It may therefore be arranged that whenever x moves from one region to another, the registers that describe the polynomial are updated according to a table, so that the polynomial used is always the best one. In this way the methods can be used for any function. The only difference between an exact calculation for an arbitrary function and a polynomial calculation of degree n is that the adding tree is truncated at level n instead of being infinite.

The powerful result of this work is that it shows how, at least for single functions of one variable, implicit functions can be calculated as easily as explicit ones. This has many applications, one being in the solution of differential equations, where the method enables "prediction" and "correction" to be merged into one operation without the need for iteration.

There would seem to be applications of these methods in hardware for large, fast computers where their speed could be exploited, and in small economical machines where their simplicity would be advantageous. These methods are not intended to be programmed in general on existing machines, though they would have advantages on certain machines such as those where multiplication is programmed as repeated addition.

The methods are also suitable for control computers. For example they may be used to calculate points on a contour defined by an implicit equation, for a numerical machine tool, or to evaluate the expected point of impact of a projectile whose path is defined by some points on its trajectory.

References

1. J. E. Meggitt, "Pseudo Division and Pseudo Multiplication Processes," *IBM Journal* 6, 210 (1962).
2. S. Borofsky, *Elementary Theory of Equations*, Macmillan, 1950, Ch. 6, p. 85.

Received November 2, 1962