COMPUTATIONAL COMPLEXITY AND NUMERICAL STABILITY

Webb Miller*
Computer Science Dept.
Penn State
University Park, Pa. 16802

ABSTRACT: Limiting consideration to algorithms satisfying various numerical stability requirements may change lower bounds for computational complexity and/or make lower bounds easier to prove. We will show that, under a sufficiently strong restriction upon numerical stability, any algorithm for multiplying two $n \times n$ matrices using only +, - and $\times$ requires at least n^3 multiplications. We conclude with a survey of results concerning the numerical stability of several algorithms which have been considered by complexity theorists.

1. Introduction

Proofs of lower bounds for the computational complexity of a given problem must generally consider only a few measurements of cost and ignore the others. For example a discussion of the complexity of the problem of sorting a list of items may count only the number of comparisons, disregarding issues like non-comparison operations, storage requirements and programming simplicity.

When the problem under consideration involves real (i.e., floating-point) arithmetic, then another factor is relevant, namely the propagation of rounding errors. In a recent survey of arithmetic complexity Borodin [2, p. 174] remarks:

> But eventually we will have to develop results which simultaneously talk about arithmetic costs, and the "robustness" or "stability" of the algorithm. Perhaps if we were more formal about the numerical properties of an algorithm, then it might be easier to produce non trivial lower bounds.

Of course many people have given simultaneous consideration to complexity and stability. In particular numerical analysts are daily faced with the trade-off between stability and the operation count. Moreover, investigations have been made of the effects of rounding errors upon several algorithms of interest to complexity theorists (for a survey see section 6).

Rather, the first sentence quoted from Borodin seems to suggest the possible existence of problems for which the fastest algorithms must be less than optimal with regard to stability. One motivation for this paper is to present several such examples. Thus we are interested in the effects upon complexity lower bounds of various stability requirements (essentially the opposite problem of maximizing stability subject to complexity constraints is investigated by Babuska [1] and Viten'ko [15]).

However, our primary motivation is to be found in Borodin's second sentence. When we limit consideration to all programs (in a given language) which evaluate a fixed function and which satisfy a certain stability requirement it is entirely possible that the fastest programs are excluded. More intriguing to us is the possibility that the remaining programs have a simple structure which makes finding a program that is optimal with respect to some measurement of cost substantially easier than if arbitrary (and less stable) programs are allowed to compete.

Here we have focussed on the evaluation of systems of bilinear forms by programs which apply only the operations +, - and $\times$. For simplicity we have not allowed constants, though almost everything carries over with minor modifications. One of our results is that if only such programs meeting a very restrictive stability requirement are considered, then n^3 multiplications are needed to find the product of two $n \times n$ matrices. Under a somewhat relaxed stability assumption we will show that computing the product of a 2×2 matrix and a $2 \times n$ matrix requires at least $7n/2$ multiplications. In each case, if the stability requirement is dropped, then faster algorithms can be found and tight lower bounds seem to become harder to verify (they are not yet known).

The concepts of numerical stability which we employ are closely related to those currently used by experts in roundoff analysis (e.g., Wilkinson [16,17]). They are idealized in at least two respects. First they use very few of the actual properties of floating-point arithmetic. This creates a tendency for pessimistic results in that algorithms may be more stable than we can prove (see the remarks by Kahan [9, pp. 1232-1234] on Viten'ko [15]. Second we consider only the first-order effects of rounding errors, so algorithms

*Most of the work for this paper was done at the IBM Thomas J. Watson Research Center.

may be less stable than results like ours suggest. (In section 5 we will detail several more reasons why the practical significance of the results given here is not especially great.)

We hope that this paper offers supporting evidence for Borodin's cited belief and for our conviction that numerical analysts and complexity theorists can benefit from an exchange of ideas.

2. Numerical stability of polynomial programs

A polynomial program is a sequence of instructions of the form

$$V \leftarrow W\#X \qquad (2\text{-}1)$$

where # is one of +, - or ×. For simplicity we require that each operand, W or X, is a variable (initial or defined).

We can think of the program as specifying a sequence of floating-point operations to be performed on the data $\underline{d} = (d_1,\ldots,d_n)$. In many contexts it is helpful to take the alternative point of view that the program specifies symbolic operations to be performed on multivariate polynomicals in the initial variables $\underline{d}$. Any resulting polynomial $V(\underline{d})$ has integer coefficients and no constant term.

Let V be a defined variable of such a program, i.e., V appears on the left of a unique instruction (2-1). If we omit the instruction defining V and consider V an initial variable, then any variable Z can be considered a polynomial $Z(\underline{d},V)$.

Define

$$\Delta Z_V(\underline{d}) \equiv \frac{\partial Z}{\partial V}(\underline{d},V(\underline{d}))\cdot V(\underline{d})$$

The polynomial $\Delta Z_V(\underline{d})$ measures the sensitivity of the numerical evaluation of $Z(\underline{d})$ to a floating-point rounding error committed in the operation producing $V(\underline{d})$. For suppose that the evaluation is performed exactly except that $V(\underline{d})(1+\delta)$ is used in place of $V(\underline{d})$. The induced error is

$$\begin{aligned} &Z(\underline{d},V(\underline{d})(1+\delta)) - Z(\underline{d}) \\ &= Z(\underline{d},V(\underline{d})(1+\delta)) - Z(\underline{d},V(\underline{d})) \\ &\approx \delta\cdot\Delta Z_V(\underline{d}). \end{aligned}$$

for small δ.

For example let $\underline{d} = (a_1,a_2,b_1,b_2)$ and consider

$$\begin{aligned} &T \leftarrow a_1 \times a_2 \\ &U \leftarrow b_1 \times b_2 \\ &V \leftarrow a_1 + b_2 \\ &W \leftarrow a_2 + b_1 \\ &X \leftarrow V \times W \\ &Y \leftarrow X - T \\ &Z \leftarrow Y - U \end{aligned} \qquad (2\text{-}2)$$

One easily computes

$$Z(\underline{d}) \equiv a_1b_1 + a_2b_2$$
$$Z(\underline{d},W) \equiv (a_1 + b_2)W - a_1a_2 - b_1b_2$$
$$\frac{\partial Z}{\partial W}(\underline{d},W) = a_1 + b_2$$
$$\Delta Z_W(\underline{d}) = (a_1 + b_2)(a_2 + b_1)$$

Stability conditions can sometimes be interpreted as restricting the form of certain polynomials $\Delta Z_V(\underline{d})$. If

$$\frac{\partial Z}{\partial V}(\underline{d},V(\underline{d})) \not\equiv 0 \qquad (2\text{-}3)$$

then the form of $V(\underline{d})$ is also restricted since

$$V(\underline{d})\cdot\frac{\partial Z}{\partial V}(\underline{d},V(\underline{d})) = \Delta Z_V(\underline{d}).$$

Our goal in the remainder of this section is to exhibit a useful condition which guarantees (2-3).

Let Z be a defined variable of a polynomial program. Consider the following process for marking certain instructions of the program. If Z is defined by an addition or subtraction, then mark that instruction. If an instruction

$$V \leftarrow X\#Y \quad , \quad \#\text{ is + or -}$$

is marked and if X and/or Y is defined by + or -, then mark those instructions.

Now delete any unmarked instructions. The set I(Z) of initial (i.e., undefined) variables of the resulting polynomial program contains only variables which were originally either initial or defined by multiplication (take I(Z) = {Z} if Z is defined by multiplication).

Clearly Z is a linear combination of the U in I(Z). Specifically, there exists a unique integer $i_Z(U)$ for each U in I(Z) such that in the resulting program

$$Z \equiv \sum_I i_Z(U)\cdot U$$

The dependence of Z on I(Z) is essentially the same in the original program. In particular,

$$Z(\underline{d}) \equiv \sum_I i_Z(U)\cdot U(\underline{d}) \qquad (2\text{-}4)$$

and for any defined variable V we have

$$Z(\underline{d},V) \equiv \sum_I i_Z(U)\cdot U(\underline{d},V) \qquad (2\text{-}5)$$

For an example of these notions recall (2-2). We find that I(Z) = {T,U,X} and

$$1 = i_Z(X) = -i_Z(T) = -i_Z(U)$$

Theorem 1.1. If V is a defined variable in I(Z) and if $i_Z(V) \neq 0$, then

$$\frac{\partial Z}{\partial V}(\underline{d},V(\underline{d})) \not\equiv 0$$

Proof: If U in I(Z) is originally defined by a multiplication then U may depend on V in the sense that

$$\frac{\partial U}{\partial V}(\underline{d},V(\underline{d})) \not\equiv 0$$

The only other U in I(Z) are originally initial variables and do not depend on V. From (2-5) we compute

$$\frac{\partial Z}{\partial V}(\underline{d},V(\underline{d})) = \sum\nolimits_I i_Z(U)\cdot\frac{\partial U}{\partial V}(\underline{d},V(\underline{d}))$$
$$= i_Z(V) + \sum\nolimits^* i_Z(U)\cdot\frac{\partial U}{\partial V}(\underline{d},V(\underline{d})) \qquad (2\text{-}6)$$

where the sum $\sum^*$ is over all $U \neq V$ in $I(Z)$ which are defined by a multiplication

$$U \leftarrow X \times Y$$

For such U we find

$$\frac{\partial U}{\partial V}(\underline{d},V(\underline{d}))$$
$$= \frac{\partial X}{\partial V}(\underline{d},V(\underline{d}))\cdot Y(\underline{d}) + \frac{\partial Y}{\partial V}(\underline{d},V(\underline{d}))\cdot X(\underline{d})$$

Neither of these last two summands has a constant term. It follows that the constant term in (2-6) is $i_Z(V)$. This proves Theorem 1.1. □

3. Stable evaluation of bilinear forms

Consider a polynomial program defining a variable B which evaluates a bilinear form

$$B(\underline{a},\underline{b}) \equiv \sum_{i=1}^{m}\sum_{j=1}^{n} \sigma_{ij}a_ib_j \qquad (3\text{-}1)$$

Here the input has been partitioned as $\underline{d} = (\underline{a},\underline{b}) = (a_1,\ldots,a_m,b_1,\ldots,b_n)$ and the σ_{ij} are integer constants. In this section we will define and investigate four types of numerical stability which are applicable to such variables.

If f and g are real-valued functions of $(\underline{a},\underline{b})$, then

$$f(\underline{a},\underline{b}) = O(g(\underline{a},\underline{b}))$$

means that there exists a constant K such that

$$|f(\underline{a},\underline{b})| \leq K\cdot|g(\underline{a},\underline{b})|$$

for all $\underline{a},\underline{b}$. We also need the notation

$$|\underline{a}| = \max\{|a_i|:1 \leq i \leq m\}$$
$$|\underline{b}| = \max\{|b_j|:1 \leq j \leq n\}$$
$$|(\underline{a},\underline{b})|_B = \max\{|a_ib_j|:\sigma_{ij} \neq 0\}$$

Definition 1. The following four kinds of numerical stability are defined by the requirement that each defined variable V satisfies the corresponding equality.

(1) Brent stability:

$$\Delta B_V(\underline{a},\underline{b}) = O(|\underline{a}|\cdot|\underline{b}|)$$

(2) restricted Brent stability:

$$\Delta B_V(\underline{a},\underline{b}) = O(|(\underline{a},\underline{b})|_B)$$

(3) weak stability:

$$\Delta B_V(\underline{a},\underline{b}) = O(|\underline{a}|\cdot\sum_{i=1}^{m}|\frac{\partial B}{\partial a_i}(\underline{a},\underline{b})| + |\underline{b}|\cdot\sum_{j=1}^{n}|\frac{\partial B}{\partial b_j}(\underline{a},\underline{b})|)$$

(4) strong stability:

$$\Delta B_V(\underline{a},\underline{b}) = O(\sum_{i=1}^{m}|a_i\cdot\frac{\partial B}{\partial a_i}(\underline{a},\underline{b})| + \sum_{j=1}^{n}|b_j\cdot\frac{\partial B}{\partial b_j}(\underline{a},\underline{b})|)$$

Definition (1) is an idealization of a notion studied by Brent [3,4]. It is idealized in that "second-order" effects of rounding errors are obliterated by the differentiation in the definition of ΔB_V.

Definitions (3) and (4) are formalizations of ideas from "backward error analysis" (see Miller [11]). Their intuitive thrust is that the result computed with roundoff error is the exact result for slightly altered data. Here "slightly altered" means (in the case of strong stability) that each coordinate is accurate to within a few rounding errors or (in the case of weak stability) that the error in each a_i (respectively, b_j) is small compared to $|\underline{a}|$ (respectively, $|\underline{b}|$).

Strong stability and weak stability are meaningful when discussing any rational algorithm. However, (restricted) Brent stability is meaningful only for the evaluation of bilinear forms.

Proposition 3.1. The following implications hold among the notions of Definition 1.

$$\begin{tikzcd} & (2) \arrow[dr, Rightarrow] & \\ (4) \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & (1) \\ & (3) \arrow[ur, Rightarrow] & \end{tikzcd}$$

Verifying Proposition 3.1 is easy. For instance, to show that restricted Brent stability implies Brent stability one need only note that

$$|(\underline{a},\underline{b})|_B = O(|\underline{a}|\cdot|\underline{b}|)$$

It is also easy to give conditions under which two of the stability requirements coalesce. We will use

Proposition 3.2. Suppose (3-1) is a permutation bilinear form, i.e., suppose

$$\sigma_{ij} \neq 0 \text{ and } \sigma_{IJ} \neq 0 \Rightarrow$$

(i) $i = I$ and $j = J$, or

(ii) $i \neq I$ and $j \neq J$.

Then strong stability is equivalent to restricted Brent stability.

Proof: If $\sigma_{ij} \neq 0$, then $\frac{\partial B}{\partial a_i}(\underline{a},\underline{b}) = \sigma_{ij}b_j$.

Hence

$$|(\underline{a},\underline{b})|_B = O(\Sigma|a_i\cdot\frac{\partial B}{\partial a_i}(\underline{a},\underline{b})|)$$

This shows that restricted Brent stability implies strong stability. Proposition 3.1 does the rest.

□

Restricted Brent stability does not, in general, imply strong stability. Consider $B(\underline{a},\underline{b}) = a_1b_1 + a_1b_2 + a_2b_1 + a_2b_2$ and

$$
\begin{aligned}
&U \leftarrow a_1 \times b_1\\
&V \leftarrow a_1 \times b_2\\
&W \leftarrow a_2 \times b_1\\
&X \leftarrow a_2 \times b_2 \qquad\qquad (3\text{-}2)\\
&Y \leftarrow U + Y\\
&Z \leftarrow Y + W\\
&B \leftarrow Z + X
\end{aligned}
$$

One easily computes

$$\Delta B_U(\underline{a},\underline{b}) = a_1 b_1$$

But if $a_1 = b_1 = 1 = - a_2 = - b_2$, then

$$|\underline{a}|\cdot\Sigma|\frac{\partial B}{\partial a_i}(\underline{a},\underline{b})| + |\underline{b}|\cdot\Sigma|\frac{\partial B}{\partial b_j}(\underline{a},\underline{b})| = 0$$

It follows that the program does not have weak stability. On the other hand, one easily sees that it possesses restricted Brent stability. Of course this B is not a permutation bilinear form.

Theorem 3.3. Let V in I(B) be defined by a multiplication and suppose that $i_B(V) \neq 0$ and that $V(\underline{a},\underline{b}) \neq 0$. If B has Brent stability then the definition of V must be of the form

$$L_1(\underline{a}) \times L_2(\underline{b}) \text{ (or } L_2(\underline{b}) \times L_1(\underline{a}))$$

where L_1 and L_2 are linear (e.g., $L_1(\underline{a}) = \Sigma\alpha_i a_i$ with integers α_i).

Proof: The condition $\Delta B_V(\underline{a},\underline{b}) = O(|\underline{a}|\cdot|\underline{b}|)$ can be seen to require that ΔB_V be bilinear. Theorem 1.1 shows that V must be bilinear. The result follows. □

Theorem 3.3 shows that example (2-2), an instance of Winograd's method for inner products, does not possess Brent stability (see also Brent [3,4]).

The next result shows that if we add to the hypotheses of Theorem 3.3 the assumption of restricted Brent stability, then any term appearing in V must appear in B. The intuitive reason is that otherwise a term, $\eta_{ij}a_i b_j$, must cancel algebraically in a later ± operation, and corresponding numerical cancellation creates an error not $O(|(\underline{a},\underline{b})|_B)$.

Theorem 3.4. Let V in I(B) be defined by a multiplication and suppose $i_B(V) \neq 0$. Write

$$V(\underline{a},\underline{b}) = \sum\sum \eta_{ij} a_i b_j.$$

If $\eta_{ij} \neq 0$, then the coefficient σ_{ij} in B (3-1) is nonzero.

Proof: From $\Delta B_V(\underline{a},\underline{b}) = O(|(\underline{a},\underline{b})|_B)$ we may conclude that if $a_i b_j$ appears in ΔB_V then it appears in B. The result follows from the fact that if $V(\underline{a},\underline{b}) \neq 0$, then ΔB_V must be a constant multiple of V. □

Theorem 3.5. Suppose that B is a permutation bilinear form evaluated by a polynomial program and that B has strong stability (or equivalently, that B has restricted Brent stability--see Proposition 3.2). If $a_i b_j$ appears in B with nonzero coefficient, then there is a multiplication in the program of the form

$$(\alpha a_i) \times (\beta b_j)$$

Proof: The term $a_i b_j$ must appear in some V in I(B) satisfying $i_B(V) \neq 0$ (see (2-4)). If V is defined by

$$(\sum \alpha_I a_I) \times (\sum \beta_J b_J)$$

where, say, α_i, α_I and β_j are nonzero, $i \neq I$, then a term $a_I b_j$ must appear in B by Theorem 3.4. This contradicts the assumption that B is a permutation bilinear form. □

Two remarks are in order. If a permutation bilinear form B is computed by directly finding its terms and adding and subtracting them to get B, then B has strong stability. Second, if B is not a permutation bilinear form, then such programs may lack strong stability (see (3-2)) and "fast" programs may possess it (for example, the evaluation $B = (a_1+a_2)(b_1+b_2)$).

4. Speed sacrifices stability

We will say that a polynomial program to evaluate a system of s bilinear forms

$$B_k(\underline{a},\underline{b}) \equiv \sum_{i=1}^{m}\sum_{j=1}^{n} \sigma_{ijk}\, a_i b_j;\ k = 1,\ldots,s \qquad (4\text{-}1)$$

possesses, e.g., simultaneous Brent stability if each B_k has Brent stability. In this section we will draw two conclusions from our previous results.

Conclusion 1. Any polynomial program for (4-1) which has simultaneous Brent stability can make use of multiplications only of the form

$$L_1(\underline{a}) \times L_2(\underline{b})$$

Example 4.1. Any polynomial program of the "$L_1(\underline{a}) \times L_2(\underline{b})$" form to multiply a 2 × 2 matrix times a 2 × n matrix requires at least $\lceil 7n/2 \rceil$ multiplications, and this bound can be achieved (Hopcroft and Kerr [8]). However, Winograd's method requires only 3n + 2 multiplications. Thus if n ≥ 5, then requiring simultaneous Brent stability increases the minimum number of multiplications. Moreover, the requirement seems to simplify the verification of lower bounds, since it is not known if Winograd's method is optimal.

Example 4.2. Often one does not count multiplications which are "preconditioning", i.e., which produce only polynomials in $\underline{a}$ alone or in $\underline{b}$ alone. To reduce the number of multiplications in a polynomial program by preconditioning, one must sacrifice simultaneous Brent stability.

Example 4.3. It can be shown that any program of the "$L_1(\underline{a}) \times L_2(\underline{b})$" form must exhibit simultaneous Brent stability. Thus Strassen's algorithm for matrix multiplication has simultaneous Brent stability (see also Brent [3]).

Conclusion 2. If each B_k is a permutation bilinear form, then any polynomial program with simultaneous strong stability must perform t multiplications, where t is the number of pairs (i,j) such that some σ_{ijk} in (4-1) is nonzero.

Example 4.4. The n^3 multiplications in the usual algorithm for multiplying $n \times n$ matrices make it optimal among polynomial programs with simultaneous strong stability. For general polynomial programs the arithmetic complexity of matrix multiplication seems far from resolved.

Example 4.5. Conclusion 2 also applies to the multiplication of complex numbers or polynomials (i.e., computing the coefficients of the product of polynomials). Consider

$$(a_1 + a_2 i) \times (b_1 + b_2 i)$$

i.e., computing

$$B_1(\underline{a},\underline{b}) = a_1 b_1 - a_2 b_2 = \text{real part}$$

$$B_2(\underline{a},\underline{b}) = a_1 b_2 + a_2 b_1 = \text{imaginary part.}$$

If we compute $B_1 = X - Y$ and $B_2 = X - Z$ where

$$X = (a_1 + a_2) b_1$$

$$Y = a_2(b_1 + b_2)$$

$$Z = a_1(b_1 - b_2)$$

then our previous results indicate that the influence of roundoff error upon B_1 should be "unduly large" for $\underline{a},\underline{b}$ such that

$$|(\underline{a},\underline{b})|_{B_1} \ll |\underline{a}| \cdot |\underline{b}|$$

The correct result for

$$(0.0015 + 1.01i) \times (1.01 + 0.001i)$$

is $B_1 = 0.000505$, whereas computing in "three-digit floating-point arithmetic", i.e., rounding symmetrically to three digits after each operation, gives 0.00051 with the usual method, and 0.0 with the "fast" method.

5. Caveat

The practical value of the above results is limited by several factors.

1. They consider only polynomial operations. If a system of bilinear forms can be evaluated in n multiplications, then it can be evaluated with n multiplications of the form

$$L_1(\underline{a},\underline{b}) \times L_2(\underline{a},\underline{b})$$

for linear L_i (Winograd [18]. Such polynomial programs (with constants) are "nearly Brent stable" in the sense that

$$\Delta B_V(\underline{a},\underline{b}) = O([\max\{|a_i|,|b_j|\}]^2)$$

for all B and defined V. They can be given simultaneous Brent stability be scaling $\underline{a}$ and $\underline{b}$ to have roughly the same size (see Brent [3,4] for a special case).

2. Sometimes it costs as much to run a "more stable" algorithm in single precision as it costs to run a fast algorithm in double precision.

3. For a particular method, range of data, machine and software *ad hoc* roundoff analyses will often override general results like ours.

6. Some known stability results

This section contains a brief review of stability results concerning algorithms of (possible) interest to complexity theorists.

(a) Evaluation of a polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

given x, $a_0, \ldots, a_n$. A popular stability condition is that the computed value be exactly $\sum a_i^* x^i$ where each relative difference $|a_i^* - a_i|/|a_i|$ is small. Following our approach this might be formalized as

$$\Delta P_V(x,\underline{a}) = O(\sum |a_i x^i|)$$

for all defined variables V. There is nothing uniquely plausible about this requirement, and others have been considered, e.g., [10].

However, the condition does seem to be widely applicable. Both Horner's role and the naive method are easily seen to be stable in this sense. Also, Wozniakowski [19] has verified this property for a family of algorithms of Shaw and Traub [14] for evaluating a polynomial and its normalized derivatives.

(b) Polynomial evaluation with preconditioning. Workers involved in producing subroutines for evaluating common functions have considered the possible use of the Pan and Motzkin-Belaga forms. However, these procedures are too often contaminated by numerical errors (see Rice [13] or pp. 67-73 of Hart, *et al.* [7]).*

(c) Interpolation. A semi-formal roundoff analysis of Lagrange's formula can be found on pp. 287-291 of Dorn and McCracken [6]. It seems natural to compare Lagrange's or Newton's form with fast methods [2], both in their use for evaluating

*On the other hand, preliminary work by M. Rabin and S. Winograd indicates the existence of stable preconditioning methods of practical value.

the interpolating polynomial at a point and for finding its coefficients.

However, plausible stability conditions need to be agreed upon before one can formally discuss the propagation of rounding error in these methods. The problem here is more acute than for general polynomial evaluation or (especially) matrix multiplication. Much of the difficulty stems from the fact that one may well not care how a method performs with completely arbitrary data, e.g., when the polynomial assumes alternating values of 1 and -1. Moreover, in practice only very low degree interpolating polynomials are used (with rare exceptions).

(d) The fast Fourier transform. Many studies have concluded that the FFT is quite stable (see Ramos [12] for results and references).

(e) Parallel evaluation of arithmetic expressions. Brent [5] proves the stability (in approximately the sense of our strong stability) of certain near-optimal schemes for the parallel evaluation of arithmetic expressions lacking division. For expression containing division his schemes may lose strong stability.

7. Acknowledgments

Allan Borodin convinced me that this subject should be explored and provided many helpful ideas. S. Winograd pointed out reference [8] and his work with Rabin mentioned in section 6. Support from H. R. Strong made this work possible.

References

[1] Babuska, I., Numerical stability in mathematical analysis, Proc. 1968 IFIP Congress, Vol. I, pp. 11-23, North-Holland, Amsterdam, 1969.

[2] Borodin, A., On the number of arithmetics required to compute certain functions--circa May 1973, in Complexity of Sequential and Parallel Numerical Algorithms, (J. Traub, ed.), pp. 149-180, Academic, New York, 1973.

[3] Brent, R. P., Algorithms for matrix multiplication, Stanford Report CS157, 1970.

[4] Brent, R. P., Error analysis of algorithms for matrix multiplication and triangular decomposition using Winograd's identity, Numer. Math. 16 (1970), 145-156.

[5] Brent, R. P., The parallel evaluation of arithmetic expressions in logarithmic time, in Complexity of Sequential and Parallel numerical algorithms, (J. Traub, ed.) Academic, New York, 1973.

[6] Dorn, W. and McCracken, D., Numerical Methods with FORTRAN IV Case Studies, Wiley, New York, 1972.

[7] Hart, J., *et al.*, Handbook of Computer Approximations, Wiley, New York, 1965.

[8] Hopcroft, J. and Kerr, L., On minimizing the number of multiplications necessary for matrix multiplication, SIAM J. Appl. Math. 20 (1971), 30-36.

[9] Kahan, W., A survey of error analysis, Proc. 1971 IFIP Congress, pp. 1214-1239, North-Holland, Amsterdam, 1972.

[10] Mesztenyi, C. and Witzgall, C., Stable evaluation of polynomials, J. Research Nat. Bureau Standards 71B (1967), 11-17.

[11] Miller, W., Remarks on the complexity of roundoff analysis, to appear in Computing.

[12] Ramos, G., Roundoff error analysis of the fast Fourier transform, Math. Comp. 25 (1971), 757-768.

[13] Rice, J., On the conditioning of polynomial and rational forms, Numer. Math. 7 (1965), 426-435.

[14] Shaw, M. and Traub, J., On the number of multiplications for the evaluation of a polynomial and some of its derivatives, JACM 21 (1974), 161-167.

[15] Viten'ko, I., Optimal algorithms for adding and multiplying on computers with a floating point, USSR Comp. Math. and Math. Phy. 8, #5 (1968), 183-195.

[16] Wilkinson, J., Rounding errors in algebraic processes, Prentice Hall, New Jersey, 1963.

[17] Wilkinson, J., Modern error analysis, SIAM Review 13 (1971), 548-568.

[18] Winograd, S., On the number of multiplications necessary to compute certain functions, Comm. on Pure and Appl. Math. 23 (1970), 165-179.

[19] Wozniakowski, Rounding error analysis for the evaluation of a polynomial and some of its derivatives, to appear in SIAM J. Numer. Anal.