ON THE NUMBER OF MULTIPLICATIONS FOR THE EVALUATION OF A POLYNOMIAL AND SOME OF ITS DERIVATIVES

Mary Shaw
J.F. Traub

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

August 7, 1972
Revised November 11, 1972

This work was supported in part by the National Science Foundation under Grant GJ-32111 and the Office of Naval Research under Contract N00014-67-A-0314-0010, NR 044-422. Part of the work was done while J.F. Traub was a Visiting Scientist at the National Center for Atmospheric Research, which is sponsored by the National Science Foundation.

CONTENTS

1. Introduction 1

2. An Algorithm for Calculating All Derivatives in a Linear Number of Multiplications 3

3. A Family of Splitting Algorithms 4

4. Arithmetic Operation Counts 8

5. Special Cases 9

6. Optimality 12

7. Applications 16

References 18

1. INTRODUCTION

Some of the recent work in computational complexity has dealt with the number of arithmetic operations needed to evaluate a polynomial or a polynomial and its first derivative [BO72], [MU71], [PA71]. Here we consider the evaluation of a polynomial and its first m derivatives and, in particular, the calculation of all the derivatives.

Let P denote a polynomial of degree n. Define a normalized derivative as $P^{(i)}/i!$. The normalized derivatives are commonly needed for applications (Section 7). When we refer to derivatives in this paper we always mean normalized derivatives. We consider P itself to be the zeroth derivative $P^{(0)}$.

All arithmetic operations are counted. A multiplication or division is denoted by M/D. Preconditioning is not permitted.

Prior to the new results reported here, the best algorithm for computing all the derivatives was the iterated use of Horner's rule (synthetic division), which requires $\frac{1}{2}n(n+1)$ multiplications and the same number of additions. (The iterated Horner's rule is defined in Example II of Section 5).

We give a new algorithm which computes all the derivatives of a polynomial in 3n-2 M/D. Unlike many algorithms which reduce the number of multiplications required to calculate some function, this algorithm is does not increase the number of additions. If n is odd, we give an algorithm (Example VI of Section 5) which computes all derivatives in 3n-3 M/D.

Both these algorithms belong to a family of algorithms for computing the first m derivatives. All the algorithms in the family require the same number of additions as the iterated Horner's rule for the first m derivatives.

Since n is the optimal number of multiplications for evaluating P alone, 3n-2 M/D is within a constant factor of optimality. Although we have no optimality results for derivatives, we can report related optimality results. We show that the calculation of either $a_i x^i$, $i=1,\ldots,n$ or $x^i P^{(i)}(x)/i!$, $i=0,\ldots,n$ in $2n-1$ multiplications is optimal.

We summarize the contents of the paper: in Section 2 we present the algorithm for computing all derivatives in 3n-2 M/D. A family of splitting algorithms for computing the first m derivatives and arithmetic operation counts for these algorithms are given in the next two sections. In Section 5 we obtain three known algorithms, two new algorithms, and one algorithm very similar to a recently discovered technique as special cases. Optimality results are presented in Section 6. We close by giving applications of these algorithms.

2. AN ALGORITHM FOR CALCULATING ALL DERIVATIVES IN A LINEAR NUMBER OF MULTIPLICATIONS

In this section we exhibit an algorithm for computing a polynomial and all its derivatives at a point x in a number of M/D which is linear in the degree of the polynomial. Let

$$P(x) = \sum_{i=0}^{n} a_{n-i} x^i$$

Algorithm

$$T_i^{-1} = a_{i+1} x^{n-i-1}, \quad i=0,1,\ldots,n-1$$

$$T_j^j = a_0 x^n, \quad j=0,1,\ldots,n$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j, \quad j=0,1,\ldots,n-1, \quad i=j+1,\ldots,n$$

This is a special case of a one-parameter family of algorithms presented in the next section. We show there that

$$T_n^j = \frac{P^{(j)}(x)}{j!} x^j, \quad j = 0,1,\ldots,n-1$$

Since $x^2,\ldots,x^n$ may be obtained in n-1 multiplications while $a_0 x^n,\ldots, a_{n-1} x$ may be obtained in n multiplications, the T_n^j may be obtained in 2n-1 multiplications. We show in Section 6 that this is optimal. The derivatives $P^{(j)}(x)/j!$, $j=1,\ldots,n-1$ may then be obtained in n-1 divisions. Since $P^{(n)}(x)/n! = a_0$, it need not be computed. Thus this algorithm yields all the normalized derivatives in 3n-2 M/D.

Consider the matrix of T_i^j with i and j indicating row and column index. The derivatives may be calculated in increasing order by calculating the matrix by columns or in decreasing order by calculating the matrix by diagonals. These two variations have identical roundoff properties, since they produce the same chains of intermediate results. No rounding error analysis has yet been performed.

3. A FAMILY OF SPLITTING ALGORITHMS

We study a family of algorithms for computing the first m derivatives of a polynomial. Assume, without loss of generality, that

$$n+1 = p\ q$$

Define

$$s(j) = (n-j) \bmod q\ ,\ j=0,1,\ldots,n$$

Algorithm

$$(3.1)\quad T_i^{-1} = a_{i+1}x^{s(i+1)}\ ,\ i=0,1,\ldots,n-1$$

$$(3.2)\quad T_j^j = a_0 x^{s(0)}\ ,\ j=0,1,\ldots,m$$

$$(3.3)\quad T_i^j = T_{i-1}^{j-1} + T_{i-1}^j x^{s(i-j)-s(i-j-1)+1}\ ,\ j=0,\ldots,m,\ i=j+1,\ldots,n$$

We show that this recurrence may be used to compute the derivatives. The key is the following theorem. Let C(k,j) denote binomial coefficients.

Theorem

$$(3.4)\quad T_i^j = x^{s(i-j)} \sum_{k=j}^{i} C(k,j)a_{i-k}x^{k-j}$$

Proof In the triangle where they are defined, the T_i^j are uniquely determined by the starting values (3.1), (3.2) and the recurrence relation (3.3). We need only verify that the T_i^j given by (3.4) satisfy the starting conditions and the recurrence.

The starting conditions are satisfied since

$$T_i^{-1} = x^{s(i+1)} \sum_{k=-1}^{i} C(k,-1)a_{i-k}x^{k+1} = x^{s(i+1)}a_{i+1}\ ,\ \text{since } C(-1,-1)=1,\ C(k,-1)=0$$

$$T_j^j = x^{s(0)} \sum_{k=j}^{j} C(k,j)a_{j-k}x^{k-j} = x^{s(0)}a_0$$

The recurrence is satisfied since

$$T_i^j - T_{i-1}^{j-1} = x^{s(i-j)} \left[\sum_{k=j}^{i} C(k,j)\, a_{i-k} x^{k-j} - \sum_{k=j-1}^{i-1} C(k,j-1)\, a_{i-1-k} x^{k+1-j} \right]$$

$$= x^{s(i-j)+1} \sum_{k=j}^{i-1} C(k,j)\, a_{i-1-k} x^{k-j}$$

$$= x^{s(i-j)-s(i-j-1)+1} T_{i-1}^j ,$$

which completes the proof.

Corollary

$$T_n^j = \frac{P^{(j)}(x)}{j!} \quad x^j \bmod q$$

Proof

$$T_n^j = x^{s(n-j)} \sum_{k=j}^{n} C(k,j)\, a_{n-k} x^{k-j}$$

Using the properties of s(j), the recurrence may be written as

$$T_i^{-1} = a_{i+1} x^{s(i+1)} , \quad i=0,1,\ldots,n-1$$

$$T_j^j = a_0 x^{q-1} , \quad j=0,1,\ldots,m$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j , \quad (i-j) \bmod q \neq 0$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j x^q , \quad (i-j) \bmod q = 0, \; j=0,\ldots,m, \; i=j+1,\ldots,n$$

The following pseudo-Algol program implements the algorithm. Assume still that n+1 = p q. Further let

$$m = r\, q + s$$

where r and s are obtained by division of m by q. In the interest of clarity, an n+2 by m+2 element array is used to develop the derivatives.

It is clearly possible to rewrite the program to use only about n storage locations.

```
begin
[x is the point of evaluation (x≠0), a(i) are coefficients]
[x(i) will be x^i, T(i,j) will be T_i^j]
x(0) ← 1
x(1) ← x
for i = 2,3, ..., q
     x(i) ← x * x(i-1)
     [powers of x require q-1 multiplications]
for i = 0,q, 2q, ..., (p-1)*q
     begin
     for k = 0, 1, ..., q-2
          T(i+k-1,-1) ← a(i+k)*x(q-k-1)
          [inner loop requires q-1 multiplications each time]
     T(i+q-2,-1) ← a(i+q-1)
     [multiplication by coefficients requires total of p*(q-1) multiplications]
     end
[entire initialization requires (p+1)(q-1) multiplications]
for j = 0,1, ..., m
     begin
     T(j,j) ← T(j-1,j-1)
     for i = j+1, j+2, ..., n
          if (i-j) mod q = 0 then
               T(i,j) ← T(i-1,j-1) + T(i-1,j) * x(q)
          else
               T(i,j) ← T(i-1,j-1) + T(i-1,j)
     end
```

[recurrence requires $(m+1)\ (n-\frac{1}{2}\ m)$ additions and $(m+1)(p-r-1)+\frac{1}{2}\ q\cdot r(r+1)$ multiplications]

[T(n,j) is now $x^{j \bmod q}\ P^{(j)}(x)/j!$]

```
for j = 0,q, 2q, ..., (r-1)*q
    for k = 1,2,..., q-1
        T(n,j+k) ← T(n,j+k)/x(k)
for j = 1,2,..., s
    T(n,r*q+j) ← T(n,r*q+j)/x(j)
```

[m-r divisions used to obtain normalized derivatives]

end

4. ARITHMETIC OPERATION COUNTS

Recall that we wish to calculate the first m derivatives of a polynomial of degree n. As above, let

$$n + 1 = p\ q\ , \qquad m = r\ q + s$$

where r and s are obtained by division of m by q. It can be shown that the family of algorithms uses $(m+1)(n-\frac{1}{2}\ m)$ additions, independent of the parameter q. It uses the following number of M/D:

$(p+1)(q-1)$	multiplications for initialization,
$(m+1)(p-r-1) + \frac{1}{2}\ q\ r(r+1)$	multiplications for the recurrence,
$m - r$	divisions to calculate $P^{(j)}(x)/j!$

Let $f_{m,n}(q)$ denote the total number of M/D required to calculate the first m derivatives of a polynomial of degree n if the splitting q is used. Then

$$(4.1) \qquad f_{m,n}(q) = n-1 + q + \frac{m(n+1)}{q} - (m+2)r + \frac{1}{2}\ q\ r(r+1)$$

The algorithm can be slightly improved in two cases. If all n derivatives are required, $P^{(n)}(x)/n! = a_0$ and it is not necessary to compute it from $x^{q-1}\ \frac{P^{(n)}(x)}{n!}$, so one division is saved. If $q = n + 1$, it is not necessary to compute x^{n+1}, so one multiplication is saved. These special cases are <u>not</u> reflected in the function $f_{m,n}(q)$.

Given m and n, the best choice of q can be determined by minimizing $f_{m,n}(q)$ subject to the constraint that q be an integer. Analysis of best splittings as a function of m and n will be reported in a future paper.

5. SPECIAL CASES

By choosing particular values of m and q, we specialize the algorithm and operation count formula of the last two sections. The first three algorithms are known and are included for comparison. The fourth algorithm is very similar to an algorithm discovered by Munro [MU71]. The last two algorithms are new.

I. $m = 0,\ q = 1$

$$T_i^{-1} = a_{i+1}\ ,\ i=0,1,\ldots,n-1$$

$$T_0^0 = a_0$$

$$T_i^0 = T_{i-1}^{-1} + T_{i-1}^0\, x,\ i=1,\ldots,n$$

$$P(x) = T_n^0$$

This is Horner's rule and requires n additions and n multiplications.

II. $m = n,\ q = 1$

$$T_i^{-1} = a_{i+1}\ ,\ i=0,1,\ldots,n-1$$

$$T_j^j = a_0\ ,\ j=0,1,\ldots,n$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j x\ ,\ j=0,1,\ldots,n-1,\ i=j+1,\ldots,n$$

$$\frac{P^{(j)}(x)}{j!} = T_n^j\ ,\ j=0,1,\ldots,n$$

This is the iterated Horner's rule and requires $\frac{1}{2}\,n\,(n+1)$ additions and $\frac{1}{2}\,n\,(n+1)$ multiplications.

III. $m = 0,\ q = n + 1$

$$T_i^{-1} = a_{i+1}x^{n-i-1}\ ,\ i=0,1,\ldots,n-1$$

$$T_0^0 = a_0x^n$$

$$T_i^0 = T_{i-1}^{-1} + T_{i-1}^0 \quad , \; i=1,\ldots,n$$

$$P(x) = T_n^0$$

This is the "naive" way of evaluating a polynomial by calculating all the monomial terms first. The general specification of the algorithm requires n additions and 2n multiplications; however, in this case $x^q = x^{n+1}$ is never used, so 2n-1 multiplications suffice.

IV. $\underline{m = 1, \; q = \sqrt{n+1} = \sigma}$

$$T_i^{-1} = a_{i+1}x^{s(i+1)} \quad , \; i = 0,1,\ldots,n-1$$

$$T_j^j = a_0 x^{\sigma-1} \quad , \; j=0,1$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j x^{s(i-j)-s(i-j-1)+1} \quad , \; j=0,1, \; i=j+1,\ldots,n$$

$$\frac{P^{(j)}(x)}{j!} = x^{-j}T_n^j \quad , \; j=0,1$$

The recurrence may also be written as

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j \quad , \; (i-j) \bmod \sigma \neq 0$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j x^{\sigma} \quad , \; (i-j) \bmod \sigma = 0$$

This is a new algorithm for computing P(x) and P'(x). It requires 2n-1 additions and $n-1 + 2\sqrt{n+1}$ M/D.

For simplicity of exposition, we have assumed that n+1 is the square of an integer. If this is not the case, q may be taken as approximately $\sqrt{n+1}$.

The calculation of P(x) and P'(x) by the iterated Horner's rule requires 2n-1 additions and 2n-1 multiplications. Munro [MU71] gives an algorithm for computing P and P' which he asserts requires $2n + 2\sqrt{n}$ additions and $n + 2\sqrt{n}$ multiplications. Our algorithm uses about the

same number of multiplications but fewer additions. Our algorithm is very similar to Munro's, but we have not performed a detailed analysis of the differences.

V. $m \le n$, $q = n + 1$

$$T_i^{-1} = a_{i+1}x^{n-i-1} \quad , \; i=0,1,\ldots,n-1$$

$$T_j^j = a_0 x^n \quad , \; j=0,1,\ldots,m$$

$$T_i^j = T_{i-1}^{j-1} + T_{i-1}^j \quad , \; j=0,1,\ldots,m, \; i=j+1,\ldots,n$$

$$\frac{P^{(j)}(x)}{j!} = x^{-j}T_n^j \quad , \; j=0,1,\ldots,m$$

This algorithm requires $(m+1)(n-\frac{1}{2}m)$ additions. The predicted number of M/D is $f_{m,n}(n+1) = 2n+m$, but x^{n+1} is never used, so $2n+m-1$ M/D will suffice. Further, if $m = n$, $P^{(n)}/n! = a_0$ so we can obtain all n derivatives in $\frac{1}{2}n(n+1)$ additions and $3n-2$ M/D. This special case was presented in Section 2.

VI. Let n be odd, $m = n$, $q = \frac{1}{2}(n+1)$

This algorithm calculates all derivatives in $\frac{1}{2}n(n+1)$ additions and $3n-3$ M/D.

6. OPTIMALITY

We shall prove that the calculation of the $x^j p^{(j)}(x)/j!$, $j=0,\ldots,n$ in 2n-1 multiplications, as done by the algorithm of Section 2, optimizes the number of multiplications. Before proving this result we summarize what is known on optimality with respect to additions, multiplications, and total arithmetic operations.

If m = 0, Horner's rule (Example I of Section 5) optimizes both additions and multiplications. Furthermore, Borodin [BO71] has shown it is the only algorithm which optimizes additions and multiplications.

If m = 1, the new algorithm given in Example IV of Section 5 requires $n-1+2\sqrt{n+1}$ M/D. Munro's algorithm requires about the same number of M/D. These are best algorithms known for P and P'.

If m = n, the new algorithms given in Examples V and VI, which require 3n-2 M/D and 3n-3 M/D (for n odd), respectively, are the best algorithms known.

All the splittings require $(m+1)(n-\frac{1}{2}m)$ additions. For m = 0, this reduces to n additions which is known to be optimal. For m = 1, this reduces to 2n-1 additions which Kirkpatrick [KI71] has shown to be optimal.

If m = 0, 2n arithmetic operations are optimal. If m = n, Borodin [BO72a] has shown that a polynomial and all its derivatives can be evaluated in $O(n \log^3 n)$ arithmetic operations. Our algorithms do this calculation in $\frac{1}{2}n^2 + O(n)$ arithmetic operations. Since $n^2 < n \log^3 n$ for $n <$ about 10^3, Borodin's result pertains only asymptotically. The optimal number of arithmetic operations for both n small and n large is open.

We now consider the optimality of the evaluation of $x^j P^{(j)}(x)/j!$, $j=0,1,\ldots,n$, with respect to multiplication. First we shall require a result which is interesting in its own right. We use the notation $\partial(V)$ to denote the degree of the polynomial V.

Theorem

Given a number x and n arbitrary numbers $a_1, \ldots, a_n$, the computation of $a_1 x, \ldots, a_n x^n$ using multiplications and additions requires at least

2n-1 multiplications. Furthermore, if only 2n-1 multiplications are used, it is impossible to evaluate a_1x, ..., a_nx^n, $Q(x)$, for any polynomial $Q \ni \partial(Q) > n$.

Proof

The proof is by induction on n. The theorem is certainly true for n = 1.

Assume the theorem has been proven for n = L. Let $\sigma_L(x)$ denote the set $a_1x,\ldots, a_Lx^L$. We shall first show that 2L+1 multiplications are required to evaluate $\sigma_{L+1}(x)$. By the inductive hypothesis 2L-1 multiplications are not enough. We now show that 2L multiplications are not enough. The most general form involving a_{L+1} which can be built without multiplications involving a_{L+1} is $Ka_{L+1} + U(x)$ where K is an integer and U(x) is a polynomial which is independent of a_{L+1}. Let

$$(6.1) \quad T(x) = [K_1a_{L+1} + U_1(x)]\,[K_2a_{L+1} + U_2(x)]$$

be the first multiplication in the chain of operations leading to $a_{L+1}x^{L+1}$. Then

$$(6.2) \quad T(x) = C + a_{L+1}Z(x) + W(x)$$

where

$$C = K_1K_2a_{L+1}^2 \ , \ Z(x) = K_1U_2(x) + K_2U_1(x) \ , \ W(x) = U_1(x)U_2(x)$$

If $\partial(Z) \geq L+1$, then by the inductive assumption, the evaluation of $\sigma_L(x)$, $Z(x)$, $T(x)$ requires 2L+1 multiplications. If $\partial(Z) < L+1$, then the evaluation of $\sigma_L(x)$, $T(x)$ requires 2L multiplications and another multiplication is required to evaluate $a_{L+1}x^{L+1}$. Hence we have shown that the evaluation of $\sigma_{L+1}(x)$ requires at least 2L+1 multiplications.

Suppose now that exactly 2L+1 multiplications are used. The induction is complete if we show it is then impossible to evaluate $\sigma_{L+1}(x)$, $Q(x)$, for any polynomial $Q \ni \partial(Q) > L+1$. We use the notation (6.1), (6.2). We consider three cases depending on $\partial(Z)$.

Case 1. $\partial(Z) > L+1$. Then the evaluation of $\sigma_L(x)$, $Z(x)$, $T(x)$ requires 2L+1 multiplications. $T(x)$ has a term in $a_{L+1}x^{\mu}$, $\mu > L+1$. To extract $a_{L+1}x^{L+1}$ from $T(x)$ requires another polynomial depending on $a_{L+1}x^{\mu}$ which is impossible.

Case 2. $\partial(Z) = L+1$. Then the evaluation of $\sigma_L(x)$, $Z(x)$, $T(x)$ requires 2L+1 multiplications. If $\partial(W) = L+1$, no polynomial of degree greater than L+1 has been produced. If $\partial(W) = \mu > L+1$, another polynomial of degree μ must be available and this is impossible.

Case 3. $\partial(Z) = \mu < L+1$. Then the evaluation of $\sigma_L(x)$, $T(x)$, requires 2L multiplications. We may write

$$T(x) = a_{L+1}x^{\mu} + V(x) + W(x)$$

where $\partial(V) < \mu$. If $\partial(W) \leq L+1$ it is impossible to evaluate a polynomial of degree greater than L+1 and extract $a_{L+1}x^{L+1}$ with just one more multiplication. If $\partial(W) > L+1$ it is impossible to produce $a_{L+1}x^{L+1}$ and extract it with just one more multiplication.

Thus with 2L+1 multiplications it is impossible to evaluate a_1x_1 ..., $a_{L+1}x^{L+1}$, $Q(x)$ for any polynomial $Q \ni \partial(Q) > L+1$. This completes the proof and the theorem.

Theorem

The evaluation of $x^j P^{(j)}(x)/j!$, $j=0,1, \ldots,n$, requires at least 2n-1 multiplications.

Proof

By the previous theorem, the calculation of $a_j x^{n-j}$, $j=0,1,\ldots,n-1$ requires $2n-1$ multiplications. The $x^j P^{(j)}(x)/j!$ can be expressed in terms of the $a_j x^{n-j}$ as a triangular linear system with integer coefficients, with unity multiplying the $a_j x^{n-j}$. This system can be solved for the $a_j x^{n-j}$ as linear combinations of the $x^j P^{(j)}(x)/j!$ with integer coefficients. If the $x^j P^{(j)}(x)/j!$ could be calculated with less than $2n-1$ multiplications, then so could the $a_j x^{n-j}$ which contradicts the previous theorem and completes the proof.

7. APPLICATIONS

Polynomial derivatives often occur in applications of Taylor series, which, of course, involve normalized derivatives.

As an example we consider the problem of shifting the zeros of a polynomial, given its coefficients. Let $P(t) = \sum_{i=0}^{n} a_{n-i} t^i$ have zeros $\lambda_1, \ldots, \lambda_n$. Then

$$Q(t) = P(t+x) = \sum_{i=0}^{n} \frac{P^{(i)}(x)}{i!} t^i$$

has zeros $\lambda_1 - x, \ldots, \lambda_n - x$.

Shifting of zeros is a key ingredient in what is called Horner's method for solving polynomial equations. Horner's name is attached to three algorithms: nested evaluation of polynomials, calculation of all derivatives by the iterated "Horner's rule", and a method for solving polynomial equations. Ironically, he was anticipated in each of these by other workers (Traub [TR66, pp. 298-299]).

Horner's method for solving polynomial equations is a digit-at-a-time technique for calculating polynomial zeros. Although Horner's method is no longer competitive with modern zero finding algorithms, zero shifting is still a useful technique. Stewart [ST71] has performed an analysis of the effect of rounding errors in the iterated Horner's rule and concludes that zeros near the shift are not unduly perturbed. No rounding error analysis has yet been performed on the new algorithms of this paper.

Finally we observe that sometimes it is the $x^j p^{(j)}(x)/j!$ rather than the derivatives which are needed. Let

$$\nu_i = \frac{x^i p^{(i)}(x)}{i!} .$$

Many one-point iterations (Traub [TR64, Chapter 5]) can be written in terms of the ν_i. Thus Newton iteration is

$$\Phi = x\left(1 - \frac{\nu_0}{\nu_1}\right)$$

and the third order iteration of Euler type is

$$\psi = x\left[1 - \frac{\nu_0}{\nu_1} - \left(\frac{\nu_0}{\nu_1}\right)^2 \frac{\nu_2}{\nu_1}\right].$$

ACKNOWLEDGEMENT

We would like to thank A. Cline, National Center for Atmospheric Research, for the ideas underlying the proof of the first theorem of Section 6.

REFERENCES

[BO71] Borodin, A., Horner's Rule is Uniquely Optimal. Theory of Machines and Computations, edited by Kohavi and Paz, 1971, pp. 45-58.

[BO72] Borodin, A., Computational Complexity - Theory and Practice. To appear in Currents in the Theory of Computing, edited by A. Aho, Prentice-Hall, 1972.

[BO72a] Borodin, A., Private communication.

[KI71] Kirkpatrick, D., On the Additions Necessary to Compute Certain Functions. M.Sc. Thesis, Department of Computer Science, University of Toronto, 1971.

[MU71] Munro, J. I., Some Results in the Study of Algorithms. Technical Report 32, Department of Computer Science, University of Toronto, 1971.

[PA71] Paterson, M.S. and Stockmeyer, L., Bounds on the Evaluation Time for Rational Polynomials. IEEE Conference Record, Twelfth Annual Symposium on Switching and Automata Theory, 1971, pp. 140-143.

[ST71] Stewart, G. W., Error Analysis of the Algorithm for Shifting the Zeros of a Polynomial by Synthetic Division. Mathematics of Computation, Vol. 25 (1971), pp. 135-139.

[TR64] Traub, J. F., Iterative Methods for the Solution of Equations. Prentice-Hall, 1964.

[TR66] Traub, J. F., Associated Polynomials and Uniform Methods for the Solution of Linear Problems. SIAM Review (8) 1966, pp. 277-301.