comput. complex. *Online First*
DOI 10.1007/s00037-010-0295-z

computational complexity

THE COMPLEXITY OF THE INERTIA

Thanh Minh Hoang and Thomas Thierauf

Abstract. The *inertia* of an $n \times n$ matrix A is defined as the triple $(i_+(A), i_-(A), i_0(A))$, where $i_+(A)$, $i_-(A)$, and $i_0(A)$ are the number of eigenvalues of A, counting multiplicities, with positive, negative, and zero real part. We show in this paper that the problem to compute the inertia of an integer matrix is complete for the complexity class **PL** (*Probabilistic Logspace*). Furthermore, we extend this result to the problem of testing the stability of integer matrices.

Keywords. Inertia, matrix stability, logspace complexity.

Subject classification. 68Q15, 68Q25, 15A15, 15A18.

1. Introduction

The inertia of a matrix states how many of the eigenvalues of a matrix have positive, negative, or zero real part, respectively. It plays an important role in linear algebra and in a number of applied fields, in particular in control theory and robotics, where one is interested in the stability of solutions of differential equations and dynamical systems. The stability of fixed points of linear differential equations can be analyzed using the eigenvalues of the corresponding linear transformation. When all the eigenvalues have negative real parts, the system is assymptotically stable. Another area is semidefinite programming where one minimizes a linear function subject to the constraint that an affine combination of symmetric matrices is positive semidefinite, i.e., all the (real) eigenvalues are ≥ 0. The inertia is therefore a fundamental topic in linear algebra and it has been studied by many researchers (see e.g. [12], Volume 2, Chapter XV).

In the present paper we study the *computational complexity* of the inertia of an integer matrix. It was not known whether the inertia is complete for a complexity class. Since many problems from linear algebra can be solved within certain *logspace counting classes* contained in the parallel complexity class (uni-

form) $\mathbf{NC}^2$, it is a natural question whether the inertia can be characterized by one of these logspace counting classes.

Maybe the most important class for problems in linear algebra is the logspace counting class **GapL** (studied by [3]) because it captures the complexity of computing the determinant of an integer matrix [4, 9, 25, 29, 28]. Some other complexity classes are defined on the basis of **GapL**. The class *Exact Counting in Logspace* $\mathbf{C}_=\mathbf{L}$ consists of problems that verify the value of a **GapL**-function. An example is the singularity problem that asks whether the determinant of an integer matrix is zero. In an analogous way, the complexity class **PL** consists of problems that are lower bounded by some **GapL**-function. For example, the problem of testing if the determinant of a matrix is positive, is complete for **PL**.

A method to compute the inertia of a matrix is due to Routh and Hurwitz (see e.g. [12], Volume 2, Chapter XV). Using the Routh–Hurwitz method, one has to compute the sign of the determinant of several matrices. This motivates the conjecture that the inertia can be computed in **PL**. In this paper we summarize the results for the inertia which have been presented in two conference papers [15, 17]. We show in Section 4 that the inertia can be computed in **PL**. In Section 5 we show that the inertia is hard for **PL** under logspace many-one reductions. Hence the inertia is complete for **PL**. In Section 6 we apply our results to the problem of deciding the stability of a matrix and the congruence of two matrices.

We want to mention that Neff and Reif [22, 23] have developed a method to approximate the roots of a polynomial. Hence the eigenvalues of an integer matrix can be determined by approximating the roots of the characteristic polynomial of the matrix. However, we don't see how to use this method to compute the interia within logspace counting classes.

2. Preliminaries

In this section we describe some basic materials and notations that are used throughout the paper.

Complexity classes. By **L** we denote the class of decision problems that can be solved by logarithmically space bounded Turing machines. **FL** is the corresponding function class. **NL** is the nondeterministic counter part of **L**.

For a nondeterministic Turing machine M on input x, we denote the number of accepting and rejecting computation paths by $acc_M(x)$ and $rej_M(x)$, respectively. The difference of these two quantities is denoted by $gap_M(x)$.

That is, $gap_M(x) = acc_M(x) - rej_M(x)$. The complexity class $\#\mathbf{L}$ consists of all functions f such that $f = acc_M$, for some nondeterministic logspace Turing machine M. Similarly, the class **GapL** consists of all functions f such that $f = gap_M$, for some nondeterministic logspace Turing machine M. These classes were first introduced in the polynomial-time setting as $\#\mathbf{P}$ [27] and **GapP** [11].

To analyze the complexity of the inertia later on, we sum up some known closure properties of **GapL**.

THEOREM 2.1 ([3]). *Let $f \in \mathbf{GapL}$. The following functions are in* **GapL** *as well:*

(i) $f(g(\cdot))$, *for any* $g \in \mathbf{FL}$,

(ii) $\sum_{i=0}^{|x|^c} f(x,i)$, *for any constant* c,

(iii) $\prod_{i=0}^{|x|^c} f(x,i)$, *for any constant* c,

(iv) $\binom{f(x)}{g(x)}$, *for any* $g \in \mathbf{FL}$ *such that* $g(x) = O(1)$.

The first property has been improved considerably. [1] showed that for a matrix $A = (a_{i,j})$, where each element $a_{i,j}$ is a **GapL**-function $a_{i,j}(x)$, the function $x \mapsto \det(A)$ is in **GapL**. The same argument can be used to show that the function $x \mapsto (A^m)_{i,j}$ is in **GapL**, for given m, i, j.

THEOREM 2.2 ([1]). *The determinant and the m-th power of a matrix that has* **GapL***-computable elements can be computed in* **GapL**.

The inverse of a matrix A can be computed in **GapL** in the following sense. If A has an inverse, then we have by Cramer's rule that $A^{-1} = \frac{1}{\det(A)}\mathrm{adj}(A)$. The entries of the adjoint matrix are determinants and can therefore be computed in **GapL**. Hence we can compute the numerator and the denominator of the rational elements of A^{-1} in **GapL**. In some cases it suffices to compute the matrix $\det(A)A^{-1}$. Hence all the (integer) entries of this matrix can be computed in **GapL** by the above properties.

Based on **GapL**, complexity classes $\mathbf{C_=L}$ and **PL** and hierarchies based on these classes have been defined in [3] as follows.

- A set L is in $\mathbf{C_=L}$, if there exists a $f \in \mathbf{GapL}$ such that for all x:

$$x \in L \iff f(x) = 0\,.$$

- A set L is in $\mathbf{PL}$, if there exists a $f \in \mathbf{GapL}$ such that for all x:

$$x \in L \iff f(x) > 0\,.$$

- The *Exact Counting Logspace Hierarchy* (over $\mathbf{C_=L}$) collapses to $\mathbf{L}^{\mathbf{C_=L}} = \mathbf{AC}^0(\mathbf{C_=L})$ [2], where $\mathbf{AC}^0(\mathbf{C_=L})$ it is the class of problems $\mathbf{AC}^0$-reducible to some set in $\mathbf{C_=L}$.

- The *Probabilistic Logspace Hierarchy* (over $\mathbf{PL}$) collapses to $\mathbf{PL}$ by the fact $\mathbf{AC}^0(\mathbf{PL}) = \mathbf{NC}^1(\mathbf{PL}) = \mathbf{PL}$ [24, 5], i.e. $\mathbf{PL}$ is closed under $\mathbf{AC}^0$- and $\mathbf{NC}^1$-reductions. In particular, $\mathbf{PL}$ is closed under union, intersection, and complement.

It is known that $\mathbf{NL} \subseteq \mathbf{C_=L} \subseteq \mathbf{AC}^0(\mathbf{C_=L}) \subseteq \mathbf{PL} \subseteq \mathbf{NC}^2$.

Complete problems. Logspace counting classes are interesting because of the complete problems therein. We list some natural problems from linear algebra which are complete for these classes. When nothing else is said, by matrices we mean square integer matrices of order n. All hardness results are via logspace many-one reductions.

The problem of computing one element of the m-th power of a matrix, PowElem, and the problem of computing the determinant of a matrix, Det, are known to be complete for $\mathbf{GapL}$ [25, 9, 29, 28].

Consequently, the problem of testing if the determinant of a matrix is zero, i.e. if a matrix is singular, is complete for $\mathbf{C_=L}$. More general, the set

$$\textsc{Rank}_{\leq} = \left\{ (A, r) \mid \operatorname{rank}(A) \leq r \right\}$$

is complete for $\mathbf{C_=L}$ [2]. Hence the complementary set

$$\textsc{Rank}_{\geq} = \left\{ (A, r) \mid \operatorname{rank}(A) \geq r \right\}$$

is complete for $\mathbf{coC_=L}$.

The verification of the rank can be written as the intersection of a set in $\mathbf{C_=L}$ and in $\mathbf{coC_=L}$ as follows:

$$\begin{aligned} \textsc{v-Rank} &= \left\{ (A, r) \mid \operatorname{rank}(A) = r \right\} \\ &= \textsc{Rank}_{\leq} \cap \textsc{Rank}_{\geq}. \end{aligned}$$

This means that v-Rank $\in \mathbf{C_=L} \wedge \mathbf{coC_=L}$. Moreover, the problem of computing (one bit of) the rank, i.e. the set

$$\textsc{Rank} = \left\{ (A, k, b) \mid \text{ the } k\text{-th bit of } \operatorname{rank}(A) \text{ is } b \right\}.$$

is a complete problem for $\mathbf{AC}^0(\mathbf{C_=L})$ [2].

Problems complete for **PL** are

$$\textsc{PosDet} = \left\{ A \mid \det(A) > 0 \right\},$$
$$\textsc{PosPowElem} = \left\{ (A, m) \mid (A^m)_{1,n} > 0 \right\}.$$

The restriction to integer matrices in the above problems is not essential. Suppose we have a rational matrix A of order n, where the elements are given as pairs (a, b) to represent the rational a/b, for integers a and b. Let d be the product of all denominator in A. Now let $B = dA$. Note that B is an integer matrix and we have $\det(A) = \det(B)/d^n$. Now both, the numerator $\det(B)$ and the denominator d^n can be computed in **GapL**. In this sense, we can say that also the determinant of a rational matrix is in **GapL**.

The inertia. The *inertia* of an $n \times n$ matrix A is defined as the triple $i(A) = (i_+(A), i_-(A), i_0(A))$, where $i_+(A)$, $i_-(A)$, and $i_0(A)$ are the number of eigenvalues of A, counting multiplicities, with positive, negative, and zero real part. We can define (with respect to some fixed coding) the problem of computing the k-th bit of the inertia as follows:

$$\textsc{Inertia} = \left\{ (A, k, b) \mid \text{the } k\text{-th bit of } i(A) \text{ is } b \right\}.$$

The verification of the inertia is the set

$$\text{v-}\textsc{Inertia} = \left\{ (A, i_+, i_-, i_0) \mid i(A) = (i_+, i_-, i_0) \right\}.$$

Note: Inertia as defined above is a decision problem. When we say that *the inertia is in* **PL**, we actually refer to the decision problem Inertia (and not to a function computing the inertia).

3. Tools for polynomials

In the computation of the inertia that we present in Section 4, we need to compute the greatest common divisor (gcd) and the division of univariate polynomials with rational coefficients. Parallel algorithms for polynomial gcd and

polynomial division are known [6] and there are excellent textbooks [30, 18, 19] where these algorithms are explained in detail. Here we want to make the point that polynomial division and gcd are not only in $\mathbf{NC}^2$, but in **GapL**.

Furthermore we present a method to compute the number of distinct real roots of a polynomial in **PL**. Based on this, we show how to compute the number of real roots of a polynomial in **PL**. The technique hereby is to compute a square-free decomposition of a polynomial. We give a short summary of these algorithms. The underlying field are the rationals.

3.1. Polynomial division. There are parallel algorithms for polynomial division with remainder. [10] showed that polynomial division with remainder is in **P**-uniform $\mathbf{TC}_0$. This was improved by [13] to DLOGTIME-uniform $\mathbf{TC}_0$. In our case it suffices to consider polynomials p and q such that q is a divisor of p. We want to compute the polynomial t such that $p = tq$. Let

$$\begin{aligned} p &= a_m x^m + a_{m-1}x^{m-1} + \cdots + a_0\,, \\ q &= b_n x^n + b_{n-1}x^{n-1} + \cdots + b_0\,, \\ t &= c_{m-n}x^{m-n} + c_{m-n-1}x^{m-n-1} + \cdots + c_0\,, \end{aligned}$$

where $a_m, b_n \neq 0$. From the coefficients of these polynomials we define vectors $\boldsymbol{a}$ and $\boldsymbol{c}$, and a $(m+1) \times (m-n+1)$ matrix B as follows.

$$\boldsymbol{a} = \begin{pmatrix} a_m \\ a_{m-1} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix}, \quad B = \begin{pmatrix} b_n & 0 & \cdots & 0 \\ b_{n-1} & b_n & & \vdots \\ \vdots & b_{n-1} & \ddots & 0 \\ b_0 & \vdots & \ddots & b_n \\ 0 & b_0 & & b_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & b_0 \end{pmatrix}, \quad \boldsymbol{c} = \begin{pmatrix} c_{m-n} \\ c_{m-n-1} \\ \vdots \\ c_1 \\ c_0 \end{pmatrix}.$$

Then the condition $p = tq$ is equivalent to the linear equation $B\boldsymbol{c} = \boldsymbol{a}$. Note that matrix B has full column rank because $b_n \neq 0$. Let $\widehat{B}$ be the square matrix of order $m-n+1$ obtained from B by deleting the last n rows, and let $\widehat{\boldsymbol{a}}$ be the vector obtained from $\boldsymbol{a}$ by deleting the lower n entries. Then we have $\widehat{B}\boldsymbol{c} = \widehat{\boldsymbol{a}}$, and furthermore $\widehat{B}$ has full rank. Therefore we get the coefficients of polynomial t by the equation

$$\boldsymbol{c} = \widehat{B}^{-1}\widehat{\boldsymbol{a}} = \frac{1}{b_n^{m-n+1}}\mathrm{adj}(\widehat{B})\widehat{\boldsymbol{a}}\,.$$

It follows that the numerator and the denominator of the rational coefficients of polynomial t can be computed in **GapL**.

3.2. Polynomial GCD computation. For univariate polynomials with leading coefficients different from zero let

$$p(x) = a_m x^m + \cdots + a_0\,,$$
$$q(x) = b_n x^n + \cdots + b_0\,,$$

for $n \leq m$. We want to compute

$$g = \gcd(p, q) = x^d + c_{d-1}x^{d-1} + \cdots + c_0\,.$$

(W.l.o.g. we can choose g to be monic.)

The *Sylvester* or *resultant matrix of p and q* is defined as the matrix S of order $n + m$, where n columns are taken from the coefficients of p, and m columns are taken from the coefficients of q. The following Sylvester matrix is an example for $n = 2$ and $m = 4$:

$$S = \begin{pmatrix} a_4 & 0 & b_2 & 0 & 0 & 0 \\ a_3 & a_4 & b_1 & b_2 & 0 & 0 \\ a_2 & a_3 & b_0 & b_1 & b_2 & 0 \\ a_1 & a_2 & 0 & b_0 & b_1 & b_2 \\ a_0 & a_1 & 0 & 0 & b_0 & b_1 \\ 0 & a_0 & 0 & 0 & 0 & b_0 \end{pmatrix}.$$

For $1 \leq e \leq n$, define the following submatrices of S:

- $S^{(e)}$ is obtained from S by deleting the last e rows, the last e columns of coefficients of p, and the last e columns of coefficients of q,
- S_e is obtained from S by deleting the last $2e$ rows, the last e columns of coefficients of p, and the last e columns of coefficients of q.

Let furthermore $\boldsymbol{t}_e$ be the vector $(0, 0, \ldots, 0, 1)^T$ of length $(m + n - 2e)$. The following matrices give us an example for $n = 2$, $m = 4$, and $e = 1$:

$$S^{(1)} = \begin{pmatrix} a_4 & b_2 & 0 & 0 \\ a_3 & b_1 & b_2 & 0 \\ a_2 & b_0 & b_1 & b_2 \\ a_1 & 0 & b_0 & b_1 \\ a_0 & 0 & 0 & b_0 \end{pmatrix}, \quad S_1 = \begin{pmatrix} a_4 & b_2 & 0 & 0 \\ a_3 & b_1 & b_2 & 0 \\ a_2 & b_0 & b_1 & b_2 \\ a_1 & 0 & b_0 & b_1 \end{pmatrix}, \quad \boldsymbol{t}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then g can be computed by the following two steps:

1. determine the degree of g: this is the value d such that $\det(S_d) \neq 0$ and $\det(S_e) = 0$ for all $e < d$,
2. compute the solution $\boldsymbol{x}_0$ of the equation $S_d\boldsymbol{x} = \boldsymbol{t}_d$ and let $\boldsymbol{c} = S^{(d)}\boldsymbol{x}_0$. Then the vector $\boldsymbol{c}$ has length $m+n-d$ and contains the coefficients of g, namely $\boldsymbol{c} = (0, \ldots, 0, 1, c_{d-1}, \ldots, c_0)^T$.

For the complexity of the two steps we observe that:

- The values $\det(S_e)$ can be computed in **GapL** for all $e \leq n$. The predicate $\det(S_e) = 0$ is in $\mathbf{C}_=\mathbf{L}$. Recall that $\mathbf{C}_=\mathbf{L} \subseteq \mathbf{PL}$.
- In step 2, the equation $S_d\boldsymbol{x} = \boldsymbol{t}_d$ has the unique solution $\boldsymbol{x}_0 = (S_d)^{-1}\ \boldsymbol{t}_d$. By the definition of $\boldsymbol{t}_d$, this is the last column of $(S_d)^{-1}$. Therefore, the numerator and the denominator of the rational coefficients $c_{d-1}, \ldots, c_0$ of g can be computed in **GapL**.

The known closure properties of **GapL** don't suffice to combine all these **GapL**-functions into one **GapL**-function for each coefficient of g. However, we are aiming for a **PL**-algorithm for the inertia and recall that **PL** is closed under $\mathbf{NC}^1$-reductions. Hence we can afford to define several **PL**-predicates and then put some $\mathbf{NC}^1$-circuitry on top of them. For example in step 1, it is easy to construct an $\mathbf{NC}^1$-circuit (actually an $\mathbf{AC}^0$-circuit) that computes the degree of g given the predicates $[\det(S_e) = 0]$ for all $e \leq n$. Similar in step 2, we compute all **GapL**-functions in parallel for all $d \leq n$, i.e. independent of the result of the first step. Now assume that some **PL**-predicate is defined for the **GapL**-functions that compute the coefficients of g. Then it remains to select those that that correspond to the correct degree d of g. Again, this is easy to achieve in $\mathbf{NC}^1$. We summarize the outline in the following lemma.

LEMMA 3.1. *Given two polynomials p and q, the degree of $g = \gcd(p, q)$ and its coefficients can be computed in* **PL**. *Furthermore the coefficients of g are computable in* **GapL** *in the sense that there are functions $f_k^{(d)} \in$ **GapL** such that if $d = \deg(g)$ then $f_k^{(d)}$ computes the k-th coefficient of g. This holds even when the coefficients of p and q are computable in* **GapL**.

3.3. The number of distinct real roots of a polynomial. We will explain next a theorem from linear algebra that shows how to determine the number of *distinct real roots* of a polynomial.

Let $r(x) = x^d + b_1 x^{d-1} + b_2 x^{d-2} + \cdots + b_d$. The *companion matrix of* r is the square matrix C of order d defined as

$$C = \begin{pmatrix} 0 & 0 & \cdots & 0 & -b_d \\ 1 & 0 & \cdots & 0 & -b_{d-1} \\ 0 & 1 & \cdots & 0 & -b_{d-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -b_1 \end{pmatrix}.$$

The *Hankel matrix* $H = (h_{i,j})$ associated with $r(x)$ is defined as

$$h_{i,j} = \mathrm{trace}(C^{i+j-2}), \quad \text{for} \quad i,j = 1,\ldots,d,$$

where $\mathrm{trace}(C^{i+j-2})$ is the sum of all diagonal elements of C^{i+j-2}. Note that H is symmetric. By $\mathrm{sig}(H)$ we denote the *signature* of H which is defined by $\mathrm{sig}(H) = i_+(H) - i_-(H)$. The following theorem can be found in [12], Volume 2, Theorem 6 on page 202:

THEOREM 3.2. *Let H be the Hankel matrix associated with polynomial $r(x)$. Then* $\mathrm{sig}(H)$ *is the number of* distinct real *roots of $r(x)$.*

Note that all the elements of the Hankel matrix H are computable in **GapL** and that H is symmetric. We will show in Theorem 4.6 below that the signature of symmetric matrices can be computed in **PL**, even when the entries are computabe in **GapL**. That is, we have $\mathrm{sig}(H) \in \mathbf{PL}$, and therefore

COROLLARY 3.3. *The number of distinct real roots of a polynomial with* **GapL***-computable coefficients can be determined in* **PL**.

3.4. The number of real roots of a polynomial. In our algorithm to compute the inertia, it will be necessairy to compute the number of real roots of a polynomial $q(x)$. By Corollary 3.3 we get only the number of *distinct* real roots. Since the real roots of $q(x)$ may have some multiplicity > 1 these numbers might not be the same.

To solve this problem, we decompose $q(x)$ into polynomials

$$q(x) = q_1(x) q_2(x) \cdots q_t(x) \tag{3.4}$$

such that each polynomial q_j has only roots of multiplicity 1. Such a polynomial decomposition is called *square-free*. Observe that for each of these polynomials the number of its real roots is the same as the number of *distinct* real roots,

which can be determined by Corollary 3.3. Therefore, the sum of these numbers yields the number of real roots of $q(x)$.

There are known methods to compute a square-free decomposition for $q(x)$ (see e.g. [30]). We give a short outline of one such method that shows that the coefficients of the polynomials $q_i(x)$ can be computed in **GapL**. Let d be the degree of $q(x)$.

Let α be a root of $q(x)$ with multiplicity m, i.e.

$$q(x) = (x-\alpha)^m \; h(x)\,,$$

where $h(\alpha) \neq 0$. Consider the first derivative $q^{(1)}$ of $q(x)$:

$$q^{(1)}(x) = m(x-\alpha)^{m-1} \; h(x) + (x-\alpha)^m \; h^{(1)}(x)\,.$$

Since α is a root of $q^{(1)}(x)$ with multiplicity $m-1$, it is also a root of $\gcd(q, q^{(1)})$ with multiplicity $m-1$. It follows that the polynomial

$$q_1(x) = \frac{q(x)}{\gcd(q, q^{(1)})}$$

is square-free.

More general, let $\alpha_1, \ldots, \alpha_r$ be all the distinct roots of q with multiplicities $m_1, \ldots, m_r$, i.e. $q(x) = \prod_{j=1}^{r} (x-\alpha_j)^{m_j}$, and let $q^{(i)}(x)$ be the i-th derivative of $q(x)$. Then we have

$$\gcd(q, q^{(i)}) = \prod_{\substack{j=1 \\ m_j \geq i}}^{r} (x-\alpha_j)^{m_j - i}\,.$$

It follows that $\gcd(q, q^{(i)})$ is a divisor of $\gcd(q, q^{(i-1)})$ and

$$q_i(x) = \frac{\gcd(q, q^{(i-1)})}{\gcd(q, q^{(i)})}$$

is square-free, for all i. That is, the polynomials $q_i(x)$ yield the desired decomposition (3.4). The sum of the real roots of the polynomials $q_i(x)$ is the number of real roots of $q(x)$, counting multiplicities.

We argue that the algorithm can be implemented in **PL**.

- The coefficients of polynomial q and the derivatives $q^{(i)}$ are computable in **GapL**, for $i = 1, \ldots, d$.

- By Lemma 3.1, the gcd-algorithm on input q and $q^{(i)}$ computes several vectors of **GapL**-functions, one for each potential degree of the gcd, where one of them provides the coefficients of $\gcd(q, q^{(i)})$.
- To compute polynomial $q_i(x)$, we apply the polynomial division algorithm explained above to every pair of vectors from the computation of $\gcd(q, q^{(i-1)})$ and $\gcd(q, q^{(i)})$. This provides us with polynomially many vectors of **GapL**-functions. The vector that corresponds to the correct degrees of the gcd-polynomials provides the coefficients of $q_i(x)$.
- Then we apply Corollary 3.3 to each of the polynomials represented by the above vectors.
- Up to this point we have set up several **PL**-predicates. The final computation is done by a $\mathbf{NC}^1$-circuit that uses these predicates as gates. First we have to determine the actual degrees of $\gcd(q, q^{(i)})$, for all i (Lemma 3.1). Then we can select the coefficients of polynomials $q_i(x)$, for all i and sum up the number of real roots of $q_1(x), \ldots, q_d(x)$. This yields the number of real roots of $q(x)$.

We conclude

LEMMA 3.5. *Let $q(x)$ be a polynomial with coefficients computable in* **GapL**. *The number of real roots of $q(x)$ can be computed in* **PL**.

4. Computing the inertia in PL

In this section we prove the following theorem.

THEOREM 4.1. INERTIA *and* V-INERTIA *are in* **PL**.

Let A be an $n \times n$ matrix. For the computation of $i(A)$, the inertia of A, it suffices to compute $i_+(A)$ because we have $i_-(A) = i_+(-A)$ and $i_0(A) = n - i_+(A) - i_+(-A)$.

The eigenvalues of A are the roots of the characteristic polynomial $\chi_A(x) = \det(xI - A)$. Hence the inertia has an equivalent formulation with respect to the location of the roots of polynomials. In order to compute $i_+(A)$, we show how to determine $i_+(p)$, the number of roots with positive real part of an integer polynomial $p(x)$, counting multiplicities, and apply it with $p(x) = \chi_A(x)$.

A known method to determine the number of roots in the right half-plane of a given real polynomial is provided by Routh and Hurwitz (see e.g. [12], Volume 2, Chapter XV), which we explain in the next section. However, as we

will see, the Routh–Hurwitz theorem does not completely solve the problem. There remains one case that requires considerably more effort to solve. We show how to handle the remaining case and how to compute $i_+(p)$ in **PL**.

4.1. The Routh–Hurwitz theorem. Let $p(x) = x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_n$ be a polynomial with integer coefficients. Define $a_0 = 1$. The *Routh–Hurwitz matrix* of p is defined by the $n \times n$ matrix $\Omega(p)$,

$$\Omega(p) = \begin{pmatrix} a_1 & a_3 & a_5 & a_7 & \cdots & 0 \\ a_0 & a_2 & a_4 & a_6 & \cdots & 0 \\ 0 & a_1 & a_3 & a_5 & \cdots & 0 \\ 0 & a_0 & a_2 & a_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_n \end{pmatrix}.$$

That is, the diagonal elements of $\Omega(p) = (\omega_{i,j})$ are $\omega_{i,i} = a_i$. In the i-th column, the elements above the diagonal are $\omega_{i-1,i} = a_{i+1}$, $\omega_{i-2,i} = a_{i+2}$, ... until we reach either the first row $\omega_{1,i}$ or a_n. In the latter case, the remaining elements are filled with zeros. The elements below $\omega_{i,i}$ are $\omega_{i+1,i} = a_{i-1}$, $\omega_{i+2,i} = a_{i-2}$, ..., a_1, a_0, 0, 0, ... down to the last row $\omega_{n,i}$.

The successive leading principal minors D_i of $\Omega(p)$ are called the *Routh–Hurwitz determinants*. They are $D_1 = \det(a_1)$, $D_2 = \det\left(\left(\begin{smallmatrix} a_1 & a_3 \\ a_0 & a_2 \end{smallmatrix}\right)\right)$, ..., $D_n = \det(\Omega(p))$. The following theorem is taken from [12], Volume 2, Chapter XV, Theorem 5 on page 201:

THEOREM 4.2 (Routh–Hurwitz). *If $D_n \neq 0$, then the number of roots of the polynomial $p(x)$ in the right half-plane is determined by the formula*

$$i_+(A) = V\left(1, D_1, \frac{D_2}{D_1}, \ldots, \frac{D_n}{D_{n-1}}\right),$$

where $V(x_1, x_2, \ldots)$ computes the number of sign alternations in the sequence of numbers $x_1, x_2, \ldots$. For the calculation of the values of V, for every group of l successive zero Routh–Hurwitz determinants (l is always odd!)

$$D_s \neq 0\,, \quad D_{s+1} = \cdots = D_{s+l} = 0\,, \quad D_{s+l+1} \neq 0$$

we have to set $V(\frac{D_s}{D_{s-1}}, \frac{D_{s+1}}{D_s}, \ldots, \frac{D_{s+l+2}}{D_{s+l+1}}) = k + \frac{1-(-1)^k\varepsilon}{2}$, where $l = 2k-1$ and $\varepsilon = \operatorname{sign}(\frac{D_s}{D_{s-1}}\frac{D_{s+l+2}}{D_{s+l+1}})$. For $s = 1$, $\frac{D_s}{D_{s-1}}$ has to be replaced by D_1; and for $s = 0$, by a_0.

Note that the assumption $D_n \neq 0$ is used in the theorem. That is, we can apply the theorem directly only in the case that the Routh–Hurwitz matrix $\Omega(p)$ is regular. We analyze this *regular case* in Section 4.2 and then turn to the *singular case* where $D_n = 0$ in Section 4.3.

4.2. The regular case. Assume that $D_n \neq 0$. We apply the Routh–Hurwitz Theorem 4.2 to determine $i_+(A)$. It is clear that all elements of the Routh–Hurwitz matrix are computable in **GapL**. Therefore, by Theorem 2.2, all the Routh–Hurwitz determinants D_i are computable in **GapL**, too. It follows that one can decide in **PL** whether D_i is positive, negative, or zero. The k-th bit of i_+ can be therefore computed by a family of $\mathbf{AC}^0$-circuits with **PL** oracles. Since $\mathbf{AC}^0(\mathbf{PL}) = \mathbf{PL}$, the sets INERTIA and V-INERTIA are in **PL**. By Theorem 2.2 this holds even if the elements of matrix A are computed by **GapL**-functions.

THEOREM 4.3. *For matrices, even with* **GapL***-computable elements, that have a regular Routh–Hurwitz matrix,* INERTIA *and* V-INERTIA *are in* **PL**.

4.3. The singular case. We consider the case when $D_n = 0$. Our first step is to reduce the problem to compute $i_+(p)$ to compute $i_0(g)$ for some polynomial g derived from p.

It is known from linear algebra (see [12], Volume 2, Chapter XV, Section 7) that $D_n = 0$ iff $p(x)$ has a *pair of opposite roots*, i.e. there is a pair of roots x_0 and $-x_0$ such that $p(x_0) = p(-x_0) = 0$. Let us split $p(x)$ into even and odd terms: $p(x) = p_1(x) + p_2(x)$ where

$$p_1(x) = x^n + a_{n-2}x^{n-2} + a_{n-4}x^{n-4} + \cdots ,$$
$$p_2(x) = a_{n-1}x^{n-1} + a_{n-3}x^{n-3} + \cdots .$$

Note that precisely the pairs of opposite roots of $p(x)$ are the common roots of $p_1(x)$ and $p_2(x)$. Therefore the roots of $g(x) = \gcd(p_1(x), p_2(x))$ are precisely all the pairs of opposite roots of $p(x)$. Now consider the decomposition

$$p(x) = g(x)p_0(x) . \tag{4.4}$$

It follows that $p_0(x)$ has no pair of opposite roots. Therefore we can determine $i_+(p_0)$ in **PL** by Theorem 4.3.

Since $i_+(p) = i_+(g) + i_+(p_0)$, it now suffices to compute $i_+(g)$. However, the Routh–Hurwitz method doesn't apply to $g(x)$ because $g(x)$ has purely pairs of opposite roots. Nevertheless we can conclude that $g(x)$ has an even number of roots and is therefore of even degree, and we have

$$i_+(g) = \frac{1}{2}\big(\deg(g) - i_0(g)\big) . \tag{4.5}$$

The degree $\deg(g)$ can be determined in **PL** by Lemma 3.1. In order to compute $i_+(g)$ in **PL** it therefore suffices to compute $i_0(g)$ in **PL**.

We distinguish the case whether the input matrix A is symmetric or not.

Case 1: A is symmetric. In this case all eigenvalues of A are real. Therefore the roots of $p(x)$, and hence of $g(x)$ are real too. Hence $i_0(g)$ is exactly the multiplicity of $x = 0$ as a root of $g(x)$. The latter value is the smallest exponent of x in $g(x)$ with a nonzero coefficient. Since the coefficients of $g(x)$ are computable in **PL** by Lemma 3.1, $i_0(g)$ can be determined in **PL**. We conclude

THEOREM 4.6. *For symmetric matrices, even with* **GapL***-computable entries,* INERTIA *and* V-INERTIA *are in* **PL**.

Recall that we used this theorem in the proof of Corollary 3.3 which in turn was used in the proof of Lemma 3.5. Hence, by now we can apply the lemma.

Case 2: A is non-symmetric. We reduce the problem to compute $i_0(g)$ to the problem to compute $i_0(p)$.

By decomposition (4.4) we have

$$i_0(g) = i_0(p) - i_0(p_0)\,. \tag{4.7}$$

Because p_0 has no opposite roots $i_0(p_0)$ can be easily determined by

$$i_0(p_0) = \begin{cases} 1 & \text{if } p_0(0) = 0\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

In summary, in order to compute $i_+(p)$ it suffices to compute $i_0(p)$, the number of purely imaginary roots of $p(x)$: from $i_0(p)$ we get $i_0(g)$ by Equation (4.7), from which we get $i_+(g)$ by Equation (4.5). By adding $i_+(g)$ and $i_+(p_0)$ we get $i_+(p)$. Therefore it remains to show how to compute the number of purely imaginary roots of a polynomial.

By Lemma 3.5 we can compute the number of *real roots* of a polynomial with **GapL**-computable coefficients in **PL**. In order to apply this lemma to determine $i_0(p)$, we transform $p(x)$ into a polynomial $q(x)$ by rotating the complex plane by 90°. The effect is that the purely imaginary roots of $p(x)$ become the purely real roots of $q(x)$.

A 90° rotation. For matrices A_1 and A_2 of order n and m, respectively, we denote the *Kronecker* or *tensor product* by $A_1 \otimes A_2$. Let $\lambda_j(A_1)$ and $\lambda_k(A_2)$ be the eigenvalues of A_1 and A_2. Then the eigenvalues of $A_1 \otimes A_2$ are $\lambda_j(A_1)\lambda_k(A_2)$, for all j, k. For our purpose, observe that the skew-symmetric matrix $E = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ has the eigenvalues $\lambda_1(E) = +i$ and $\lambda_2(E) = -i$. Define

$$B = E \otimes A = \begin{pmatrix} 0 & -A \\ A & 0 \end{pmatrix},$$

where A is the given matrix with the characteristic polynomial $p(x)$. Then the eigenvalues of B are $i\lambda_k(A)$ and $-i\lambda_k(A)$, where $\lambda_k(A)$ runs through all eigenvalues of A. It follows that the number of real eigenvalues of B is exactly equal to $2i_0(p)$. Let

$$q(x) = \chi_B(x)\,,$$

the characteristic polynomial of B. Then we have

$$2i_0(p) = \text{the number of real roots of } q\,.$$

We conclude that in order to compute $i_0(p)$, it suffices to compute the number of real roots of $q(x)$. The latter problem is in **PL** by Lemma 3.5. From the number of real roots of $q(x)$, a $\mathbf{NC}^1$ circuit can compute $i_+(p)$. Since **PL** is closed under $\mathbf{NC}^1$-reductions, this proves Theorem 4.1.

5. The inertia is hard for PL

In this section we show a logspace reduction from PosPowElem to Inertia and v-Inertia. Recall that PosPowElem is complete for **PL**. Together with Theorem 4.1 we conclude that the inertia problem is complete for **PL** under logspace reductions. In the initial step of the reduction we take an instance of PosPowElem and construct a matrix with a certain characteristic polynomial. This is a slight modification of a reduction presented in [16]. We give a short outline in the following lemma.

Lemma 5.1. *Given an instance A and m for* PosPowElem *and an integer b. Let $a = (A^m)_{1,n}$. We can construct a matrix B in logspace, such that*

$$\chi_B(x) = x^{N-2m-1}\big(x^{2m+1} - (a+b)\big)\,.$$

Proof. Interpret the matrix A of order n as the representation of a directed, weighted, bipartite graph G_0 on $2n$ nodes and e edges, where e is the number

of nonzero entries of A. That is, the nodes of G_0 are arranged in two columns of n nodes each. In both columns, nodes are numbered from 1 to n. If element $a_{k,l}$ of A is not zero, then there is an edge with weight $a_{k,l}$ from node k in the first column to node l in the second column. Next replace each edge (u, v) of G_0 by two edges (u, w) and (w, v), for a new node w with weights 1 and $a_{k,l}$, respectively. Call the new graph G_1.

Now, take m copies of graph G_1, put them in a sequence and identify each second column of nodes with the first column of the next graph in the sequence. Call the resulting graph G'. Observe that $(A^m)_{1,n}$ is the sum of the weights of all paths in G' from node 1 in the first column to node n in the last column. Call these two nodes s and t, respectively. Then we take $2m-1$ new nodes $u_1, \ldots, u_{2m-1}$ and add the directed path $s \to u_1 \to u_2 \to \cdots \to u_{2m-1} \to t$. The first edge, (s, u_1), gets weight b, the other edges get weight 1. Finally add an edge from t to s with weight 1. Call the resulting graph G. Note that G has $N = m(n+e)+n+2m-1$ nodes and all cycles in G have precisely length $2m+1$.

Let B be the adjacency matrix of G and let $\chi_B(x) = x^N + \sum_{i=0}^{N-1} c_i x^i$ be the characteristic polynomial of B. From combinatorial matrix theory we know that the coefficients of the characteristic polynomial c_i in $\chi_B(x)$ equals the sum of the disjoint weighted cycles that cover $N-i$ nodes in G, with appropriate sign (see [7, 8, 31, 20, 21]). Hence we get

$$c_{N-(2m-1)} = -\big((A^m)_{1,n} + b\big)$$

and all other coefficients must be zero. This proves the lemma. □

THEOREM 5.2. INERTIA *and* V-INERTIA *are hard for* **PL** *under logspace reductions.*

PROOF. Let (A, m) be an input for POSPOWELEM. One has to decide whether $a = (A^m)_{1,n} > 0$. Apply Lemma 5.1 with $b = 0$. Then matrix B has characteristic polynomial $\chi_B(x) = x^{N-2m-1}(x^{2m+1} - a)$. The eigenvalues of B are the roots of $\chi_B(x) = 0$. We first consider the case $a \neq 0$. The roots of $x^{2m+1} - a = 0$ are $a^{\frac{1}{2m+1}} e^{i\frac{2\pi k}{2m+1}}$ for $k = 0, 1, \ldots, 2m$. Geometrically, these roots are the corners of a regular $(2m+1)$-gon inscribed in a circle of radius $a^{\frac{1}{2m+1}}$ with its center at the origin. Since $2m+1$ is odd, none of these roots lies on the imaginary axis. This implies that $i_0(B) = N-(2m+1)$, and one of $i_+(B)$ and $i_-(B)$ is m and the other is $m+1$. Moreover, these values depend on the

sign of a. Namely, if $a > 0$, we have

$$i_+(B) = \begin{cases} m+1\,, & \text{if } 2m+1 \equiv 1 \pmod 4\,, \\ m\,, & \text{if } 2m+1 \equiv 3 \pmod 4\,. \end{cases} \tag{5.3}$$

Note in particular that $i_+(B)$ is always odd in this case.

If $a < 0$, then the values for $i_+(B)$ just switch and $i_+(B)$ is always even. In the case where $a = 0$, all the eigenvalues of B are zero, i.e. we have $i(B) = (0, 0, N)$. In particular, $i_+(B)$ is even.

In summary, given A and m, by equation (5.3) we can compute three numbers i_+, i_-, and i_0 in logspace such that

$$(A^m)_{1,n} > 0 \iff i(B) = (i_+, i_-, i_0) \tag{5.4}$$

$$\iff i_+(B) = \text{odd}\,. \tag{5.5}$$

Equivalence (5.4) provides a reduction from PosPowElem to v-Inertia, and by equivalence (5.5) we get a a reduction from PosPowElem to Inertia. □

By Theorems 4.1 and 5.2 we obtain the main result of the paper.

Corollary 5.6. Inertia *and* v-Inertia *are complete for* **PL**.

6. Problems related to the inertia

There are several properties of matrices that depend on the number of positive or negative eigenvalues of the matrix. In this section, we study the stability and the congruence of matrices. All hardness results are via logspace reductions.

6.1. The stability of matrices. Recall that a matrix A said to be *positive stable*, if all its eigenvalues have positive real parts, i.e. if $i_+(A) = n$, where n is the order of A.

Therefore, the problem of testing if A is positive stable, PosStable, is equivalent to the problem of deciding whether the inertia of A is $(n, 0, 0)$ and is therefore in **PL** too. We show in the following theorem that stability is also hard for **PL**.

Theorem 6.1. PosStable *is complete for* **PL**.

Proof. It suffices to reduce PosPowElem to PosStable. Let matrix A and m be given. W.l.o.g. we can assume that all the elements of A are from $\{-1, 0, 1\}$ [26]. As a bound for $a = (A^m)_{1,n}$ we have $|a| \leq n^m$.

We apply Lemma 5.1 with $b = -n^{2m+1}$. Hence B has the characteristic polynomial

$$\chi_B(x) = x^{N-2m-1}\big(x^{2m+1} - (a+b)\big)\,.$$

By our choice of b we have $a + b < 0$.

The $2m+1$ nonzero eigenvalues of B lie on the circle of radius $r = |a+b|^{\frac{1}{2m+1}}$ around the origin. Because $a + b < 0$ and $2m + 1$ is odd the eigenvalue of B with the smallest real part is $\lambda_1(B) = -r$. Define $C = nI + B$, where I is the identity matrix of order N. The eigenvalues of C are

$$\lambda_i(C) = n + \lambda_i(B)\,, \quad \text{for} \quad 1 \le i \le N\,.$$

Hence, the eigenvalue of C with the smallest real part is $\lambda_1(C) = n - r$. Therefore, we have

$$\begin{aligned} a > 0 &\iff a + b > b \\ &\iff -r^{2m+1} > -n^{2m+1} \\ &\iff -r > -n \\ &\iff \lambda_1(C) > 0\,. \end{aligned}$$

In summary, we have $(A^m)_{1,n} > 0 \iff C \in$ PosStable. □

Note that the **PL**-hardness of v-Inertia (Theorem 5.2) also follows directly from Theorem 6.1.

A matrix is called *positive definite*, if it is symmetric and positive stable. Let PosDefinite be the set of all positive definite matrices. Clearly PosDefinite $\in$ **PL** because the upper bound for PosStable applies to PosDefinite as well. Unfortunately, the matrix C constructed in the proof of Theorem 6.1 is not symmetric. Therefore we don't get a lower bound for PosDefinite from this proof. A trivial observation is that for any matrix A we have:

$$\det(A) \neq 0 \iff AA^T \text{ is positive definite}\,.$$

Because AA^T is symmetric, we have a logspace reduction from $\mathbf{coC_{=}L}$ to PosDefinite.

Proposition 6.2. PosDefinite *is in* **PL** *and hard for* $\mathbf{coC_{=}L}$.

6.2. The congruence of symmetric matrices. Recall that matrices A and B of order n are called *congruent* (via a real matrix) if there exists a nonsingular real matrix S such that $A = SBS^T$. Sylvester's law of inertia

says that two symmetric matrices are congruent if and only if they have the same inertia. Therefore, the problem of testing if two symmetric matrices are congruent, MatCong, is in **PL** by Theorem 4.1. Since the matrix B constructed in the proof of Theorem 5.2 is not symmetric it doesn't follow from this proof that MatCong is hard for **PL**. The following theorem shows a lower bound for MatCong.

Theorem 6.3. MatCong *is in* **PL** *and is hard for* $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$ *under logspace reductions.*

Proof. The problem of testing if two matrices have the same rank is known to be complete for $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$ [2, 14]. We reduce this problem to MatCong.

Let A and B be two $n \times n$ matrices. We will construct two matrices C and D, such that $\operatorname{rank}(A) = \operatorname{rank}(B)$ if and only if C and D are congruent.

Define $C = A^T A$ and $D = B^T B$. Note that C and D are symmetric and we have

$$\operatorname{rank}(A) = \operatorname{rank}(A^T A) = \operatorname{rank}(C)\,.$$

Similarly, we have $\operatorname{rank}(B) = \operatorname{rank}(D)$.

Moreover, the eigenvalues of C and D are all real and non-negative. Therefore, we have $\operatorname{rank}(C) = \operatorname{sig}(C)$ and $\operatorname{rank}(D) = \operatorname{sig}(D)$. It follows that A and B have the same rank if and only if C and D have the same rank *and* signature, i.e. if and only if C and D are congruent. This proves the claim. □

Open problems

Close the gap between upper and lower bound for PosDefinite and MatCong.

Acknowledgements

We thank Eric Allender and the referees of the paper for very helpful comments. Supported by DFG grants Scho 302/7-2.

References

[1] E. Allender, V. Arvind & M. Mahajan. Arithmetic complexity, Kleene closure, and formal power series. *Theory of Computing Systems*, 36(4):303–328, 2003.

[2] E. Allender, R. Beals & M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999.

[3] E. Allender & M. Ogihara. Relationship among PL, #L, and the determinant. *RAIRO-Theoretical Informatics and Applications*, 30:1–21, 1996.

[4] S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[5] R. Beigel & B. Fu. Circuits over PP and PL. *Journal of Computer and System Sciences*, 60:422–441, 2000.

[6] A. Borodin, J. von zur Gathen & J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52:241–256, 1982.

[7] R. Brualdi & H. Ryser. *Combinatorial Matrix Theory*, volume 39 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1991.

[8] D. Cvetković, M. Doob & H. Sachs. *Spectra of Graphs, Theory and Application*. Academic Press, 1980.

[9] C. Damm. DET = $L^{(\#L)}$. Technical Report Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[10] W. Eberly. Very fast parallel polynomial arithmetic. *SIAM Journal on Computing*, 18(5):955–976, 1989.

[11] S. Fenner, L. Fortnow & S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[12] F. Gantmacher. *The Theory of Matrices*, volume 1 and 2. AMS Chelsea Publishing, 1977.

[13] W. Hesse, E. Allender & D. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[14] T. M. Hoang & T. Thierauf. The complexity of verifying the characteristic polynomial and testing similarity. In *15th IEEE Conference on Computational Complexity (CCC)*, pages 87–95. IEEE Computer Society Press, 2000.

[15] T. M. Hoang & T. Thierauf. The complexity of the inertia. In *22nd Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Lecture Notes in Computer Science 2556, pages 206–217. Springer-Verlag, 2002.

[16] T. M. Hoang & T. Thierauf. The complexity of the characteristic and the minimal polynomial. *Theoretical Computer Science*, 295:205–222, 2003.

[17] T. M. Hoang & T. Thierauf. The complexity of the inertia and some closure properties of GapL. In *20th IEEE Conference on Computational Complexity (CCC)*, pages 28–37. IEEE Computer Society Press, 2005.

[18] D. Ierardi & D. C. Kozen. Parallel resultant computation. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 679–720. Morgan Kaufmann, 1993.

[19] D. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1991.

[20] M. Mahajan & V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 1997(5), 1997.

[21] M. Mahajan & V. Vinay. Determinant: Old algorithms, new insights. *SIAM Journal on Discrete Mathematics*, 12(4):474–490, 1999.

[22] C. A. Neff. Specified precision root isolation is in NC. *Journal of Computer and System Science*, 48:429–463, 1994.

[23] C. A. Neff & J. H. Reif. An efficient algorithm for the complex roots problem. *Journal of Complexity*, 12:81–115, 1996.

[24] M. Ogihara. The PL hierarchy collapses. *SIAM Journal on Computing*, 27:1430–1437, 1998.

[25] S. Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. of Computer Science and Information Mathematics, University of Electro-Communications, Chofu-shi, Tokyo 182, Japan, 1991.

[26] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.

[27] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[28] L. Valiant. Why is boolean complexity theory difficult. In M. S. Paterson, editor, *Boolean Function Complexity*, London Mathematical Society Lecture Notes Series 169. Cambridge University Press, 1992.

[29] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *6th IEEE Conference on Structure in Complexity Theory*, pages 270–284, 1991.

[30] J. von zur Gathen & J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.

[31] D. Zeilberger. A combinatorial approach to matrix algebra. *Discrete Mathematics*, 56:61–72, 1985.

Manuscript received 16 November 2007

Thanh Minh Hoang
Institut Theoretische Informatik
Universität Ulm
89069 Ulm, Germany
thanh.hoang@uni-ulm.de
www.uni-ulm.de/in/theo/mitarbeiter/hoang

Thomas Thierauf
Fakultät Elektronik und Informatik
HTW Aalen
73430 Aalen, Germany
thomas.thierauf@uni-ulm.de
image.informatik.htw-aalen.de/Thierauf/