Proceedings of the International Congress of Mathematicians
Berkeley, California, USA, 1986

Equation Solving in Terms of Computational Complexity

ARNOLD SCHÖNHAGE

1.

1.1. *Introduction.* Computational complexity is a new principle in Mathematics, rooted in the algorithmic and constructive tradition of our science. Beyond its prominent role in theoretical Computer Science this aspect has meanwhile entered many different areas of Mathematics. Complexity considerations are intimately related to Logic and Foundations and to Numerical Methods with their innumerable applications, but they are also of growing interest in other fields like Geometry, Number Theory, and Algebra.

In order not to get stuck in pure generality, we confine our very broad subject to the computational treatment of *algebraic* equations. Even in this restricted sense, though, "equation solving" has been investigated for centuries and is now a central topic of Numerical Mathematics and in Computer Algebra. The specific interest of this survey is in the computational *complexity* of such problems. Beginning with a period of a few early papers, corresponding research has been carried on continuously for about twenty years now. Thus it seems to be timely to report on some of the results on this occasion.

In view of limited space and time we will discuss *sequential* algorithms only. A survey of important complexity results for models of *parallel* computation has recently been given by Cook [**9**]. According to personal taste and preference, we will furthermore restrict our considerations to *deterministic* computations, not ignoring the fact that, from a practical point of view, *probabilistic* algorithms may prove to be superior for certain difficult problems. Moreover, there is always a natural interplay between equation solving and the nondeterministic mode of solution guessing. Accordingly, it will sometimes be illuminating to compare the complexity of equation *solving* with that of *verification*, which means checking whether given data are really forming a solution.

Solving algebraic equations can be understood in different ways. For the prime fields $GF(p)$ and $\mathbf{Q}$ together with their finite extensions the discrete methods of symbolic computation apply. In this framework the major task may be defined as the construction of suitable splitting fields. We mention two papers, [**23**] and [**6**], dealing with the complexity of the general problem. With regard to applications, however, the upper bounds obtained by these authors are discouragingly high. It is possibly premature to expect complete answers while we are still unable to master the complexity bounds for solving systems of *linear* equations (see §4). Considerable progress has been made in factoring polynomials and on related topics. In their pioneering paper [**26**], A. K. Lenstra, H. W. Lenstra, and L. Lovász have shown how to factor univariate polynomials over $\mathbf{Q}$ in polynomial time. Meanwhile many variations, extensions, refinements of this result, or of the underlying basis reduction algorithm, have been published (see [**5**, **13**, **21**, **24**, **25**, **39**] and the extensive bibliography in [**15**]). In particular, testing for *solvability* in the very classical sense of Galois theory is possible in polynomial time, cf. [**22**].

When dealing with equations over $\mathbf{R}$ or $\mathbf{C}$, we should adopt a different point of view. Our complexity results about the fundamental theorem of algebra presented in §3 will refer to computations with dynamic precision in the sense of recursive dependence, quite in the spirit of H. Weyl's farseeing paper [**53**] of 1924. More specific explanations about the corresponding model of computation are given below in §1.2.

Within the last years, the complexity of root finding and related problems have independently been investigated by S. Smale and others, but along completely different lines so that their results are incomparable to ours. In view of the program of this Congress it should perhaps suffice just to mention [**43**] and the overview [**44**] with its further references.

The subsequent presentation deals with rather basic problems of equation solving. §2 on the complexity of multiplication and division of integers and polynomials can be regarded as a complexity discussion of the primitive equation $ax = b$ over corresponding domains. In §3 we present the fundamental theorem of algebra in terms of computational complexity, including a neat solution for this long standing numerical problem. The final section will concern systems of linear equations and the complexity of matrix computations, characteristic equations, and the computation of eigenvalues.

1.2. *Models of computation.* There are many equivalent ways to define computability. Among these, the classical Turing machine concept is especially suited to provide intuitively appealing measures for the quantitative analysis of algorithms. As long as we are only interested in complexity classes invariant under polynomial reducibilities, further details about the underlying model do not matter. In lower order complexity, however, finer distinctions should be made. Many of the concrete algorithms given in the literature can be made to work for *multitape* Turing machines. For higher flexibility, *random access machines* with a variety of instruction sets can be used, conveniently modeling the addressable storage features of present-day computers. In some sense, the "true," canonical basis for measuring time-complexity is furnished by the theoretical model of *pointer machines* (previously also called "linking automata," or "storage modification machines," cf. [**35**]). They are real-time equivalent to very simple random access machines under unit cost, only capable of indirect addressing, testing for equality, and computing the successor function.

With respect to practical implementations, it will be more realistic to consider random access machines under the *logarithmic cost criterion*, where handling any integer is weighted by the length of its binary code. Then any upper bound T on "pointer time" (the number of steps of some pointer machine under consideration) will get amplified by not more than a factor of order $\log T$. One should keep in mind, however, that the applicability of this model is limited by the size of inner storage. For large scale problems with higher storage requirements, the multitape Turing machines may form the more adequate model again.

When discussing time complexity for such machine models, some convention about the encoding of input and output data is required. For simplicity, let us assume here that *integers* shall always be represented in standard *binary* form, though it cannot be excluded that other encodings may lead to different results (as an example, testing for powers of ten seems to be much easier for inputs in decimal form). Elements of $GF(p)$ are given as (reduced) residues $\bmod p$, rational numbers as pairs of integers, and elements from finite extensions of these domains are then representable as tuples of binary integers in a straightforward manner.

For approximate computations with real or complex numbers it is especially convenient to use *binary rationals*, i.e., binary integers scaled with some power of two. There is, however, a significant difference between the ad hoc notion of floating point numbers (called "reals" in everyday programming languages) and our understanding of a real number. Any *input* number $\alpha \in \mathbf{R}$ shall (potentially) be available at any desired precision: when called with a specified parameter value N, some *oracle* will deliver some binary rational a such that $|\alpha - a| < 2^{-N}$ is satisfied, without extra cost. Different oracles may possibly specify the same real number in this way. Despite the availability of arbitrary high precision for inputs α, β their equality $\alpha = \beta$ is recursively undecidable. Similarly, outputs resulting from an approximating computation will be due up to some prescribed precision again.

In the abstract setting of *algebraic complexity theory* (see [**4**] and the more recent survey [**50**]) one uses the machine independent notion of *straight-line programs*. Usually, the inputs are considered as indeterminates $x_1, x_2, \ldots$ over some ground field F, and time is measured (sequentially) by the number of arithmetical operations carried out by such a program in $F(x_1, \ldots)$, or in $F[x_1, \ldots]$, if divisions are not admitted. *Nonscalar complexity* refers to the mode where only *essential* multiplications and divisions are counted, while additions and scalar multiplications are taken for free. In case of problems for which comparison based branching (or zero testing) is required the more general model of computation trees applies (cf. [**49**]).

2.

2.1. *The complexity of multiplication and division.* Let us begin with the very simple equation $ax = b$. When considered over the integers, testing its solvability and finding a solution x is possible by integer division with remainder zero. For checking a given solution x, integer *multiplication* will suffice, but it is an interesting open problem whether there are faster methods of verification. Over the rationals (represented by pairs of integers), solving $ax = b$ amounts to two integer multiplications, followed by a gcd computation provided the result shall be delivered in reduced form. Similarly, an *extended* gcd computation will be required for solving $ax = b \bmod q$ (assuming that b is divisible by $d = \gcd(a, q)$), i.e., to find the gcd d and *cofactors* u, v satisfying the conditions

$$au + qv = d \quad \text{and} \quad d|a \quad \text{and} \quad d|q. \tag{2.1}$$

The key to all these tasks is approximate division over the reals. Its complexity can be bounded by that of integer multiplication, since computing an n-bit approximation for $1/a$ (for $1 \leq a < 2$, say) is possible by means of a division-free Newton iteration with two multiplications per step, whence the time complexity of n-bit division is bounded by

$$O(\mu(n) + \mu(n/2) + \mu(n/4) + \mu(n/8) + \cdots) \leq O(\mu(n)), \tag{2.2}$$

where $\mu(N)$ denotes any bound for the time complexity of N-bit integer multiplication (satisfying some regularity condition to justify the estimation of the sum in (2.2)). Therefore, solving $ax = b$ over the reals (with a bounded away from zero) has essentially no higher complexity than verification of a solution by multiplication. Conversely, $ab = b/(1/a)$ shows how to reduce multiplications to divisions.

In [**18**, §4.3.3], D. Knuth gives a rather complete account of what is presently known about the time complexity of integer multiplication. For multitape Turing machines, the upper bound from our 1971 paper [**41**] based on FFT methods modulo numbers of the form 2^L+1 is still the best we have. A streamlined version of this approach is contained in [**37**]. On the other hand, so far nobody has derived any nonlinear *lower* bound, except for models under certain additional restrictions. In fact, any such lower bound proof would necessarily have to employ some specific properties of Turing machines, since pointer machines, for instance, are capable of integer multiplication in *linear time* (cf. [**35**]). The latter result is based upon the numerical FFT with suitable precision. Accordingly, our present knowledge about the time complexity of solving the equation $ax = b$ over **Z**, **R** (or over **C** as well) is captured by the following bounds.

(2.3) Upper bounds for the time complexity of N-bit multiplication:

$\mu(N) = cN \log(N+1) \log\log(N+2)$ for multitape Turing machines,

$\mu(N) = cN$ for pointer machines (unit cost),

$\mu(N) = cN \log(N+1)$ for pointer machines (logarithmic cost).

For the other cases mentioned above, where gcd computations come in, the complexity seems to be higher. Based on Knuth's idea to combine fast integer multiplication with a "half-gcd" technique due to Lehmer, it has been shown [**31**] that computing the gcd of two numbers of at most N bits in length (together with corresponding cofactors) is possible in time $O(\mu(N)\log N)$ (opposed to time of order N^2 for Euclid's algorithm). It seems to be a rather safe conjecture that the extra factor of order $\log N$ over multiplication time is inevitable. Note, however, that, for given cofactors u, v, *verification* of (2.1) is indeed possible within the order of multiplication time.

In the framework of algebraic complexity with unit cost per arithmetical operation, the complexity of solving $ax = b$ seems to be a trivial subject, as one division will suffice, but things become highly nontrivial, if we discuss this problem of equation solving for finite-dimensional algebras over some field F. Consider, for instance, the field of complex numbers as an algebra over $\mathbf{R}$. One can fairly easily see that the nonscalar complexity of multiplying two complex numbers equals 3 (counting real multiplications and divisions only), but it was only recently that the corresponding problem for the *division* of complex numbers could be settled; cf. [**27**] for the rather intricate proof that the known upper bound of 6 real multiplications/divisions is indeed optimal.

Much work has been done concerning the multiplicative complexity of algebras. For further details and corresponding references we recommend the survey [**50**]. Less information is available about "division" in this framework. In the next section we shall briefly discuss the special case of the division of univariate polynomials. The central topic of matrix inversion will be postponed to §4.3.

2.2. *Basic computations with polynomials.* Now we consider the equation $a(t)x(t) = b(t)$ for polynomials over some field F, at first with regard to the algebraic model counting all arithmetical operations in F at unit cost. Input and output shall be coefficientwise, with dense encoding. More precisely, F is assumed to have the form $F = G(a_0, a_1, \ldots, b_0, \ldots)$ with all the inputs as indeterminates over some ground field G. Again we may distinguish various domains like $F[t], F(t)$, or $F[t]/(q(t))$, in analogy to $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{Z}/q\mathbf{Z}$ discussed before. Here the role of $\mathbf{R}$ is taken by the ring of formal power series over F. Operating with variable precision in $F[[t]]$ means to operate in $F[t]/(t^m)$ for increasing values of m. Thus the complexity of the division of polynomials can be reduced to that of polynomial multiplication (again up to a constant factor) via computing approximate reciprocals of units in $F[[t]]$ by means of Newton iteration, see [**42**, **19**]. Actually, these authors are discussing the simpler case of *nonscalar* complexity, which, for polynomial multiplication $\bmod t^m$, is exactly known ($= 2m - 1$, provided the ground field G contains at least $2m-2$ elements). Kung's upper bound of $4m$ for computing reciprocals $\bmod t^m$ and its conjectured optimality can be replaced by the better estimate $3.75m$, while the precise constant factor is yet unknown.

The algebraic complexity of polynomial multiplication is essentially that of discrete convolutions, thus closely related to discrete Fourier transforms. If the ground field G contains suitable roots of unity, e.g. all 2^kth or all 3^kth roots of 1 (for all k), then FFT can be used immediately; otherwise extra measures are required. The best bounds known so far are very similar to those for integer multiplication.

(2.4) Upper bounds for the number of arithmetical operations for the multiplication of mth degree polynomials over ground field G:

$$M(m) = cm \log(m+1) \quad \text{if } G \text{ supports FFT,}$$
$$M(m) = cm \log(m+1) \log\log(m+2) \quad \text{for any field.}$$

The general bound with the extra log log factor comes from an FFT method recursively applied to the polynomial rings $F[t]/(t^K + 1)$ for various values of $K = 2^k$, analogous to the corresponding fast integer multiplication technique. This approach fails for fields of characteristic 2, but then one can use

$$F[t]/(t^{2K} + t^K + 1)$$

with $K = 3^k$ instead (see [**34**]). Also for these algebraic models, no nonlinear lower bounds are known for the discrete Fourier transform or for polynomial multiplication.

The best upper bounds presently available for the complexity of solving the equation $a(t)x(t) = b(t) \bmod q(t)$ for an arbitrary mth degree polynomial $q(t)$ are higher by a factor of order $\log m$ again. Rather precise upper and lower bounds for the nonscalar complexity of extended polynomial gcd computations have been obtained by Strassen (see [**49**], where also further references can be found). A numerical version of the gcd problem (over the field $\mathbf{C}$, or $\mathbf{R}$) is treated in [**40**].

This last remark leads us to the crucial question to be studied next: What are the implications of the *algebraic* complexity results for the (machine bounded) time complexity of the corresponding *numerical* computations with polynomials?

By inserting numbers (elements of $\mathbf{Z}$, $\mathbf{R}$, or $\mathbf{C}$) for the coefficients of the polynomials, fast algebraic algorithms clearly should get transformed into fast numerical procedures (provided one has numerical stability), but there are also other kinds of *machine algorithms*, not obtainable in this way, and we cannot exclude the possibility that some of these are significantly faster than anything constructed purely algebraically. In fact, the numerical multiplication of polynomials seems to furnish an example of this phenomenon. Simply combining the unit cost FFT bounds from (2.4) with the estimates (2.3) for integer multiplication will at best yield the time bound $O(m \log m\ \mu(N))$—but one can do better!

Replacing the variable t by 2^K with some $K \geq 2N + \log(m+1) + 1$ reduces the multiplication of two mth degree polynomials with coefficients of at most N bits in length to one long integer multiplication of size $O(m(N + \log(m+1)))$, and for $N > \log(m+1)$ that leads to the improved time bound $O(\mu(mN))$. For pointer machines we thus have even the linear bound $O(mN)$. The linearity in m also shows that, at least asymptotically, the celebrated FFT is not an optimal numerical method for the discrete Fourier transform $\mathbf{C}^m \to \mathbf{C}^m$. The details of this approach can be found in [**37**].

When multiplying polynomials with complex coefficients in this way via fast integer multiplication modulo $2^{2L} + 1$ one can nicely exploit that $2^L = \mathrm{sqrt}(-1)$

can serve as the imaginary unit in this domain. This feature considerably enhances the applicability of the method.

Numerical *division* of polynomials requires some extra care with respect to stability. For the division by some (complex) polynomial F, its leading coefficient should be bounded away from zero. A good way to express this quantitatively is by an upper bound on the *root radius* $\rho(F)$ (defined as the maximal modulus of the roots of F in $\mathbf{C}$), together with some normalization of the *size* of F, measured by the l^1-norm $|F|$ of the coefficient vector. The corresponding result from [**37**] is

(2.5) Numerical division of $G \in \Pi_m$ by $F \in \Pi_n$ within error 2^{-N}, i.e., computing some $Q \in \Pi_{m-n}$ and $R \in \Pi_{n-1}$ such that $|G - QF - R| < 2^{-N}$, is possible in pointer time $O(m(N + m + m\log(1 + r)))$, where r is a *given* bound on $\rho(F)$, and $|G| \leq 1 \leq |F| \leq 2$ is assumed.

Restricted to divisions with a uniform bound on the root radius of the denominators, the given bound on pointer time is of order $O(mN)$ again, provided the precision N is of order m at least.

3. The fundamental theorem of algebra in terms of computational complexity.

For many centuries the problem of solving polynomial equations was mainly studied in terms of *formulae*. After the early days when Renaissance mathematicians had mastered the third and fourth degree it took more than 250 years until Gauss gave rigorous proofs for the *existence* of the roots of any real equation and Abel could show that, in general, one cannot obtain them by merely applying arithmetical operations and successively solving pure equations. This narrow notion of 'solvability' has been carried on up to this day, at least verbally, despite the rather clear comments in §9 of Gauss's dissertation that 'resolutio aequationis' and 'ipsius reductio ad aequationes puras' should properly be distinguished.

It was only after about another one hundred years that solving general equations with complex coefficients was understood adequately, namely in terms of *algorithms*. Weyl's constructive proof [**53**] for the fundamental theorem of algebra essentially shows that the zeros of a complex polynomial depend on its coefficients *recursively* (see also Specker's contribution in [**11**]). In other words, there exists a Turing machine which, when fed with an integer n and with oracles for the coefficients of some mth degree polynomial, will output approximations for the m zeros of this polynomial within an error bound of 2^{-n}. In terms of *computational complexity* the decisive question is now: *how fast* can this be done?

3.1. *The computational problem.* It is a well-known fact that clustered zeros of a polynomial are less stable under small perturbations of the coefficients than isolated zeros. Therefore it seems to be more appropriate to understand any prescribed accuracy in the sense of backward analysis: a collection of approximate zeros should be considered acceptable, if their elementary symmetric functions agree well enough with the coefficients of the given polynomial. This applies to $P \in \Pi_m$, $P(z) = a_0 + a_1z + \cdots + a_mz^m$ with leading coefficient $a_m = 1$, but more generally also $a_m \to 0$ should be admitted, i.e., some of the zeros may tend to infinity. In this way we are led to the following problem specification.

(3.1) Computational task of *approximate factorization*: Given any integer $N > 0$ and a polynomial $P \in \Pi_m$ of norm $|P| \leq 1$, compute linear factors $L_j(z) = u_j z + v_j$ $(1 \leq j \leq m)$ such that $|P - L_1 L_2 \cdots L_m| < 2^{-N}$ is satisfied.

Here and in the sequel $|\cdot|$ stands for the l^1-norm of the coefficient vectors. Using other common norms like the l^2-norm or $\max_{|z|=1} |p(z)|$ would not make much of a difference, as the trade-off factors are less than $m+1$, which means a variation of N by not more than $\log(m+1)$.

In analyzing the efficiency of algorithms for this task we are mainly interested in their *worst case behavior* (note that the 'worst' case need not be really 'bad'). For any such algorithm and for fixed values of the parameters m, N, there is a well-defined maximal running time $T(m, N)$, maximal with respect to all inputs (oracles) $(a_0, \ldots, a_m)$ of l^1-norm ≤ 1. (If this maximum would not exist, then König's lemma would imply the existence of certain inputs for which the algorithm would never stop.) Now we can restate the main problem of this section more precisely: What is the true order of growth of the maximal running time $T(m, N)$ of (nearly) optimal algorithms for approximate factorization, what is the asymptotic behavior of $T(m, N)$ for $N \to \infty$, or for increasing degree? Most likely the answers will depend on the underlying machine model to a certain extent. In the light of our incomplete knowledge about much simpler complexity problems like integer multiplication we cannot hope for more than partial answers here. Our main result, presented below, is a good upper bound for the complexity of approximate factorization, but before going into this, we shall briefly review some of the partial answers which, at least implicitly, are contained in the vast literature on root finding methods.

In 1967 a symposium on constructive aspects of the fundamental theorem of algebra was held; the proceedings [**11**] provide a good record of the state of the art at that time. Furthermore, Chapter 6 of Henrici's book [**14**] gives an extensive account of existing techniques. Many of the numerical methods used in practice try to compromise between speed and universal applicability. A major difficulty arises from the fact that all reliable algorithms for approximate factorization must necessarily employ multiprecision techniques which are usually slow, and rigorous a priori estimates for the round-off errors are required in order to guarantee reliable results. The same applies to all iterative methods usually posing the additional difficulty of finding suitable starting values.

By adapting the ideas found in the numerical literature to the strict conventions explained before we arrive at the conclusion that approximate factorization is certainly possible in *polynomial time*. On the base of Weyl's exclusion method as worked out by Henrici [**14**, pp. 517–522] we can, for instance, obtain the estimate $T(m, N) = O(m^7 N^3)$. Similar time bounds appear for methods of root isolation developed in Computer Algebra (see [**8**] and the references given there). There remains the challenge of finding more precise bounds. Application of the fast multiplication techniques for integers and polynomials (cf. 2.1 and 2.2) reduces the bound mentioned to something like $O(m^{5+\delta}N^{2+\delta})$ for any $\delta > 0$, but the factor N^2 seems to be unavoidable with these methods.

3.2. *An upper bound.* The main result to be presented here is a good upper bound on the time complexity of approximate factorization found by the author in 1981/82. It is based upon a new technique called the *splitting circle method* ("Trennkreisverfahren" in German), described in the Preliminary Report [**38**]. Some of the material is highly technical, and many details of its implementation need to be worked out even more thoroughly. A full account of the new results will come out as a monograph. In the sequel we shall give a brief outline of the main ideas and results.

THEOREM. *Approximate factorization of complex polynomials as specified in* (3.1) *is possible within maximal running time*

$$T(m,N) = O(m\mu(m^2 \log m) + m\mu(mN)), \tag{3.2}$$

with respect to the machine models and the corresponding time bounds on integer multiplication listed in (2.3).

In order to simplify the presentation we shall restrict further discussion to the case of pointer time with the linear bound $\mu(N) = cN$. (In case of the other models additional logarithmic factors come in.) Then the upper bound becomes $T(m,N) = O(m^3 \log n + m^2N)$. It will be instructive to compare this with obvious time bounds for the corresponding verification problem. In doing so we may assume, as a general rule of thumb, that computations with polynomials of degree of order m should be carried out with m-bit precision at least, because of the following theoretical observations.

First of all, one should be aware of the fact that small polynomials may have big factors. For $m = 2k$ the simple example $P(z) = 2^{-k}(z^2 - i)^k = 2^{-k}(z-a)^k(z+a)^k$ (with $a^2 = i$) shows a partial factorization $P = FG$ with $|P| = |F| = 1$, but $|G| = 2^{m/2}$, thus the corresponding amplification of errors previously made in the computation of F will necessitate an extra precision of $m/2$ bits. The precise upper bound in the case of many factors is contained in the following *quantitative supplement to the fundamental theorem of algebra*—usually not found in textbooks!

$$\text{Let } f \in \Pi_m, \quad f = f_1 \cdots f_k; \qquad \text{then } |f| \le |f_1| \cdots |f_k| \le 2^{m-1}|f|. \tag{3.3}$$

The upper bound is attained for such tame polynomials as $f(z) = z^m - 1$ in case of its complete factorization, i.e., for $k = m$.

Similar error amplification may be caused by *Taylor shifts* (for fixed a the linear mapping defined on Π_m by $g(z) = f(z+a)$ has operator norm $(1+|a|)^m$), or by the numerical division of polynomials, see (2.5). Assuming $N \ge m$ we thus see that any naive method just for *checking* the accuracy of a given approximate factorization will require $O(m^4)$ bit operations at least, and even by means of fast multiplication techniques we cannot do better than in pointer time of order $Nm \log m$, which in fact seems to be the precise order of growth for the complexity of this verification problem.

Therefore the bound (3.2) seems to be optimal up to a factor of order m at most. For increasing accuracy the second term $O(m^2N)$ becomes dominant, and for a wide class of polynomials admitting *balanced splittings* (explained below) it can be replaced by the smaller bound $O(Nm \log m)$. The main advantage with all these bounds is their linearity in N. For any *fixed* degree m the bound (3.2) becomes simply $O(\mu(N))$ (with the implicit constants depending on m), and this is indeed optimal. For pointer machines we arrive at a linear time bound, not more than a constant multiple of the time needed to output the results. But even in the case of machine models, for which the precise complexity of integer multiplication is yet unknown, we have the following relative result.

(3.4) THEOREM. *For $N \to \infty$ and fixed $k, m \geq 2$ the complexity bounds for the following tasks all have the same order of growth*:

(a) *N-bit integer multiplication,*

(b) *computing $(1+x)^{1/k}$ for $|x| \leq 1/2$ within error bound 2^{-N},*

(c) *approximate factorization of mth degree polynomials within error 2^{-N}.*

For bounding (c) by (a) again some regularity condition is assumed, as needed for (2.2), and then (3.2) applies. Approximate factorization of $z^k-(1+x)$ reduces (b) to (c) for $k \leq m$, and in general via (a). The transition from (b) to (a) is by an idea due to H. Alt, to simulate integer squaring by means of a sufficiently precise evaluation of

$$(2-(1+x)^{1/k}-(1-x)^{1/k})k^2/(k-1) = x^2 + O(x^4)$$

for small x.

In §3.4 we will provide the perturbation argument needed to infer from an approximate factorization to approximate *solutions* (the zeros) of a polynomial equation, with leading coefficient one and bounded coefficients, say. In case of fixed degree this will amplify the time bounds by not more than another constant factor. Therefore the equivalence of (b) and (c) shows that, in contrast to Abel's and Galois's findings on "solvability," in terms of computational complexity there is no significant difference between solving pure or general equations with complex coefficients. Computationally, the approximate solution of general polynomial equations is reduced to the obviously more fundamental task of integer multiplication.

3.3. *The splitting circle method.* The general strategy for the approximate factorization of a given $P \in \Pi_m$ is based on a substrategy for approximately splitting P into two factors $F \in \Pi_k$, $G \in \Pi_{m-k}$, to be applied recursively. The case $k = 1$ simply means approximate determination of a single zero and its deflation. High accuracy can be achieved fast by means of Newton iteration, provided the zero is simple, well isolated, and a suitable starting value is known. Similarly the case $k = 2$ is covered by Bairstow's method, quadratically convergent, if the two zeros are well separated from the $m-2$ other zeros. The splitting circle method employs the corresponding general Newton iteration (cf. J. Schröder's paper in [**11**]), where the choice of a suitable k will depend on the distribution of the zeros, such that there are k zeros of P *inside* of some suitable *splitting circle* while the $m-k$ other zeros lie *outside* of this circle, possibly some of them close to infinity. Moreover all the zeros shall stay at some distance of the circle. It is a fundamental fact that such a circle can always be determined, except for the singular cases $P(z) \approx (az+b)^m$ which will be found out properly to be approximately factored already.

There are stable transformations (scalings, Taylor shifts), by which any circle can be reduced to the standard case of the unit circle $E = \{z\colon |z| = 1\}$. Due to the prior condition of a zero free annulus around E it is possible to obtain a reasonable lower bound on the decisive quantity $\nu = \min_{|z|=1} |P(z)|$, both theoretically and computationally with a rather moderate amount of work. The details in choosing such splitting circles can be arranged such that always $\log(1/\nu) \leq O(m)$. Based on this ν the analysis of the general Newton method then leads to explicit bounds for its quadratic convergence in the following way.

Given some approximate unit circle splitting $|P-FG| < \varepsilon$ (i.e., the k roots of F and the reciprocals of the $m-k$ roots of G are bounded by $1-\delta$), the *Newton correction* is a pair $(f,g) \in \Pi_{k-1} \oplus \Pi_{m-k-1}$ to be chosen such that terms up to first order will cancel each other. Hence the new approximate factors $F+f$ and $G+g$ will satisfy

$$P-(F+f)(G+g) = P - FG - fG - gF - fg = -fg$$

with the new error being bounded by $|fg| \leq |f|\,|g|$, where f and g are uniquely determined by the (incomplete) partial fraction decomposition $(P-FG)/(FG) = f/F + g/G$. Thus one can use the integral representation

$$f(z) = \frac{1}{2\pi i}\int_E \frac{(P-FG)(t)}{(FG)(t)}\,\frac{F(z)-F(t)}{z-t}\,dt$$

for deriving an upper bound for $|f|$, and similarly for $|g|$. In this way one shows, in particular, that an initial approximate splitting with precision

$$|P - F_0G_0| < \varepsilon_0 = \nu^4 2^{-cm} \quad \text{with some constant } c$$

will suffice.

Finding such initial F_0 and G_0 is possible by means of contour integration (also used in [**12**], though without error bounds). The power sums s_j of the k zeros of P which lie inside of E are expressible as

$$s_j = \frac{1}{2\pi i}\int_E \frac{P'(z)}{P(z)} z^j\,dz.$$

Here the zero free annulus and the lower bound ν enable us to give explicit bounds for the precision required for the simultaneous evaluation of these integrals by means of a discrete Fourier transform. (This step essentially contributes to the first term of the upper bound (3.2).) The approximations for $s_0 = k$, $s_1, \ldots, s_k$

easily yield an approximate factor F_0, and a suitable G_0 can then be found by polynomial division.

So far we have tacitly assumed that the zeros of the given polynomial P can (in advance) be localized well enough to admit a proper choice of the splitting circles, but this is actually the key problem of the whole method. Our algorithms for the corresponding *diagnostics* are based on the classical Graeffe process of *root squaring.* We use a variant especially suited to exploiting fast integer multiplication. Contrary to the common application of Graeffe's method, here the primary interest is not in circles *on* which the zeros are located but in circles bounded away from all the zeros, and for the latter purpose moderate precision is quite sufficient. Moreover one should not try to compute all the root moduli $\rho_1(f) \geq \rho_2(f) \geq \cdots \geq \rho_m(f)$ of some f simultaneously, which indeed may require prohibitive (i.e., too expensive) long number computations (cf. the discussion in Turán's paper [**52**]). Instead, one can shift the *focus of precision* by suitable scalings after each root squaring step such that the algorithm is directed to the computation of one particular $\rho_k(f)$. The following time bound is a typical result obtained in this way.

(3.5) Given any complex polynomial $f(z) = a_0 + a_1 z + \cdots + a_m z^m$ with $|f| \leq 1$, given some positive $\delta < 1/2$ and an index $k \leq m$, computing the kth root modulus $\rho_k(f)$ with relative error less than δ is possible in pointer time $O(m^2(\log m + \log(1/\delta))(\log\log m + \log(1/\delta)))$. (In addition, something like $|a_0|, |a_m| \geq (\delta/m)^m$ should hold.)

Applied with $\delta = m^{-O(1)}$ this leads to the rather favorable time bound $O(m^2 \log^2 m)$, while this approach cannot be recommended for much higher precision requirements, where the quadratic growth in $\log(1/\delta)$ becomes dominant.

For other similar estimates and further details we must refer to [**38**]. The overall time analysis of the splitting circle method reveals that balanced splittings (with k and $m - k$ of the same size, ideally $k = m/2$) are preferable. Even in the case of m well isolated simple zeros the deflation of one linear factor after the other is inferior compared with the *divide and conquer* principle of computational complexity, but there do exist certain hard polynomials which, by their particular zero spacing, will enforce very unbalanced splittings. A whole family of such examples (with arbitrary parameters $1 < u_j < 1.01$) is furnished by

$$f(z) = (z + u_1/4)(z + u_2/16)(z + u_3/64) \cdots (z + u_m/4^m).$$

Here $N = 2m^2$ specifies a precision well tuned with respect to the coefficient size. This property of being *strongly nested* represents a new phenomenon in the numerical treatment of polynomials. At present it is hard to say whether such polynomials are really ill-conditioned with regard to the task of approximate factorization, or whether this just shows a particular weakness of the splitting circle method. In any case this nestedness should properly be distinguished from the occurrence of clustered zeros which, in itself, does not cause extra difficulties for the approximate factorization of a polynomial, thanks to a blow-up technique by proper scaling (see [**38**]). The presence of clusters will, however, affect the *stability* of the zeros, to be discussed below.

3.4. *Some applications.* Any approximate factorization $|P - L_1 \cdots L_m| < \varepsilon$ with $0.5 \le |P| \le 1$ and known linear factors $L_j(z) = u_j z + v_j$ can be used for the approximate determination of the zeros of P. The natural candidates for this approximation are the numbers $w_j = -v_j/u_j$; in case of $|v_j| > |u_j|$ it may be preferable to consider their reciprocals as approximations for the reciprocals of the zeros of P.

The corresponding inequalities $|P(w_j)| < \varepsilon$ (or $|P^*(1/w_j)| < \varepsilon$ with the reversed polynomial $P^*(z) = z^m P(1/z)$) combined with a well-known homotopy argument induce $W = \{z\colon |P(z)| < \varepsilon \max(1, |z|^m)\}$ as an *inclusion set* for the zeros, and their perturbations are bounded by the diameters of the components of W. Here we want to consider (worst case) a priori bounds only, while in practical applications one should, of course, take advantage of possible shortcuts based on better a posteriori bounds. Keeping to the previous notations we have the following perturbation bound (Theorem 2.7 in [**40**]).

(3.6) The zeros $z_1, \ldots, z_m$ of P can be numbered such that (for $\varepsilon < 2^{-7m}$)

$$|z_j - w_j| < 9\varepsilon^{1/m} \quad \text{for } |w_j| \le 1, \qquad |1/z_j - 1/w_j| < 9\varepsilon^{1/m} \quad \text{for } |w_j| \ge 1.$$

This is optimal up to the constant factor, and is better by a factor of order m than corresponding bounds given by Ostrowski (cf. [**29**, Appendices A, B]). Aiming at an error bound of 2^{-n} for the roots of some mth degree polynomial we see that approximate factorization with $\varepsilon = 2^{-N}$, $N = (n+4)m$ will always suffice, whence the time bound (3.2) yields

(3.7) Computing the zeros (or their reciprocals) of any mth degree polynomial P (of norm $0.5 \le |P| \le 1$, say) with error less than 2^{-n} is possible in pointer time $O(m^3 \log m + m^3 n)$.

There are applications, where the determination of just one zero will suffice. For that purpose the splitting circle method can be modified such that the second term in the time bound of (3.7) may be replaced by the smaller bound $O(m^2 n)$, since after a first approximate splitting of P into two factors F and G, only one of these (of degree $\le m/2$) needs further treatment, etc., recursively. Accordingly, in this case unbalanced splittings are most welcome.

As an example of this kind we like to mention a rather fast method for the factorization of univariate integer polynomials based on high precision computation of one zero and on diophantine approximation to find its minimal polynomial as a factor (see [**26**] and [**39**]). Along these lines extensive numerical calculations with *algebraic numbers* seem to become feasible.

Another important consequence of the above time bounds is that *all algebraic functions can be computed fast.* In their paper [**20**] with exactly this title, Kung and Traub investigate the complexity of the somewhat different task to compute an initial segment of one of the (fractional) power series expansions of an algebraic function, counting arithmetical operations at unit cost, where finding a zero of a numerical polynomial (to any prescribed precision) is regarded as a primitive operation.

In our model this task cannot be solvable in general, since testing for equality is impossible. Let us consider a more direct approach, instead. Given any polynomial $F(z,v) = a_0(v)+a_1(v)z+\cdots+a_m(v)z^m$ and some point $v = (v_1,\ldots,v_k)$ i.e., the coefficients of the polynomials $a_j \in \mathbf{C}[v_1,\ldots,v_k]$ and the point v are available from oracles again, the m branches of the algebraic function $z(v)$ defined by $F(z,v) = 0$ can be evaluated at v within time bounds of the same order as in (3.7), provided some normalizing assumptions about v and the a's are made. The major strength of this assertion lies in its *uniformity* with respect to v; the hidden constants can be chosen, for instance, to be valid for all $|v| \leq 1$.

4. Linear equations and the complexity of matrix computations.

One of the pioneering results in early complexity theory was Strassen's discovery [**46**] that "Gaussian elimination is not optimal." He showed how to replace the classical $O(n^3)$ methods of matrix multiplication and inversion by algorithms which require less than $O(n^{2.81})$ arithmetical operations. The basic idea is quite elementary and widely known nowadays, though most numerical analysts have remained sceptical about the practical applicability of such methods.

Meanwhile the exponent 2.81 has been improved substantially by the efforts of several authors (cf. [**3**, **36**, **10**] and Pan's survey [**30**]). The best bound presently known to the author is a bit smaller than 2.4785, presented by V. Strassen at an informal meeting in 1985 (see also [**51**]). Actually these bounds concern the *complexity of matrix multiplication.* In §§4.1 and 4.2 we will briefly review some of the main ideas and results related to this fascinating unsolved mathematical problem; possibly this outline could stimulate other mathematicians to contribute to its complete solution.

In §4.3 we will discuss the complexity of solving linear systems and of other matrix computations like matrix inversion, evaluating determinants, etc., mainly in the framework of algebraic complexity, where the elements of matrices and vectors are considered as indeterminates over some ground field F. When dealing with real or complex numbers, we have to include numerical aspects again, especially for the complexity bounds on the approximate determination of eigenvalues.

With regard to other domains we should mention here that linear systems of diophantine equations can be solved in polynomial time as shown by J. von zur Gathen and M. Sieveking (in [**45**, pp. 57–65]). Related algorithms and further references are found in [**7**] and [**16**].

4.1. *The exponent of matrix multiplication.* The algebraic complexity of matrix multiplication can be measured in different ways. We consider $n\times n$ matrices $A = (a\ldots)$, $B = (b\ldots)$ with indeterminate entries over some scalar field F (to be held fixed for the moment). Let $L(n)$ denote the minimal length of straight-line programs in $F(a\ldots,b\ldots)$ computing the entries of AB, where all arithmetical operations are counted, and let $L^{*/}(n)$ be the corresponding nonscalar complexity, or $L^*(n)$ in case of $F[a\dots,b\dots]$, if divisions shall not be used. For infinite F we have $L^{*/}(n) = L^*(n)$ (cf. [**47**]).

Last but not least, the *bilinear complexity* $L^{\otimes}(n)$ refers to the model where only multiplications $\xi(a\dots) * \eta(b\dots)$ of F-linear forms in the a's by F-linear forms in the b's are admitted.

The *exponent of matrix multiplication* (over F) is defined as the infimum

$$\omega(F) = \inf\{\beta\colon L(n) = O(n^{\beta})\}, \tag{4.1}$$

obviously satisfying $2 \le \omega(F) \le 3$. Strassen's first nontrivial bound $\omega(F) \le \log 7/\log 2 < 2.81$ was derived from $L^{\otimes}(2) \le 7$. In the same way any bilinear algorithm for some other small n can be applied to multiply $n \times n$ *block matrices* recursively; thus $L^{\otimes}(n) \le r$ implies $L(N) = O(N^{\beta})$ with $\beta = \log r/\log n$. Combined with the elementary inequalities

$$L^{*/}(n) \le L^*(n) \le L^{\otimes}(n) \le 2L^*(n),$$

this shows that we may replace $L(n)$ in (4.1) by any of these other measures without altering the definition of $\omega(F)$. The further discussion will concentrate on $L^{\otimes}(n)$ and its generalization to the multiplication of rectangular matrices. For technical details and proofs see [**36**].

Let $\langle k,m,n\rangle$ denote the *tensor* for the bilinear mapping of multiplying $k \times m$- with $m \times n$-matrices. Its *rank* $\mathrm{rk}\langle k,m,n\rangle$ is equal to the minimal length of representations of the associated trilinear form $\mathrm{trace}(ABC)$ as a sum of *rank one products* of linear forms, where C has format $n \times k$. Spelled out in coordinates this means, with $A = (a\dots)$, $B = (b\dots)$, $C = (c\dots)$, that $\mathrm{rk}\langle k,m,n\rangle \le r$ is equivalent to the existence of linear forms ξ_j in the a's, η_j in the b's, ς_j in the c's such that

$$\sum_{j=1}^{r} \xi_j(a\dots)\eta_j(b\dots)\varsigma_j(c\dots) = \mathrm{tr}(ABC) = \sum_{\kappa,\mu,\nu} a_{\kappa,\mu} b_{\mu,\nu} c_{\nu,\kappa} \tag{4.2}$$

is satisfied, which in turn is equivalent to the solvability of a huge system of $k^2m^2n^2$ nonlinear equations over F for the $kmr+mnr+nkr$ unknown coefficients of the linear forms. The formidable size of such systems may give an impression of the difficulties to determine any of the values of $\mathrm{rk}\langle k,m,n\rangle$, even in the case of low dimensions (despite the fact that, for fields like $\mathbf{R}$ or $\mathbf{C}$ with decidable first order theory, $\mathrm{rk}\langle k,m,n\rangle$ is a *computable* function). One of the rare values explicitly known is $\mathrm{rk}\langle 2,2,2\rangle = 7$ for any field F, as always $\mathrm{rk}\langle n,n,n\rangle = L^{\otimes}(n) \ge L^*(n) \ge 2n^2 - 1$; the latter inequality is an instance of a more general lower bound on the complexity of associative algebras (see [**1**, **50**]). The next higher case of 3×3 matrices would already amount to a system of 729 equations with 567 unknowns (for $r = 21$); so far nothing better than $\mathrm{rk}\langle 3,3,3\rangle \le 23$ is known. More generally, one could also hope to derive better estimates for $\omega(F)$ from good upper rank bounds for *rectangular* formats, by means of the following lemma.

$$\begin{gathered}\mathrm{rk}\langle k,m,n\rangle \text{ is symmetric in } k,m,n, \text{ and submultiplicative, i.e.,}\\ \mathrm{rk}(\langle k',m',n'\rangle \otimes \langle k'',m'',n''\rangle) \le \mathrm{rk}\langle k',m',n'\rangle \mathrm{rk}\langle k'',m'',n''\rangle.\end{gathered} \tag{4.3}$$

Starting from some bound $\mathrm{rk}\langle k,m,n\rangle \le r$ we apply symmetrization and obtain $L^{\otimes}(kmn) = \mathrm{rk}\langle kmn, mnk, nkm\rangle \le r^3$; therefore,

$$\mathrm{rk}\langle k,m,n\rangle \le r = (kmn)^{\tau} \quad \text{implies} \quad \omega(F) \le 3\tau. \tag{4.4}$$

So far, however, none of the rank bounds available for *small* formats has proved to be good enough to yield better estimates than 2.81. In 1978, this barrier was broken by V. Pan. By means of his technique of *trilinear aggregating* he could show $L^*(n) \le n^3/3 + O(n^2)$ with such constants that $n = 48$ led to the improved estimate $\omega(F) < 2.781$.

A similar bound was obtained by Bini, Capovani, Lotti, Romani [**3**], but they used a completely new approach based on the notion of *border rank*. Let ε be a further indeterminate over F. The border rank $\mathrm{brk}\langle k,m,n\rangle$ can be defined as the minimal length of sum representations (4.2) up to errors $O(\varepsilon)$, where now *meromorphic formal power series* in ε are admitted as coefficients of the linear forms. (For a definition in the language of algebraic geometry see §§13, 14 of [**50**].)

The properties (4.3) and (4.4) are also true for border rank, whence

$$\mathrm{brk}\langle k,m,n\rangle \le r = (kmn)^{\tau} \quad \text{implies} \quad \omega(F) \le 3\tau. \tag{4.4b}$$

In [**3**] this was applied to $\mathrm{brk}\langle 2,2,3\rangle \le 10$; another example is $\mathrm{brk}\langle 3,3,3\rangle \le 21$, which yields $\omega(F) < 2.772$. For the paradigmatic case of 2×2 matrices one knows the bounds $6 \le \mathrm{brk}\langle 2,2,2\rangle \le 7$.

The proof of (4.4b) exploits the fact that the tensorial powers of any tensor t with $\mathrm{brk}\, t \le r$ satisfy a rank estimate $\mathrm{rk}(t^{\otimes s}) \le O(s^2) \cdot r^s$; the extra factor of polynomial growth does not matter when (4.4) is applied for $s \to \infty$. Similar arguments are used to show $\omega(F) = \omega(F_0)$, where F_0 denotes the prime field of F (see [**36**, Theorem 2.8]), thus the exponent of matrix multiplication over F can only depend on the characteristic of F. In the sequel we will simply write ω, since all further discussions are uniform with respect to F.

4.2. *Subadditivity.* There exist computational problems which admit substantial savings under *mass production*. An important example is furnished by the evaluation of a general mth degree polynomial at m different points (cf. [**4**]). Similar effects occur in the realm of matrix computations. Multiplying one column vector b by an $n \times n$ matrix A takes exactly n^2 multiplications, while multiplying n different vectors $b_1, \dots, b_n$ by the same A is possible by one matrix multiplication—in less than $O(n^{2.8})$ steps. One could argue that these examples are relying on a certain amount of joint information (the coefficients of the polynomial, the matrix A). So let us impose very strict rules now. We consider two completely *disjoint problems* of matrix multiplication and ask whether forming AB and UV in one compound computation may be cheaper than by performing (optimal) algorithms for the two matrix multiplications separately.

Under the bilinear cost measure this is equivalent to the question whether the *subadditivity* of *rk*, i.e. (with respect to direct sums),

$$\mathrm{rk}(\langle k',m',n'\rangle \oplus \langle k'',m'',n''\rangle) \le \mathrm{rk}\langle k',m',n'\rangle + \mathrm{rk}\langle k'',m'',n''\rangle, \tag{4.5}$$

can become a strict inequality. For 2-stage tensors (matrices) equality always holds, while the corresponding *additivity conjecture* (cf. [**47**]) for 3-stage tensors is still an open problem.

Subadditivity is true for border rank as well, but in that case we definitely know that additivity does not hold in general. Already for the special class of tensors $\langle k, m, n\rangle$, the left-hand and the right-hand side of (4.5) (with *brk* instead of *rk*) can differ unboundedly. We present an example from [**36**] which has played an important role in estimating the exponent of matrix multiplication. Based on a dimension argument, one has $\mathrm{brk}\langle 3,1,3\rangle = 9$ and $\mathrm{brk}\langle 1,4,1\rangle = 4$, but also

$$\mathrm{brk}(\langle 3,1,3\rangle \oplus \langle 1,4,1\rangle) = 10. \tag{4.6}$$

We show that the trilinear form $\mathrm{tr}(ABC) + \mathrm{tr}(UVW)$ associated with the direct sum of these disjoint problems indeed has an approximate decomposition (4.2) of length 10, with coefficients from $F((\varepsilon))$ and up to an error term $O(\varepsilon)$, namely

$$\sum_{i=1}^{3}\sum_{j=1}^{3}(a_i + \varepsilon u_{i,j})(b_j + \varepsilon v_{i,j})(c_{j,i} + \varepsilon^{-2} w) - \left(\sum_i a_i\right)\left(\sum_j b_j\right)\varepsilon^{-2} w$$
$$= \sum_{i=1}^{3}\sum_{j=1}^{3}(a_i b_j c_{j,i} + u_{i,j} v_{i,j} w) + O(\varepsilon),$$

where the u's and v's with subscripts $(i,j) = (1,1),(1,2),(2,1),(2,2)$ are the indeterminate components of two vectors of length 4, while the remaining u's and v's are chosen to yield proper canceling, namely

$$u_{i,3} = v_{3,j} = 0, \qquad u_{3,j} = -u_{1,j} - u_{2,j}, \qquad v_{i,3} = -v_{i,1} - v_{i,2}.$$

Similar to Strassen's $\mathrm{rk}\langle 2,2,2\rangle \le 7$ versus the trivial number of 8 multiplications, which gave the estimate $\omega \le 3\log 7/\log 8$, we can regard (4.6) as a bound of 10 on the border rank of a problem of *partial* matrix multiplication with the trivial number of 13 multiplications, and this indeed implies $\omega \le 3\log 10/\log 13 < 2.7$, by a corresponding extension of (4.4b) [**36**, Theorem 4.1]. There is, however, an even more productive way of estimating ω on the base of such examples, which in addition exploits the *disjointness* of the pieces. This stronger generalization of (4.4b) is the following

τ-THEOREM (de Groote's naming for a special case of Theorem 7.1 from [**36**]).

$$\mathrm{brk}(\langle k_1, m_1, n_1\rangle \oplus \cdots \oplus \langle k_p, m_p, n_p\rangle) \le r = \sum_{i=1}^{p}(k_i m_i n_i)^{\tau} \quad \textit{implies} \quad \omega \le 3\tau.$$

With regard to (4.6), for instance, we solve the equation $9^{\tau} + 4^{\tau} = 10$ and obtain the better bound $\omega < 2.6$.

The presentation of this theorem at an Oberwolfach conference in 1979 initiated a hot competition among specialists to find better and better estimates from more and more sophisticated designs, similar to (4.6), and the bounds for ω

came closer and closer to 2.5. Corresponding lower bound speculations, however, were soon abandoned by Coppersmith and Winograd (see [**10**]). They found a general method for the iterated construction of such examples, and by starting from (4.6) they succeeded in showing $\omega < 2.5$. Moreover, it is an important theoretical consequence of their work that any particular instance of the τ-theorem admits some further improvement yielding a slightly better bound for ω, i.e., we always have the strict inequality $\omega < 3\tau$.

Strassen's new method [**51**] stems from a thorough investigation of rank and border rank in a more general algebraic setting. His constructions have led to an application of the τ-theorem combined with a limit process which yields $\omega \leq \log((h+1)^3/4)/\log h$ (for $h = 2, 3, \ldots$). The best estimate is obtained for $h = 5$, namely $\omega \leq \log 54/\log 5 < 2.4785$.

4.3. *Linear systems and matrix inversion.* Solving a general system $Ax = b$ of n linear equations in n unknowns $x_1, \ldots, x_n$ can be considered as the task of computing the x's as rational functions in the indeterminate entries of A and b. Optimal algorithms for this task are straight-line programs of minimal length in $F(a\ldots, b\ldots)$, and it was with respect to this model that Gaussian elimination was originally shown not to be optimal. (For $n = 2$, by the way, and under the measure of nonscalar complexity, Gaussian elimination *is* in fact optimal, see [**28**].) The solution of the system can simply be expressed as $x = A^{-1}b$. Thus solving such a general system is possible within the upper bound $I(n) + 2n^2 - n$, where $I(n)$ denotes the minimal number of arithmetical operations sufficient for the computation of the inverse of a general $n \times n$ matrix. In [**46**], 2×2 *block inversion* was applied recursively to reduce matrix inversion to (fast) matrix multiplication. The corresponding estimate is

$$I(2n) \leq 2I(n) + 6L(n) + n^2 + n.$$

Similar reasoning shows that the complexity $D(n)$ of computing $n \times n$ determinants satisfies the recursive inequality

$$D(2n) \leq I(n) + 2L(n) + 2D(n) + n^2 + 1.$$

Therefore $L(n) = O(n^\beta)$ (for any $\beta > \omega$) implies that $I(n)$ and $D(n)$ are of the same order of growth at most. Conversely we can also show that the complexity of matrix inversion is not much smaller than that of matrix multiplication. With such proofs, however, one should keep in mind that straight-line programs valid for the model with indeterminates will usually not work for all specializations by nonsingular matrices, since substitution of algebraically dependent elements may cause divisions by zero. Below we will discuss other models without this deficiency.

The identity $X^2 = (X^{-1} - (I + X)^{-1})^{-1} - X$ shows that the complexity $Q(n)$ of *squaring* a general $n \times n$ matrix satisfies $Q(n) \leq 3I(n) + 2n^2 + n$. The multiplication of general $2n \times 2n$ matrices,

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \begin{pmatrix} B_1 & B_3 \\ B_2 & B_4 \end{pmatrix}$$

with $n \times n$ blocks A_i, B_j is reduced to squarings in the following way. With $S = B_1 + B_2$, the first n columns of the result are obtained from

$$\begin{pmatrix} A_3 - S & B_1 \\ A_1 & S \end{pmatrix}^2 - \begin{pmatrix} A_4 - S & -B_2 \\ A_2 & S \end{pmatrix}^2 = \begin{pmatrix} \cdots\cdots & A_3B_1 + A_4B_2 \\ \cdots\cdots & A_1B_1 + A_2B_2 \end{pmatrix},$$

and equally for the second half. That shows $L(2n) \leq 4Q(2n) + 12n^2$, whence $L(2n) \leq 12I(2n) + O(n^2)$. For infinite fields F a similar estimate (with other constants) can be found in [**47**].

There is also a reduction from general matrix inversion to the computation of the *determinant*, obtained by Baur and Strassen as a corollary to their following general theorem (cf. [**2**]):

(4.7) If $f \in F(x_1, x_2, \ldots, x_k)$ is computable by a straight-line program of length s, then the complexity of computing f together with all its partial first derivatives $\partial f / \partial x_j$ is bounded by $4s$.

Another illustrating example of these reduction techniques is the following quick proof for K. Kalorkoti's (unpublished) result that computing the *trace of the inverse* is as difficult as matrix multiplication, up to a constant factor (over infinite fields), hence essentially of the same complexity as computing the inverse itself. If there is a straight-line program for $\mathrm{tr}(X^{-1})$ of length s, then $4s$ steps suffice for X^{-2} (built up from the partial derivatives); applying this once more we get $8s$ as an upper bound for the complexity of computing X^4. After elimination of the divisions from such a program (at the cost of another constant factor, see [**47**]) we can finally specialize X as a suitable $3n \times 3n$ matrix such that computing X^4 will simulate a general matrix multiplication of size $n \times n$.

It is still an open question, whether the original problem of solving $n \times n$ systems, i.e., computing $A^{-1}b$ for general A, b, is powerful enough to simulate matrix multiplication or any of the other equivalent problems. Quite recently, however, T. Lickteig could show that the more general task of computing *all* solutions of *rectangular* systems $Ax = b$, which means, for instance, to compute a basis for the kernel of a general $n \times 2n$ matrix, has indeed the same order of complexity as matrix multiplication.

So far we have considered problems with algebraically independent inputs only, but with respect to applications other models seem to be preferable. In case of the prime fields with their finite extensions, *computation trees* with branchings based on zero testing furnish an adequate framework for the discussion of solving linear systems, computing the rank of matrices, or the inverse (if it exists), etc. Most of the reductions mentioned before similarly apply to this model. For further applications we refer to [**17**], including the proof that the algebraic complexity of computing the coefficients of the *characteristic polynomial* is of the same order as that of matrix multiplication, up to logarithmic factors.

With the possibility of zero testing, the open problem whether solving $Ax = b$ may have significantly lower complexity than matrix multiplication appears in a new light. The corresponding *verification* problem for a given solution x can obviously be solved within $O(n^2)$ steps. It is also an interesting question whether verification of matrix multiplication, i.e., checking whether $AB = C$ for given C, could possibly be done faster.

The preceding model with zero testing does not apply to the fields **R** or **C** with oracle inputs (cf. §2) where equality is undecidable. It is, for instance, impossible to compute the rank of arbitrary matrices, or to decide whether a given $n \times n$ matrix A has determinant zero, but if we assume the regularity of A, then the fast algebraic methods of matrix multiplications can be utilized for the approximate computation of the inverse. By means of the hermitean transpose we have $A^{-1} = (A^H A)^{-1} A^H$, and for the inversion of positive definite matrices Strassen's block elimination (cf. [**46**]) is stable (provided the underlying method for matrix multiplication does not use divisions). The running time of such machine algorithms will, of course, also depend on the desired accuracy and on the condition of A.

When discussing methods for approximately solving $Ax = b$ one should properly distinguish whether some a priori bound on the condition of A is known or not. In the latter case it is rather likely that the major amount of work will be required to obtain sufficient information about the condition in order to guarantee a certain accuracy of the solution.

4.4. *Characteristic equations.* Finally we want to give a brief outline how to combine the time bounds on root finding from §3 with the preceding results on the algebraic complexity of matrix computations in order to obtain a good upper bound on the complexity of approximately computing the *eigenvalues* of arbitrary complex $m \times m$ matrices. This is just the very beginning of more extensive complexity studies of related problems like the determination of invariant subspaces, or improved time bounds for special classes of matrices, etc., still to be undertaken.

Let A be any $m \times m$ matrix of l^1 operator norm $|A| \leq 1$, or $\leq 1/2m$ after suitable scaling, and assume that the subsequent computations are done with N-bit precision. We describe a method for the computation of the coefficients of the characteristic polynomial of A which will mimic the algebraic approach from [**17**] numerically. Let ω denote the exponent of matrix multiplication over fields of characteristic zero, and let β be any number $\beta > \omega$. Then multiplication of complex $m \times m$ matrices in N-bit precision is possible in pointer time $O(m^\beta N)$. The first step in computing the characteristic polynomial is the approximate *unitary* transformation of A into *upper Hessenberg form* B ($b_{j+d,j} = 0$ for $d \geq 2$). By the methods from [**32**, **33**], that is possible in pointer time $O(m^\beta N)$. Similar to the estimate (3.6), the perturbations of the eigenvalues caused by errors $|B - U^H A U| < \varepsilon$ are bounded by $O(\varepsilon^{1/m})$ in the worst case.

The next step is *scaling* by a similarity transformation with a diagonal matrix D such that $M = D^{-1}BD$ has only ones in its subdiagonal (without restriction we may assume that all the subdiagonal elements of B are nonzero; otherwise the problem would split into several smaller eigenvalue problems). Thus we get $M = S + R$, where R denotes the upper triangle of M, while the subdiagonal S represents a shift operator which maps the jth unit vector e_j upon e_{j+1}, with

$Se_m = 0$. The new entries

$$r_{k,k+d} = b_{k,k+d} \sum_{0 \le j < d} b_{k+j+1,k+j} \qquad (1 \le k \le k+d \le m)$$

can be computed in a stable way by means of $O(m^2)$ multiplications of complex numbers bounded by one. Moreover we have $|R| \le |B| \le 1/2m$. The characteristic polynomial $f(z) = v_0 + v_1 z + \cdots + v_{m-1} z^{m-1} + z^m$ of M satisfies $f(M) = 0$. Applied to the first unit vector e_1 this equation yields $v_0 x_1 + v_1 x_2 + \cdots + v_{m-1} x_m = -x_{m+1}$, where $x_j = M^{j-1} e_1$. Because of the dominance of S over $|R| \le 1/2m$ one easily finds the estimate $|x_j - e_j| < e^{0.5} - 1 < 0.65$. Therefore the matrix X built up from the column vectors $x_1, \ldots, x_m$ satisfies $|X - I| < 0.65$, its inverse is bounded by $|X^{-1}| < 3$, whence the coefficient vector v can be obtained as $v = X^{-1}(-x_{m+1})$, numerically in pointer time $O(m^\beta N)$. The main idea from [**17**] concerns an efficient computation of the iterates x_j for $j \le m+1$ by means of $O(\log m)$ matrix multiplications, first forming $M^2, M^4, \ldots$, then $M^q(x_1, \ldots, x_q) = (x_{q+1}, \ldots, x_{2q})$ for $q = 1, 2, 4, 8, \ldots$ recursively.

On the whole we see that pointer time $O(m^\beta N)$ is sufficient for the approximate computation of the characteristic polynomial of A such that the eigenvalue perturbations will be bounded by $O(2^{-N/m})$. Therefore, final application of (3.7) yields the following result (cf. [**38**]).

THEOREM. *Approximate determination of the eigenvalues of any complex $m \times m$ matrix (of norm ≤ 1) within error 2^{-n} is possible in pointer time*

$$O(m^\beta mn) + O(m^3 \log m + m^2 mn) \quad (\textit{for any } \beta > \omega). \tag{4.8}$$

By comparing the two terms in this time bound we see (even with β close to two) that finding the eigenvalues from the characteristic polynomial is always faster than computing its coefficients, although the latter part does not cost much more than matrix multiplication, as far as the exponents are concerned.

The factor mn stands for the order of the precision N required under worst case assumptions. In practice, however, one will clearly try to get along with lower precision first, hoping for better a posteriori estimates which then may suffice to justify the low precision.

REFERENCES

1. A Alder and V. Strassen, *On the algorithmic complexity of associative algebras*, Theoret. Comput. Sci. **15** (1981), 201–211.

2. W. Baur and V. Strassen, *The complexity of partial derivatives*, Theoret. Comput. Sci. **22** (1983), 317–330.

3. D. Bini, M. Capovani, G. Lotti, and F. Romani, $O(n^{2.7799})$ *complexity for matrix multiplication*, Inform. Process. Lett. **8** (1979), 234–235.

4. A. Borodin and I. Munro, *Computational complexity of algebraic and numeric problems*, American Elsevier, New York, 1975.

5. A. L. Chistov and D. Yu. Grigoryev, *Polynomial-time factoring of the multivariable polynomials over a global field*, LOMI preprint E-5-82, Leningrad, 1982.

6. ——, *Subexponential-time solving systems of algebraic equations*. I, II, LOMI Preprints E-9-83, E-10-83, Leningrad, 1983.

7. T. J. Chou and G. E. Collins, *Algorithms for the solution of systems of linear diophantine equations*, SIAM J. Comput. **11** (1982), 687–708.

8. G. E. Collins and R. Loos, *Real zeros of polynomials*, Computing, Suppl. **4** (1982), 83–94.

9. S. Cook, *A taxonomy of problems with fast parallel algorithms*, Inform. and Control **64** (1985), 2–22.

10. D. Coppersmith and S. Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput. **11** (1982), 472–492.

11. B. Dejon and P. Henrici (Editors), *Constructive aspects of the fundamental theorem of algebra* (Proc. Sympos., Zürich-Rüschlikon, 1967), Wiley, London, 1969.

12. L. M. Delves and J. N. Lyness, *A numerical method for locating zeros of an analytic function*, Math. Comp. **21** (1967), 543–560.

13. J. von zur Gathen, *Hensel and Newton methods in valuation rings*, Math. Comp. **42** (1984), 637–661.

14. P. Henrici, *Applied and computational complex analysis*, vol. 1, Wiley-Interscience, New York, 1974.

15. E. Kaltofen, *Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization*, SIAM J. Comput. **14** (1985), 469–489.

16. R. Kannan, *Solving systems of linear equations over polynomials*, Theor. Comput. Sci. **39** (1985), 69–88.

17. W. Keller-Gehrig, *Fast algorithms for the characteristic polynomial*, Theor. Comput. Sci. **36** (1985), 309–317.

18. D. E. Knuth, *The art of computer programming*, Vol. 2, Seminumerical Algorithms, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

19. H. T. Kung, *On computing reciprocals of power series*, Numer. Math. **22** (1974), 341–348.

20. H. T. Kung and J. F. Traub, *All algebraic functions can be computed fast*, J. Assoc. Comput. Mach. **25** (1978), 245–260.

21. S. Landau, *Factoring polynomials over algebraic number fields*, SIAM J. Comput. **14** (1985), 184–195.

22. S. Landau and G. L. Miller, *Solvability by radicals is in polynomial time*, Proc. 15th Annual ACM Sympos. on Theory of Comp., Ass. Comp. Mach., New York, 1983, pp. 140–151.

23. D. Lazard, *Résolution des systèmes d'équations algébriques*, Theor. Comput. Sci. **15** (1981), 77–110.

24. A. K. Lenstra, *Factoring multivariate integral polynomials*, Theor. Comput. Sci. **34** (1984), 207–213.

25. ——, *Polynomial-time algorithms for the factorization of polynomials*, Ph.D. Thesis, Univ. of Amsterdam, 1984.

26. A. K. Lenstra, H. W. Lenstra, and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), 515–534.

27. T. Lickteig, *The computational complexity of division in quadratic extension fields*, SIAM J. Comput. (to appear).

28. ——, *Gaussian elimination is optimal for solving linear equations in dimension two*, Inform. Process Lett. **22** (1986), 277–279.

29. A. Ostrowski, *Solution of equations and systems of equations*, 2nd ed., Pure and Applied Mathematics, vol. 9, Academic Press, New York, 1966.

30. V. Ya. Pan, *How can we speed up matrix multiplication?*, SIAM Rev. **26** (1984), 393–415.

31. A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Inform. **1** (1971), 139–144.

32. ——, *Unitäre Transformationen grosser Matrizen*, Numer. Math. **20** (1973), 409–417.

33. ——, *Fast Schmidt orthogonalization and unitary transformations of large matrices*, Complexity of Sequential and Parallel Numerical Algorithms (J. F. Traub, ed.), Academic Press, New York, 1973, pp. 283–291.

34. ——, *Schnelle Multiplikation von Polynomen über Körpern der Charakteristik* 2, Acta Inform. **7** (1977), 395–398.

35. ——, *Storage modification machines*, SIAM J. Comput. **9** (1980), 490–508.

36. ——, *Partial and total matrix multiplication*, SIAM J. Comput. **10** (1981), 434–455.

37. ——, *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients*, Computer Algebra (Marseille, 1982), Lecture Notes in Comput. Sci., Vol. 144, Springer, Berlin-New York, 1982, pp. 3–15.

38. ——, *The fundamental theorem of algebra in terms of computational complexity*, Technical Report, Univ. Tübingen, 1982, 74 pp.

39. ——, *Factorization of univariate integer polynomials by diophantine approximation and an improved basis reduction algorithm*, Automata, Languages and Programming (ICALP, Antwerp, 1984), Lecture Notes in Comput. Sci., Vol. 172, Springer, Berlin-New York, 1984, pp. 436–447.

40. ——, *Quasi-GCD computations*, J. of Complexity **1** (1985), 118–137.

41. A. Schönhage and V. Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing **7** (1971), 281–292.

42. M. Sieveking, *An algorithm for division of power series*, Computing **10** (1972), 153–156.

43. S. Smale, *The fundamental theorem of algebra and complexity theory*, Bull. Amer. Math. Soc. (N.S.) **4** (1981), 1–36.

44. ——, *On the efficiency of algorithms of analysis*, Bull. Amer. Math. Soc. (N.S.) **13** (1985), 87–121.

45. E. Specker and V. Strassen, *Komplexität von Entscheidungsproblemen*, Lecture Notes in Comput. Sci., Vol. 43, Springer, Berlin-New York, 1976.

46. V. Strassen, *Gaussian elimination is not optimal*, Numer. Math. **13** (1969), 354–356.

47. ——, *Vermeidung von Divisionen*, J. Reine Angew. Math. **264** (1973), 184–202.

48. ——, *Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten*, Numer. Math. **20** (1973), 238–251.

49. ——, *The computational complexity of continued fractions*, SIAM J. Comput. **12** (1983), 1–27.

50. ——, *Algebraische Berechnungskomplexität*, Perspectives in Mathematics, Anniversary of Oberwolfach 1984 (W. Jäger, J. Moser, and R. Remmert, eds.), Birkhäuser, Basel, 1984, pp. 509–550.

51. ——, *Relative bilinear complexity and matrix multiplication*, Manuscript, Universität Zürich, 1986.

52. P. Turán, *Power sum method and an approximative solution of algebraic equations*, Math. Comp. **29** (1975), 311–318.

53. H. Weyl, *Randbemerkungen zu Hauptproblemen der Mathematik.* II, *Fundamentalsatz der Algebra und Grundlagen der Mathematik*, Math. Z. **20** (1924), 131–150.

UNIVERSITÄT TÜBINGEN, D-7400 TÜBINGEN, FEDERAL REPUBLIC OF GERMANY