Theoretical Computer Science 36 (1985) 309-317
North-Holland

FAST ALGORITHMS FOR THE CHARACTERISTIC POLYNOMIAL

Walter KELLER-GEHRIG
Dietzingerstrasse 12, 8003 Zürich, Switzerland

Communicated by A. Schönhage
Received November 1982
Revised May 1984

Abstract. Computing the coefficients of the characteristic polynomial is about as hard as matrix multiplication.

1. Introduction

Since the development of the first fast matrix multiplication algorithm, fast matrix multiplications have been used for fast computations of other problems [3, 6, 7]. So, for example, the inverse or the determinant of an $n \times n$ matrix may be computed in $O(M(n))$ time if two $n \times n$ matrices may be multiplied with $M(n)$ nonscalar multiplications and divisions, where $M(n) = g(n)n^{\alpha}$ with $\alpha > 2$ and g monotonically increasing. Computing the coefficients of the characteristic polynomial with classical algorithms costs $O(n^3)$ nonscalar multiplications/divisions and with an algorithm by Winograd, $O(g(n)n^{\alpha+1/2})$ [2, 5]. In this paper we give a straight line algorithm for the computation of the characteristic polynomial with a time bound of $O(M(n))$. On the other hand, if there is an algorithm to compute the coefficients of the characteristic polynomial of an $n \times n$ matrix with cost $F(n)$, two $n \times n$ matrices may be multiplied with cost $O(F(n))$ (this follows from [1]).

In this paper we give three algorithms. All compute the coefficients of the characteristic polynomial. For simplicity we call them the simple, the fast, and the branching algorithms. The fast algorithm has a time bound of $O(M(n))$, the other two algorithms have a time bound of $O(M(n) \log n)$.

The main idea of all three algorithms is to transform the input matrix A by a similarity transformation to Frobenius form. If we do not have to deal with special matrices, we transform to simple Frobenius form:

$$A \to U^{-1}AU = \begin{array}{|c|c|} \hline 0 & \\ \hline 1 & * \\ \hline \end{array}$$

$n-1$ 1

Indeed, the coefficients of the last column of $U^{-1}AU$ are up to the sign equal to the coefficients of the characteristic polynomial.

If we interpret this transformation as a base transformation of k^n from the unit vectors $(e_1, e_2, \ldots, e_n) = e$ to the base $e' = (e'_1, e'_2, \ldots, e'_n) = eU$, this means that $U = e' = (e'_1, Ae'_1, A^2e'_1, \ldots)$ and the characteristic polynomial essential describes $A^n e'_1$ as a linear combination of the e'_i.

In this paper, $\log n$ denotes the binary logarithm of n; 1 the unity of k, of the integers, or of a (square) unity matrix; 0 the zero of the integers, of k, or of a (rectangular) zero matrix; and $\lceil c \rceil$ the smallest integer not smaller than c.

2. The two computational models

In this paper we use two computational models: the model of straight line algorithms and the model of branching algorithms.

A straight line algorithm computes an m-tuple of rational functions in n variables over a field k. An algorithm is a sequence of (g_i), where g_i is a product or a quotient of linear functions in the variables, and the g_j, with $j < i$, and the rational functions (the results) are linear in the variables and the g_i. r is the cost or time of the algorithm. That means that only nonscalar operations are counted.

A branching algorithm may take decisions of the form

$$\text{IF } A = B \text{ THEN} \ldots \text{ELSE} \ldots$$

free of charge and hence gets tree structured. It evaluates a function f on a set M of inputs. Counted operations are division and multiplication if neither of the factors is a constant (nonscalar operations). The cost of the algorithm is a function on M. In this paper we only consider worst-case complexity, that is the maximum of the cost function on M.

The following theorem shows that for rational functions both models have the same time bounds on nearly all inputs.

Theorem 2.1. *Let k be an infinite field, and let f be an m-tuple of rational functions in n variables over k. Then there is a straight line algorithm over k that computes f in time t if and only if there is a nonempty $M \subset k^n$ open in the Zarisky Topology of the k^n and a branching algorithm that computes $m \mapsto f(m)$ for $m \in M$ in time t* [8].

In this paper we only count nonscalar operations, but our theorems still hold if one counts all operations (and tests). For simplicity we have defined $M(n) = g(r)n^\alpha$ with $\alpha > 2$. But if two $n \times n$ matrices may be multiplied with $M'(n) = n^2 g'(n)$ such that g' is monotonically increasing, then the time bound for the fast algorithm is $O(M'(n) \log n)$, and $O(M'(n) \log^2 n)$ for the two other algorithms.

3. The simple algorithm

Let A be an $n \times n$ matrix with indeterminate coefficients and $e \neq 0$ a vector from k^n. Then $U = (e, Ae, A^2e, A^3e, \ldots, A^{n-1}e)$ is a regular $n \times n$ matrix and $U^{-1}AU$ has simple Frobenius form.

Now let $k = \lceil \log n \rceil - 1$ and compute the following matrices:

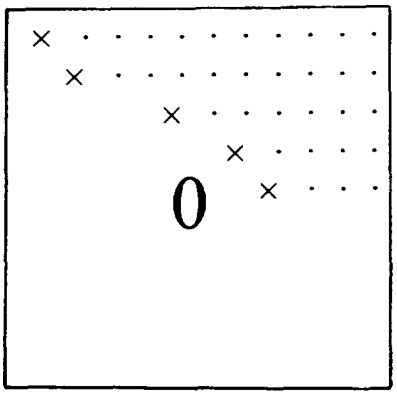
$$
\begin{aligned}
&A, A^2, A^{2^2}, A^{2^3}, \ldots, A^{2^k},\\
&Ae && = Ae,\\
&(A^3e, A^2e) && = A^2(Ae, e),\\
&(A^7e, A^6e, A^5e, A^4e) && = A^4(A^3e, A^2e, Ae, e),\\
&&&\vdots\\
&(A^{2\cdot 2^k-1}e, \ldots, A^{2^k}e) && = A^{2^k}(A^{2^k-1}e, \ldots, e).
\end{aligned}
$$

In this way we compute U and hence the coefficients of the characteristic polynomial in $O(\log n\, M(n))$ time.

4. A fast version of Gaussian elimination

In this chapter we describe an algorithm by Schönhage [6, and private communication, modified], that we need in Section 5 for the branching algorithm. This algorithm transforms a matrix A to step-form as follows:

$A \to UA =$



with elements not 0 in the positions marked with $\times$ and U regular.

Algorithm p_1. If $m \leq n \leq 2m$, B an $(n-m) \times m$ matrix, R_1 an upper triangular $m \times m$ matrix, and

$$A = \begin{pmatrix} B \\ R_1 \end{pmatrix},$$

then p_1 computes an $n \times n$ matrix U and an upper triangular $m \times m$ matrix R_2 with

$$UA = \begin{pmatrix} R_2 \\ 0 \end{pmatrix},$$

where $\det(U) = 1$. For $m = 1$ one easily finds an appropriate U (one test and some arithmetic). For $m > 1$, p_1 works analogously to the algorithm described in [6, Chapter 1].

Algorithm p_2. To an $n \times n$ matrix A, p_2 computes an $n \times n$ matrix U with $\det(U) = 1$ such that $R = UA$ is upper triangular. For $m > 1$, p_2 works analogously to the QR-algorithm in [6, Chapter 2].

Algorithm p_3. p_3 transforms an $n \times n$ matrix R in upper triangular form to step-form. More precisely, it computes matrices U and Q, where $UR = Q$, $\det(U) = 1$, and Q in step-form.

Algorithm p_3 works recursively. We assume $n = 2m$ (for odd n the algorithm works similarly) and show the algorithm in the following diagram:

$$\begin{pmatrix} R_1 & B \\ 0 & R_2 \end{pmatrix} \xrightarrow{t_1} \begin{pmatrix} Q_1 & C \\ 0 & D \\ 0 & R_2 \end{pmatrix} \xrightarrow{t_2} \begin{pmatrix} Q_1 & C \\ 0 & R_3 \\ 0 & 0 \end{pmatrix} \xrightarrow{t_3} \begin{pmatrix} Q_1 & C \\ 0 & Q_2 \\ 0 & 0 \end{pmatrix}.$$

The R_i are upper triangular $m \times m$ matrices, B is an $m \times m$ matrix, the Q_i are $\operatorname{rank}(Q_i) \times m$ matrices in step-form, C is some $\operatorname{rank}(Q_1) \times m$ matrix and D some $(n - m - \operatorname{rank}(Q_1)) \times m$ matrix. For t_1 we use p_3 recursively to find an U_1 with $U_1 R_1 = Q_1$; then t_1 is a leftmultiplication with the $n \times n$ matrix

$$\begin{pmatrix} U_1 & 0 \\ 0 & 1 \end{pmatrix}.$$

t_2 uses p_1 to transform

$$\begin{pmatrix} D \\ R_2 \end{pmatrix} \mapsto \begin{pmatrix} R_3 \\ 0 \end{pmatrix}$$

and t_2 is an expansion of this transformation. Finally, t_3 is an expansion of the transformation

$$R_3 \mapsto \begin{pmatrix} Q_2 \\ 0 \end{pmatrix},$$

which is computed by p_3 recursively.

Algorithm p_4. p_4 transforms an $n \times m$ matrix A to step-form. Again, p_4 computes the regular transformation matrix U, such that $UA = Q$ has step-form. If $m \leq n$, we concatenate zero-columns to get a square matrix, transform it with p_2 to upper triangular and then with p_3 to step-form. If $m > n$, we do the following:

$$A = (B \quad C) \xrightarrow{t_1} \begin{pmatrix} Q_1 & D \\ 0 & E \end{pmatrix} \xrightarrow{t_2} \begin{pmatrix} Q_1 & D \\ 0 & Q_2 \end{pmatrix}.$$

Here, B is an $n \times n$ matrix, t_1 transforms it to $\binom{Q_1}{0}$ in step-form as described above. Q_1 is a rank$(B) \times n$ matrix in step-form and t_2 is an expansion of the transformation of the $(n - \text{rank}(Q_1)) \times (m-n)$ matrix E to Q_2 in step-form, and this transformation is computed recursively by p_4.

Algorithms p_1, p_2, and p_3 have cost $O(M(n))$ and this is also true for p_4 if $m \leqslant 2n$.

5. The branching algorithm

Theorem 5.1. *There is a computation tree that computes the coefficients of the characteristic polynomial of an $n \times n$ matrix A from k^{n^2} in $O(M(n) \log n)$ time.*

Proof. We modify the simple algorithm, such that it works on arbitrary matrices.

Let us introduce a $k[x]$-module structure of k^n by $xv := Av$ for v from k^n, and let $e_1, e_2, \ldots, e_n$ be a k-base of k^n. Then it suffices to compute $U = U(A)$, which is defined as follows:

(i) $U = (U_1, U_2, \ldots, U_n)$ where U_j is an $n \times m_j$ matrix with $m_j \geqslant 0$ (to simplify notation, we allow $m_j = 0$).

(ii) $U_j = (e_j, Ae_j, A^2e_j, \ldots, A^{m_j-1}e_j)$ if $m_j > 0$.

(iii) The columns of $(U_1, U_2, \ldots, U_i)$ are a k-base of the $k[x]$-module generated by $e_1, e_2, \ldots, e_i$.

Indeed, U is a uniquely defined regular $n \times n$ matrix and $U^{-1}AU$ has the form

$$U^{-1}AU = \begin{pmatrix} A_1 & B_2 & B_2 & & B_n \\ & A_2 & & & \\ & & A_3 & & \\ & 0 & & \ddots & \\ & & & & A_n \end{pmatrix}$$

where A_j is in simple Frobenius form and B_j has all zeroes except for the last column. To see this we interpret the conjugation with U as a base transformation.

Thus, the characteristic polynomial of A is the product of the characteristic polynomials of the A_j and this product can be computed from A_j with $O(n^2)$ operations.

We compute U by successively computing the matrices $1 = V_0, V_1, V_2, V_3, \ldots, V_r = U$ where $r = \lceil \log n \rceil$ and $V_i = (V_{i1}, V_{i2}, \ldots, V_{in})$, where the submatrix V_{ij} has n rows and between 0 and 2^i columns and $V_{ij} = (e_j, Ae_j, A^2e_j, \ldots)$. To define V_{i+1} we show that its computation after V_i has already been computed:

First we compute the matrices W_{ij} defined as follows:

$$W_{ij} = \begin{cases} V_{ij} & \text{if } V_{ij} \text{ has less than } 2^i \text{ columns,} \\ (A^{2^i}V_{ij}, V_{ij}) & \text{otherwise.} \end{cases}$$

For this it is sufficient to compute $A^{2^i}V_i$. Then, $V_{i+1,j}$ is the submatrix of W_{ij} that consists of the first $l_{ij}-1$ columns of W_{ij}, where the l_{ij}th column of W_{ij} is the first one which is linearly dependent on the preceding columns of W_{ij}, and all columns of the matrices $W_{i1}, \ldots, W_{i,j-1}$. (Note that $V_{i+1,j}$ may be empty.) To find the $V_{i+1,j}$, we have to transform $W_i = (W_{i1}, W_{i2}, \ldots, W_{in})$ to step-form.

To show that $V_r = U$ we notice that the k-vector-space generated by the columns of V_{ij} and by the $k[x]$-module generated by $e_1, e_2, \ldots, e_{j-1}$ lies in the $k[x]$-module generated by $e_1, e_2, \ldots, e_j$, and they are equal if V_{ij} has less than 2^i columns.

To compute U we need $r-2$ matrix multiplications for the A^{2^i}, $r-1$ matrix multiplications for the W_{ij}, and $r-1$ transformations to step-form, for which we use p_4 from the last section. In this way, the computation of U and hence also of the coefficients of the characteristic polynomial costs $O(M(n)\log n)$. □

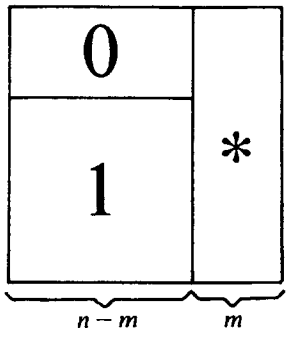
6. The fast algorithm

Theorem 6.1. *There is a straight line algorithm for the coefficients of the characteristic polynomial of an $n \times n$ matrix with indeterminate entries with cost* $O(M(n))$.

We transform the input-matrix A to simple Frobenius form without computing the transforming matrix. This algorithm has been inspired by the algorithm by Danilewski [2].

6.1. The algorithm

We say a matrix is in m-Frobenius form if it has the following form:



1-Frobenius form is a simple Frobenius form and an arbitrary $n \times n$ matrix is always in n-Frobenius form. For simplicity we also call any $n \times n$ matrix in m-Frobenius form for $m \geq n$.

Now let r an integer with $2^r \geq n > 2^{r-1}$. We transform A successively to the similar matrices $A_r = A, A_{r-1}, A_{r-2}, \ldots, A_1, A_0$ where A_i has 2^i-Frobenius form. For the transformation from A_{i+1} to A_i we compute step by step the similar matrices $A_{i,0} = A_{i+1}, A_{i,1}, A_{i,2}, \ldots, A_{i,s_i} = A_i$ where $s_i = \lceil n/2^i \rceil - 1$. We get $A_{i,j+1}$ by conjugating A_{ij} with U_{ij}, the matrix in 2^i-Frobenius form whose last 2^i columns are the last 2^i columns of A_{ij} (the regularity of U_{ij} will be proved later). This is shown in Fig. 1, which also defines various submatrices.

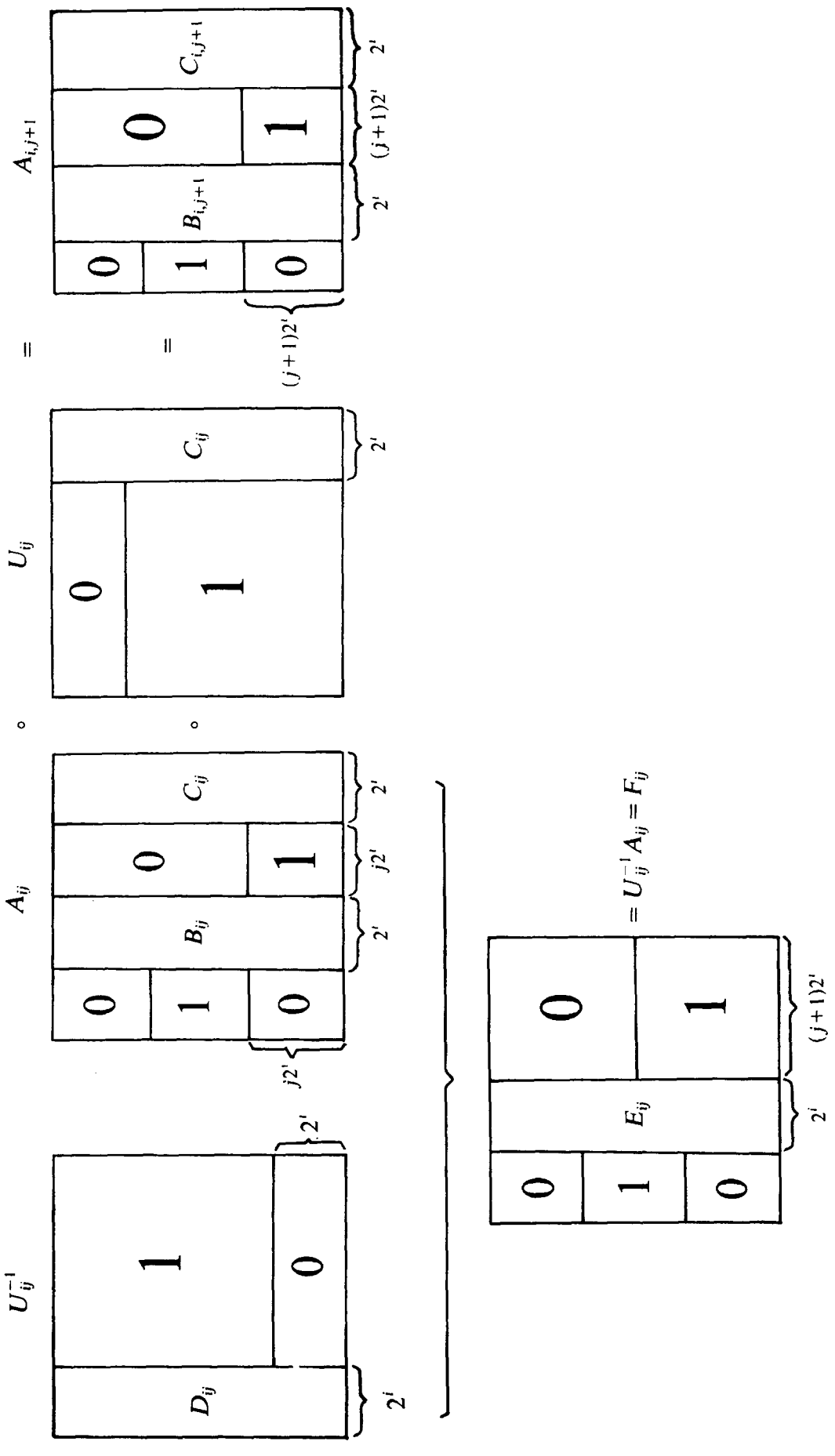

U_{ij}^{-1}
D_{ij}
1
0
2^i
2^i
A_{ij}
0
1
B_{ij}
0
1
C_{ij}
$j2^i$
2^i
$j2^i$
2^i
U_{ij}
0
1
C_{ij}
2^i
$A_{i,j+1}$
0
1
$B_{i,j+1}$
0
1
$C_{i,j+1}$
$(j+1)2^i$
2^i
$(j+1)2^i$
2^i
$= U_{ij}^{-1} A_{ij} = F_{ij}$
0
1
E_{ij}
0
1
2^i
$(j+1)2^i$

Fig. 1.

Fig. 1 holds for $j \leq s_i - 2$. B_{i,s_i} is empty (has zero columns and B_{i,s_i-1} has $n \bmod 2^i$ columns (2^i if $n = 2^r$).

6.2. *Correctness of the algorithm*

It rests us to show that U_{ij} is regular, and we have to find an inversion algorithm that does not fail on any U_{ij}. It is easy to see that after the inversion of the top $2^i \times 2^i$ matrix of C_{ij}, s_i multiplications of $2^i \times 2^i$ matrices suffice to compute U_{ij}^{-1}. This $2^i \times 2^i$ matrix may be inverted by any inversion algorithm as its coefficients are algebraically independent over k; moreover, we prove this for the coefficients of (B_{ij}, C_{ij}). The coefficients of $A = A_{r0}$ are indeterminates over k. By induction it suffices to show that the algebraical independence of the coefficients of (B_{ij}, C_{ij}) implies the existence of $(E_{i,j}, C_{i,j+1})$ and that its coefficients are algebraically independent (indeed $B_{i,j+1} = E_{ij}$ for $j < s_i - 1$ and $B_{is_i} = 0$, and in any case $B_{i,j+1}$ is a submatrix of E_{ij}).

To prove this we first notice that if the coefficients of (B_{ij}, C_{ij}) are algebraically independent, A_{ij} and U_{ij} are regular and hence $A_{i,j+1} = U_{ij}^{-1} A_{ij} U_{ij}$ and $A_{ij} = F_{ij}^{-1} A_{i,j+1} F_{ij}$ hold. This means that the coefficients of A_{ij} are rational functions in the coefficients of $(A_{i,j+1}, F_{i,j})$, and vice versa. So the algebraic independence of the coefficients of (B_{ij}, C_{ij}) implies the algebraic independence of the coefficients of $(C_{i,j+1}, E_{ij})$.

6.3. *Speed*

The computation of U_{ij} is free, to invert it one needs one inversion and s_i multiplications of $2^i \times 2^i$ matrices. Overall, the computation of $U_{ij}^{-1} A_{ij} U_{ij}$ costs $O(M(2^i)n/2^i)$. For every i we have to do this about $n/2^i$ times, while i is varying from $r-1$ to 0. So the cost of the whole transformation is less than

$$c \sum_{i=0}^{r-1} n^2 M(2^i)/2^{2i} \leq cg(n) n^2 \sum_{i=0}^{r-1} 2^{i(\alpha-2)} = O(M(n))$$

for an appropriate constant c. This proves the theorem.

Acknowledgment

The contents of this paper constitutes part of the author's Master's Thesis (Diplomarbeit, Institut für Angewandte Mathematik, Universität of Zürich, 1982). The author is indebted to Prof. V. Strassen who directed this thesis.

References

[1] W. Baur and V. Strassen, The complexity of partial derivatives, *Theoret. Comput. Sci.* **22** (1982) 317–330.

[2] I.S. Beresin and N.P. Shidkon, Das Verfahren von Danilewski, *Numerische Methoden 2* (Deutscher Verlag der Wissenschaften, 1971) Chapter 8.4, 206–214.

[3] J. Bunch and J.E. Hopcroft, Triangular factorization and inversion by fast matrix multiplication, *Mathematics of Computation* **125** (1974).
[4] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems* (American Elsevier, New York, 1975).
[5] D.E. Knuth, *The Art of Computer Programming Vol. II: Seminumerical Algorithms* (Addison-Wesley, Reading, MA, 1969) 560.
[6] A. Schönhage, Unitäre Transformationen grosser Matrizen, *Numerische Math.* **20** (1973) 409–417.
[7] V. Strassen, Gaussian elimination is not optimal, *Numerische Math.* **13** (1969) 354–356.
[8] V. Strassen, Berechnung und Programm, *Acta Informatica* **1** (1971/1972) 320–335; *Acta Informatica* **2** (1973) 46–79.