J. Symbolic Computation (2001) **11**, 1–000

Symbolic and numeric methods for exploiting structure in constructing resultant matrices

IOANNIS Z. EMIRIS AND VICTOR Y. PAN

INRIA, B.P. 93, Sophia-Antipolis 06902, France.

Ioannis.Emiris@inria.fr, http://www-sop.inria.fr/galaad/emiris

Mathematics and Computer Science Department, CUNY, Bronx, NY 10566, USA.

vpan@lehman.cuny.edu

(Received 20 June 2001)

Resultants characterize the existence of roots of systems of multivariate nonlinear polynomial equations, while their matrices reduce the computation of all common zeros to a problem in linear algebra. Sparse elimination theory has introduced the sparse resultant, which takes into account the sparse structure of the polynomials. The construction of sparse resultant, or Newton, matrices is the critical step in the computation of the multivariate resultant and the solution of a nonlinear system. We reveal and exploit the quasi-Toeplitz structure of the Newton matrix, thus decreasing the time complexity of constructing such matrices by roughly one order of magnitude to achieve quasi-quadratic complexity in the matrix dimension. The space complexity is also decreased analogously. These results imply similar improvements in the complexity of computing the resultant polynomial itself and of solving zero-dimensional systems. Our approach relies on fast vector-by-matrix multiplication and uses the following two methods as building blocks. First, a fast and numerically stable method for determining the rank of rectangular matrices, which works exclusively over floating point arithmetic. Second, exact polynomial arithmetic algorithms that improve upon the complexity of polynomial multiplication under our model of sparseness, offering bounds linear in the number of variables and the number of nonzero terms.

1. Introduction

Resultants characterize the solvability of zero-dimensional systems of multivariate nonlinear polynomial equations, and their matrix formulae reduce the computation of all common solutions to a matrix eigenproblem. Multivariate resultants have a long and rich history in the context of classical elimination. More recently, sparse elimination theory introduced the *sparse resultant*, which generalizes the classical resultant and whose degree depends on the monomial structure of the polynomials, thus leading to tighter bounds and faster algorithms for systems encountered in application areas. Sparse resultant matrices, also known as *Newton matrices,* generalize Sylvester and Macaulay matrices, and from their determinants the sparse resultant can be computed. This paper identifies and exploits the structure of Newton matrices by designing an efficient numerical rank test and exact polynomial arithmetic in the context of sparse elimination. This

0747–7171/90/000000 + 00 $03.00/0

yields better time and space complexity bounds for their construction, the computation of the sparse resultant and, ultimately, the solution of nonlinear polynomial systems. Our methods can be extended to the case of imperfectly known coefficients, or to solving overconstrained systems as long as the number of solutions is finite. Hence, our effort is a contribution in efficient and numerically stable algorithms in nonlinear algebra.

Construction and manipulation of Newton matrices is a critical operation in some of the most efficient known algorithms for solving zero-dimensional systems of equations [CKL89, Emi96, Laz81, Man94, MP97, MRP00]. They may be used to construct the multiplication table in the quotient ring defined by a polynomial ideal, which is a crucial step in numerically approximating all solutions of a well-constrained system. Our practical motivation is the real-time solution of systems with, say, up to 10 variables; or the computation of the resultant polynomial, for instance, in graphics and modeling applications where the implicit expression of a curve or surface is precisely the resultant. Such systems may give rise to matrices with dimension in the hundreds or even higher, as illustrated by specific examples in table 2. By palliating the effects of matrix size, with respect to both time and space complexity, our work deals with what is probably the Achilles heel of Newton's matrices. This general area is a field of active research.

The solution of such equations is itself irrational, even in the univariate case. Hence it requires further numeric computation following the construction of a resultant matrix, as explained in section 6.

The main contribution of the present paper is to construct Newton matrices with quasi-quadratic time complexity, whereas the existing methods have cubic complexity in the matrix dimension. This uses the incremental construction algorithm of [EC95]. We improve both time and space complexities by almost one order of magnitude and manage to rely essentially on floating-point routines, in order to fully use the power of contemporary computers as well as the availability of linear algebra software libraries. Analogous improvements are then obtained for computing the sparse resultant polynomial itself and, eventually, for solving systems of polynomial equations by resultant-based methods.

These bounds ultimately rely on the Fast Fourier Transform (FFT). Yet, for smaller input sizes, other polynomial multiplication methods, such as Karatsuba's, may offer simpler though asymptotically slower alternatives. Table 1 compares the existing and the achieved complexities, in terms of row and column dimension, respectively denoted a and c, and the number of variables n, as explained in section 6. Note that $a > c$ and typically $c \gg n$.

Table 1. Asymptotic complexity for matrix construction

method	time	space
previous	a^2c	ac
Karatsuba	$c^{2.6}n$	cn
FFT	c^2n	cn

There are some further results of independent interest. First, a fast numeric procedure, namely algorithm 3.1, is proposed for computing the numerical rank of a rectangular matrix, based on Lanczos' algorithm. It exploits matrix structure, in particular fast vector-by-matrix multiplication, and achieves numerical *stability* by the standard technique of vector orthogonalization. A faster variant, namely algorithm 3.3, is designed for testing whether a given rectangular matrix is rank deficient or not, within the prescribed tolerance. Second, the reduction of vector-by-matrix multiplication to polynomial multiplication calls for exact sparse polynomial arithmetic, namely evaluation

and interpolation. We design such algorithms, with *linear* time and space complexity in terms of n and the cardinality of the supports, thus improving the known bounds under our model of sparseness. In short, our approach relies on the interplay of numeric and symbolic building blocks. Lastly, by studying the structure of resultant matrices, we generalize the theory of Toeplitz matrices and some of their essential properties to quasi-Toeplitz matrices, which include Newton, Macaulay and Sylvester matrices. Some results can be extended to Bézout and Dixon resultant matrices.

Dealing with randomized algorithms, we shall distinguish between *Las Vegas* algorithms, which may fail with a small bounded probability but otherwise output correct solutions and, on the other hand, *Monte Carlo* algorithms, which may produce incorrect results but with a small bounded probability.

All time complexity bounds are in terms of *arithmetic complexity;* hereafter "ops" stands for "arithmetic operations." Space complexity includes the input and output storage, unless we explicitly refer to "additional" storage space. We let $O^*(c)$ stand for $O(c \log^v c)$ for any fixed constant v independent of c and we let $|S|$ denote the cardinality of a set S. In our notation, W^T is the transpose of a matrix or of a vector W, W^H is the Hermitian transpose of a matrix and I_k is a $k \times k$ identity matrix. This paper makes heavy use of dense structured matrices; for a comprehensive account of their definitions and properties, the reader may consult [BP94, Pan01]. In particular, recall that a $k \times k$ Toeplitz matrix can be multiplied by a vector in $O(k \log k)$ ops [BP94, sect. 2.5] and in $O(k)$ storage space, based on the FFT.

This paper is organized as follows. The next section expands on related work. Section 3 proposes an efficient numerical rank determination algorithm and a faster rank deficiency test. Section 4 considers exact sparse polynomial arithmetic. Certain important properties of the Newton matrix are investigated in section 5, including its multiplication by a vector. Section 6 improves the complexity of a known algorithm for constructing such matrices by exploiting their structure. Computing the sparse resultant itself is investigated in section 7. We conclude with extensions of our results, a discussion of certain alternatives, and some open questions, in section 8.

2. Related work

Resultant-based approaches to studying and solving systems of polynomial equations have a long history. Recent interest in matrix-based methods is supported by certain practical results that have established resultants, along with Gröbner bases and continuation techniques, as the method of choice in solving zero-dimensional polynomial systems [CKL89, Emi96, Laz81, Man94, MP97, vdW50]. A generalization of the classical resultant was introduced in the context of sparse elimination theory (outlined in section 5). Two main algorithms, generalizing Sylvester's as well as Macaulay's constructions, have been proposed for constructing Newton, or sparse resultant, matrices: The subdivision-based algorithm by Canny and Emiris (see [CE00] for a complete account), subsequently improved and generalized in [CP93, Stu94], and the incremental algorithm of [EC95], which constructs a rectangular matrix and then obtains a square nonsingular submatrix.

In the case of univariate polynomials, the Bézout and Sylvester matrices have a Hankel-like and Toeplitz-like structure, respectively [BP94]. In the multivariate case, things become more subtle. Canny, Kaltofen and Lakshman [CKL89] studied the structure of Macaulay matrices and proved that multiplication of a Macaulay matrix by a vector is of almost linear complexity in the matrix dimension. Then they applied Wiedemann's technique [Wie86] in order to compute the determinant of such a matrix. Our results generalize their approach. Independently, Mourrain and Pan [MP97, MP98, MP00] generalized [CKL89] in another direction, by formalizing the Toeplitz- or Hankel-like structure of general resultant matrices, including Macaulay, Bézout and Newton matrices. Mourrain and Pan's works, though technically distinct, provide a related viewpoint to our approach. Corollary 5.4 improves proposition 24 of [MP97]; our result can be drawn from this proposition by using a special set of points, such as those of algorithm 4.5.

An auxiliary issue (also important in its own right) is to devise algorithms for multiplying sparse multivariate polynomials within the computational complexity bounds expressed via the support cardinality or the Newton polytope. The existing general bounds are interesting only in the dense case [BP94] since they require at least d^n operations, where n is the number of variables and d is the maximum degree in any one variable. Sparse interpolation has received a lot of attention; see the algorithms in [KL88, Zip93], supporting complexity linear in the product of n, the maximum degree in any single variable and a bound on the number of monomials. Section 4 improves these bounds by exploiting the structure of nonzero terms, and generalizes [CKL89] from completely dense supports to arbitrary supports. Alternative models of sparseness have been studied, including straight-line programs, Khovanskii's fewnomials (to which our results apply), and Vasiliev's density model (under which evaluation requires at least 2^n operations).

Some results of sections 4-7 appeared in preliminary form in [EP97].

3. A fast numerical rank test

We describe an efficient and numerically stable method for testing whether a rectangular matrix has full rank and determining its rank. By Lanczos' method, we reduce the problem to vector-by-matrix multiplication, thus exploiting structure and achieving stability.

An exact-arithmetic Las Vegas algorithm was presented in [EP96, alg. 3.1] for testing whether an $a \times c$ matrix M, with $a \geq c$, has full rank. It relied on two results. First, the preconditioning techniques of [KS91] (see also [Pan96b, fact 7.2]). Second, the extension of [Wie86, thm. 1] given in [KP91, lem. 2]. The time and space complexities are, respectively, $O(cC + ac \log c)$ and $O(G + a)$, where C and G are the time and space complexities of premultiplying a $1 \times a$ vector by matrix M. The algorithm yields as a by-product the determinant of a square matrix and can be modified in a straightforward way to yield an algorithm that finds the rank of a rectangular matrix in $O(cC \log c + ac \log^2 c)$ ops and $O(G + a)$ storage; see also [EP97] for a brief account of this algorithm.

This approach would typically be implemented by modular arithmetic, thus introducing some additional probability of error. However, on modern day computers, fixed-precision floating point arithmetic can be substantially faster. This algorithm cannot take advantage of this feature because rounding-off to a fixed number of digits would cause it to be numerically unstable. The reason is that the vector-matrix products computed by Wiedemann's algorithm, denoted by $M^i v$ or $v^T M^i$, for some column vector v, become close to each other for larger i. This motivates the following approach, which improves the numerical method of [EP96, alg. 3.2] and [EP97, thm. 3.4].

Over the complex field $\mathbf{C}$ or its subfields, we test by a *floating point* computation whether matrix M has full numerical rank, that is, whether M has c singular values whose moduli exceed a fixed small positive tolerance value ϵ. Therefore we have to deal with the symmetrization of M, either implicit or explicit. We could have used any black box algorithm for computing the Singular Value Decomposition (SVD) of M, such as the customary SVD algorithms found in [GV96]. Based on [Pan96a], we shall instead describe a much less costly algorithm for computing the numerical rank, which exploits the structure of M and avoids computing SVD. Numerical nonsingularity is stronger than usual nonsingularity unless ϵ is replaced by 0, in which case the two definitions coincide.

ALGORITHM 3.1. (NUMERICAL RANK COMPUTATION) *Input:* An $a \times c$ matrix M, where $a \geq c$, over the complex field or its subfield, and a positive ϵ.
Output: The numerical rank of M with respect to the tolerance ϵ.

Computations:

1Apply Lanczos' algorithm to compute the orthogonal similarity transformation of

the Hermitian matrix $M^H M$ into a tridiagonal matrix $T = QM^H MQ^H$, where Q is a $c \times c$ matrix such that $QQ^H = I_c$. On Lanczos' algorithm see, for instance, [GV96, ch.9] or [BP94, alg. 2.3.1].

2Let T_i be the $i \times i$ leading principal submatrix of T and compute the values $p_i(\epsilon^2) = \det(I_i - \epsilon^2 T_i)$, for $i = 1, \ldots, c$.

3Let $s \in \mathbf{N}$ be the number of sign changes in the sequence $(p_1(\epsilon^2), \ldots, p_c(\epsilon^2))$ computed by applying Sturm sequences. Output numerical rank $c - s$.

Correctness of the algorithm follows from the well known result that $c - s$ is the number of the eigenvalues of $M^H M$ exceeding ϵ^2 (see, for instance, [GV96, Par80]) which, in turn, equals the number of singular values of M exceeding ϵ. The calculation of numerical rank includes two well-known numerical linear algebra subtasks, namely matrix tridiagonalization by Lanczos' method and a Sturm sequence computation. Since the main operation in Lanczos' method is vector-matrix multiplication, both subtasks only involve field operations, which could also be performed by rational arithmetic and hence exactly, albeit with a higher cost.

Stage 1 performs $O(c)$ vector-by-matrix multiplications involving $M^H M$. Each of these multiplications reduces to one premultiplication and one postmultiplication of the $a \times c$ matrix M by a $1 \times a$ and a $c \times 1$ vector, respectively. The computational cost is bounded by $O(cC + c^2)$ ops. The last two stages have complexity $O(c^2)$ ops since T is $c \times c$ and tridiagonal. All operations are performed over fixed precision floating point numbers, hence the bit complexity is asymptotically the same as the arithmetic complexity. This discussion is summarized in the following statement.

THEOREM 3.2. *Let C and G be the maximum time and space complexities, respectively, of pre- and post-multiplying a complex $a \times c$ matrix M, with $a \geq c$, by a vector. Then there exists a randomized algorithm that computes the numerical rank of M, within a given tolerance $\epsilon > 0$, in $O(cC + c^2)$ ops and $O(G + c)$ storage space.*

In spite of using a random vector to start Lanczos' algorithm, its numerical performance is practically valid, and the algorithm is one of the most popular methods in numerical linear algebra. Moreover, it is implemented in publicly available software packages. This responds to the numerical stability issue raised with respect to the algorithm of [EP96]. The key feature is that Lanczos' algorithm ensures the orthogonality of the computed vectors, though at the price of a certain slowdown when it is applied to structured matrices. Numerical stability also characterizes the Sturm sequence computation. Interestingly, the use of Sturm theory in order to compute the number of real roots exceeding a certain value is one of the most important tools in symbolic computation; see, e.g. [BCL82].

It is known that Lanczos' algorithm uses a random vector and, hence, it is a Las Vegas algorithm. In other words, it may fail with a small bounded probability, but never produces an incorrect result. More specifically, if the tridiagonalization of step 1 is not achieved in $O(cC)$ ops, then the procedure is stopped and reports failure. To remove randomization, one would have to accept worst-case time and space complexities in $O(c^2 C)$ and $O(G + c^2)$.

Yet another option is to use Monte Carlo randomization, that is, to accept the possibility of wrong output with a bounded small probability. Then, a simplified version of the above algorithm that tests rank deficiency is the following. A matrix is said to have full rank with respect to a given tolerance if its minimum singular value exceeds this tolerance.

ALGORITHM 3.3. (NUMERICAL RANK TEST) *Input:* An $a \times c$ matrix M, where $a \geq c$, over the complex field or its subfield, and a positive ϵ.
Output: A single bit 0 or 1 indicating whether M has full rank or not with respect to the tolerance ϵ with high probability.

Computations:

1Apply Lanczos' algorithm to compute an upper estimate τ_1^2 of the square of the largest singular value σ_1 of M. Equivalently, σ_1^2 is the largest eigenvalue of $M^H M$ and $\tau_1^2 \stackrel{>}{\sim} \sigma_1^2$.

2Apply Lanczos' algorithm again to compute an upper estimate τ_2^2 of the largest eigenvalue $\tau_1^2 - \sigma_c^2$ of the matrix $\tau_1^2 I_c - M^H M$. Here σ_c^2 is the smallest eigenvalue of $M^H M$ or, equivalently, σ_c is the smallest singular value of M. Since $\tau_2^2 \stackrel{>}{\sim} \tau_1^2 - \sigma_c^2$, then $\tau_1^2 - \tau_2^2$ is a lower estimate of σ_c^2.

3If $\tau_1^2 - \tau_2^2 < \epsilon^2$, then output bit 1 indicating rank deficiency. Otherwise output bit 0 indicating full rank.

Theoretically, the algorithm may produce a wrong output if the random vector for Lanczos' algorithm is chosen unsuccessfully, causing degeneration in step 1 or 2 or both. This has a low probability, fully estimated in [KW92, KW94].

THEOREM 3.4.[KW92, thm. 4.2(a)] *Let $\delta \in [0,1)$ be the relative error of Lanczos' algorithm in approximating the largest eigenvalue of a symmetric positive definite $c \times c$ matrix. Let k be the number of multiplications of the matrix by a vector. If k is at least as large as the number of distinct eigenvalues, then the algorithm will always produce an approximation within δ. For general k, the probability that the algorithm fails is bounded by*

$$1.648\sqrt{c}e^{-\sqrt{\delta}(2k-1)}.$$

Practically, the degeneration is much less likely since roundoff errors usually remove the computed vectors from the subspace of degeneracy [GV96, ch. 9]. If we agree to include $ac-1$ comparisons, we may replace step 1 by the computation of a deterministic upper estimate, namely the square of the largest absolute value of any entry of M. This is at least as large as the largest entry of $M^H M$ and, hence, an upper bound on σ_1^2. Then, the randomization in Lanczos' algorithm, which is potentially a source of wrong output, will be confined to step 2.

The complexity of algorithm 3.3 is $O(Ck)$ ops and $O(G+c)$ storage space, where k is the number of iterations required by Lanczos' algorithm. Since k can be smaller than c, this algorithm is possibly faster than algorithm 3.1. Theorem 3.4 implies that k depends on the probability of error that we wish to guarantee.

4. Exact sparse polynomial arithmetic

We present exact-arithmetic algorithms for polynomials defined by their supports, or nonzero terms, as is the case in the context of sparse elimination theory. In particular, we examine support evaluation and polynomial multiplication.

We will work in the ring of Laurent polynomials $P = K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$, where K is any given field of characteristic zero. The *support* of $f \in P$ is a subset of $\mathbf{Z}^n$ denoted $\mathrm{supp}(f)$ and containing all the exponent vectors of monomials with nonzero coefficients in f. If $S = \mathrm{supp}(f) \subset \mathbf{Z}^n$, then

$$f = \sum_{a \in S} c_a x^a, \qquad x^a = \prod_{i=1}^{n} x_i^{a_i},$$

where $a = (a_1, \ldots, a_n) \in \mathbf{Z}^n$, $c_a \in K$. In dealing with supports, we slightly abuse terminology and speak of a monomial in a support, referring to the monomial defined by the integer point representing its exponent. In the sequel, we assume, without loss of generality, that all polynomial supports contain the origin; this can be achieved by a translation of the supports.

For every polynomial, there is an associated *Newton polytope*, which is the convex

hull of the support. Newton polytope generalizes the classical notion of total degree of an n-variate polynomial; for a completely dense polynomial, the Newton polytope is the n-dimensional unit simplex. Define the *Minkowski sum* $A+B$ of two point sets A and B in $\mathbf{R}^n$ as the point set $A+B = \{a+b \mid a \in A, b \in B\}$. If A, B are convex polytopes, then so is $A+B$. For further information on sparse elimination see [EC95, Emi96, Stu94] and their references.

The following algorithms and their complexity analysis are of independent interest as they demonstrate that the complexity of polynomial multiplication, evaluation and interpolation on some special sets of points depends on the corresponding support cardinalities and Newton polytope volumes; these two are asymptotically equivalent. This discussion complements the known results on sparse evaluation and interpolation by settling the case where sparseness is measured by the support.

To simplify the notation, we assume when we discuss evaluation that all monomials have non-negative exponents.

LEMMA 4.1. *Consider a set S of s positive integers, such that $S \subset \mathbf{N} \cap [0, d]$, for some positive integer d. If we are given a value p, we may evaluate all powers of p with exponents in S by using $O^*(s+\sqrt{d})$ ops and $O(s)$ storage space.*

PROOF. Let $b = \lfloor \sqrt{d} \rfloor$ and represent every $a \in S$ as $a = i + bk$, where $0 \leq i < b$ and $0 \leq k < d/b$. Let $S_0 = \{i : i + bk \in S\}$ and $S_1 = \{k : i + bk \in S\}$. Then, compute p^i, for all $i \in S_0$, in $O(b)$ ops, and $q^k = p^{bk}$, for all $k \in S_1$, in $O(d/b)$ ops. Then we may compute $p^a = p^i q^k$ for all $a \in S$, in s ops. □

ALGORITHM 4.2. (SUPPORT EVALUATION) This algorithm evaluates a given vector set at a given value per coordinate.
Input: A set of monomials or, equivalently, of their exponent vectors $S \subset \mathbf{Z}^n$, and scalar values $p_1, p_2, \ldots, p_n$.
Output: The values of all monomials with exponents in S at the given values $p_1, \cdots, p_n$.

Computations:

1 Compute n sets of positive integers $S_1, S_2, \ldots, S_n$, $S_i = \{a_i \in \mathbf{N} : \exists a = (a_1, \ldots, a_i, \ldots, a_n) \in S\}$ representing all powers of the i-th variable encountered among the monomials in S.
2 Compute all powers p_i^a, for all $a \in S_i$ and $i = 1, 2, \ldots, n$, as in the proof of lemma 4.1.
3 Compute the values of all monomials in S by multiplying, for each monomial, at most n powers computed in the previous step.

LEMMA 4.3. *Consider a set S of s monomials in n variables, such that the exponent of every monomial in the ith variable lies in $[0, d]$, for $i = 1, 2, \ldots, n$. Given n scalar values $p_1, p_2, \ldots, p_n$, one may evaluate all the monomials of S at these values in $O^*(sn+n\sqrt{d})$ ops and $O(sn)$ space by the above algorithm.*

PROOF. Apply algorithm 4.2. Its first stage takes $O^*(sn)$ ops. The second stage requires $O^*(s+\sqrt{d})$ ops, for each i, by lemma 4.1. The final stage 3 can be performed in $O^*(sn)$ ops. □

REMARK 4.4. *The algorithm uses space in $O(s)$ in addition to the space required to store the input exponent vectors. The latter is in the worst case in $O(sn)$, but can be reduced to $O(s)$ if the exponent vectors have "short" entries so that each is stored in constant space. This is the typical case in practice, hence the overall space complexity becomes $O(s)$.*

The following multiplication algorithm extends the approach of [CKL89, sect. 3],

based on the widely used evaluation-interpolation scheme, with node sets from a special customary class, also used in [BP94, KL88, Zip93]. We will focus on multiplication, but our algorithm improves sparse evaluation and interpolation as a by-product.

ALGORITHM 4.5. (SPARSE POLYNOMIAL MULTIPLICATION)
Input: n-variate polynomials $f, g \in P$ with supports $A, B \subset \mathbf{Z}^n$, respectively. Also given is a set of points $S \subset \mathbf{Z}^n$ such that $A + B \subset S$, so that S contains the support of fg.
Output: The product fg.

Computations:

1Let $A = \{a_1, \ldots, a_{|A|}\}$, with each $a_k \in A$ written as $(a_{k1}, \ldots, a_{kn})$. Let $S = \{m_1, \ldots, m_s\} \subset \mathbf{Z}^n$, where s is the cardinality of S. Pick n distinct primes $p_1, \cdots, p_n$, supposed to be readily available.

2Compute the values v_k of the monomials in A at point $(p_1, \ldots, p_n)$, for $k = 1, 2, \ldots, |A|$. Since the k-th monomial is $x^{a_k} = \prod_{i=1}^n x_i^{a_{ki}}$, it follows that $v_k = \prod_{i=1}^n p_i^{a_{ki}}$. Observe that the j-th power $v_k^j = \prod_{i=1}^n (p_i^j)^{a_{ki}}$ is the value of x^{a_k} at point $(p_1^j, \ldots, p_n^j)$, whose coordinates are also j-th powers. Therefore, multiplication of the row vector of the coefficients of f by the $|A| \times s$ matrix

$$\begin{bmatrix} 1 & v_1 & \cdots & v_1^{s-1} \\ \vdots & \vdots & & \vdots \\ 1 & v_{|A|} & \cdots & v_{|A|}^{s-1} \end{bmatrix}$$

expresses the evaluation of f at the points $(p_1^j, \cdots, p_n^j)$ for $j = 0, 1, \cdots, s-1$. We append rows of powers $1, v_i, \cdots, v_i^{s-1}$, for distinct v_i, $i = |A|+1, \cdots, s$, to the matrix above in order to obtain an $s \times s$ Vandermonde matrix V (see example 4.6).

3Let c_f be the $s \times 1$ column vector whose first $|A|$ entries are the coefficients of f, in the order defined by an arbitrary but fixed monomial sequence $(a_1, \ldots, a_{|A|})$; let the last $s - |A|$ entries be zeros. Then the column vector of the values of f at $v_1, \ldots, v_{|A|}$ is expressed as $V^T c_f = (V^T V)(V^{-1} c_f)$. Compute $V^T V$, $V^{-1} c_f$ and their product as discussed in [CKL89, sect. 3a] or [BP94, ch. 2]; for improving the numerical stability or the constant of the complexity bound in solving transposed Vandermonde systems, see [Pan01, sect. 3.4,3.6]. Analogously evaluate the polynomial g at the same points. Then multiply the values of f and g pointwise, thus computing the values of fg at every point $(p_1^j, \ldots, p_n^j)$, $j = 0, \ldots, s-1$.

4Let l_{fg} and c_{fg} denote the vectors of the product values and of the unknown coefficients fg ordered, respectively, by $(p_1^j, \ldots, p_n^j)$ for $j = 0, \ldots, s-1$ and by a fixed monomial sequence $(m_1, \ldots, m_s)$. Compute w_i as the value of m_i at $(p_1, \ldots, p_n)$ and let W be the $s \times s$ Vandermonde matrix $[w_i^{j-1}]$, analogous to V in step 2. Solve the transposed Vandermonde system $W^T c_{fg} = l_{fg}$, e.g. by applying the algorithm of [KL88]. The solution c_{fg} defines fg; some coefficients are zero if and only if the support of the product is a proper subset of S.

EXAMPLE 4.6. *For illustration, let* $f_1 = c_0 + c_1 x + c_2 xy$ *and* $g_1 = s_0 + s_1 x$, *which means that* $c_f = [c_0, c_1, c_2, 0, 0]$ *and* $c_g = [s_0, s_1, 0, 0, 0]$. *Moreover,* $A = ((0,0),(1,0),(1,1))$, $B = ((0,0),(1,0))$, *and* $S = ((0,0),(1,0),(1,1),(2,0),(2,1))$. *In evaluating* f_1, *the corresponding* 5×5 *Vandermonde matrix* V *can be*

$$V = \begin{bmatrix} 1 & v_1 & v_1^2 & v_1^3 & v_1^4 \\ 1 & v_2 & v_2^2 & v_2^3 & v_2^4 \\ 1 & v_3 & v_1^2 & v_1^3 & v_3^4 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix}.$$

THEOREM 4.7. *Given n-variate polynomials $f, g \in P$ with supports $A, B \subset \mathbf{Z}^n$, respectively, and given a point set $S \subset \mathbf{Z}^n$ such that $A + B \subset S$, the product fg can be computed by algorithm 4.5 by using $O^*(sn + n\sqrt{d})$ ops and $O(sn)$ space, where d is the maximum degree of each input polynomial in any variable and $s = |S|$.*

PROOF. In steps 2 and 4, the Vandermonde matrices are defined by at most s values v_i and w_i, respectively. These are the values of all monomials from the set A at a point $(p_1, \cdots, p_2)$; according to lemma 4.3, such values can be computed in $O^*(sn + n\sqrt{d})$ ops since the maximum degree of the product polynomial in each variable is bounded by $2d$. For interpolation, note that all monomials in fg belong to S. The storage requirement is $O(sn)$. Steps 3 and 4 take each $O(s \log^2 s)$ ops and $O(s)$ storage space, due to the techniques of [CKL89, KL88] (also see [BP94, sect. 2.6]). Both estimates exploit the structure of the Hankel matrix $V^T V$. □

REMARK 4.8. *The entries of $V^T V$ equal the power sums of the roots of the polynomial $\prod_i (x - v_i)$, in the notation of the above algorithm. These entries can be computed via the identities involving the symmetric functions of the corresponding coefficients, by solving a Toeplitz linear system of $2|A|$ equations. In fact, this Toeplitz linear system is* triangular, *so its solution is substantially simpler than that stated in [CKL89, sect. 3a]. A slightly simpler way reduces the computation to a polynomial reciprocal and a polynomial product [Pan97].*

5. The structure of the Newton matrix

In this section, we describe the general problem of constructing Newton matrices, which express the sparse resultant by means of a determinant; we refer the reader to [EC95, Emi96, Stu94] and their references for a comprehensive presentation. The quasi-Toeplitz structure of these matrices is revealed and applied to establishing good upper bounds on the complexity of multiplying a Newton matrix by a row or column vector. These bounds are significantly lower than quadratic in the matrix dimension, and even quasi-linear for premultiplication by a row vector.

Sparse elimination uses certain notions from combinatorial geometry. Given convex polytopes $Q_1, \ldots, Q_n \subset \mathbf{R}^n$ and non-negative $\lambda_1, \ldots, \lambda_n \in \mathbf{R}$, the standard n-dimensional Euclidean volume of the Minkowski sum $\lambda_1 Q_1 + \cdots + \lambda_n Q_n$ is a polynomial in the $\lambda_1, \ldots, \lambda_n$, homogeneous of degree n. The coefficient of the multilinear term $\lambda_1 \cdots \lambda_n$ is defined to be the *mixed volume* of $Q_1, \ldots, Q_n$ and denoted by $MV(Q_1, \ldots, Q_n)$. If the Q_i have integer vertices, as in the case of Newton polytopes, then their mixed volume takes integer values. These facts and a number of equivalent definitions of mixed volume are demonstrated in [Ewa96]; see also [EC95] for computational issues regarding mixed volumes.

Consider a well-constrained polynomial system $f_1, \ldots, f_n \in P = K[x, x^{-1}]$, where K is the base field of characteristic zero. Bernstein's theorem states that the mixed volume of the Newton polytopes associated to the polynomial system of equations $f_1 = \cdots = f_n = 0$ bounds the number of isolated roots of this system in $(\overline{K}^*)^n = (\overline{K} \setminus \{0\})^n$, where $\overline{K}$ is the algebraic closure of the base field. The mixed volume is typically much less than Bézout's bound for sparse polynomial systems. We recall that Bézout's bound on the number of (projective) roots is $\prod_i d_i$, where d_i is the total degree of the polynomial f_i, for $1 \leq i \leq n$.

Now we pass to the context of overconstrained systems $f_1, \ldots, f_{n+1} \in P$. The *sparse resultant* R of polynomials $f_1, \ldots, f_{n+1}$ is an irreducible polynomial in the f_i coefficients, which provides a necessary condition for solvability of the overconstrained system $f_1 = \cdots = f_{n+1} = 0$ over $(\overline{K}^*)^n$, i.e., it vanishes whenever there exists a solution in $(\overline{K}^*)^n$. R is a homogeneous polynomial in the coefficients of each f_i whose degree, denoted $\deg_{f_i} R$, is given by the following mixed volume.

$$\deg_{f_i} R = MV(f_1, \ldots, f_{i-1}, f_{i+1}, \ldots, f_{n+1}). \tag{5.1}$$

The algorithmic problem of computing R is typically solved by constructing square matrices, called *resultant matrices*, whose determinant is ideally R or, more generally, a nontrivial multiple of R. Even in the second case, the resultant matrix suffices for reducing the computation of all roots of $f_1 = \cdots = f_{n+1} = 0$ to a problem in linear algebra; see, for instance, [Emi96].

For a nonempty set of monomials $S \subset \mathbf{Z}^n$, let

$$P(S) = \{f \in P : \mathrm{supp}(f) \subset S\} \subset P$$

be the vector space over some monomial basis in S, of dimension equal to the cardinality $s = |S|$. Hence a polynomial is represented by a vector, and a list of polynomials by a concatenation of vectors.

DEFINITION 5.1. *Let $n+1$ polynomials $f_1, \ldots, f_{n+1} \in P$ have supports $A_1, \ldots, A_{n+1} \subset \mathbf{Z}^n$. Let $B_1, \ldots, B_{n+1} \subset \mathbf{Z}^n$ be the supports of polynomials $g_1, \ldots, g_{n+1} \in P$ such that the linear transformation*

$$\begin{aligned} \mu: \quad & P(B_1) \times \cdots \times P(B_{n+1}) \to P\left(\textstyle\bigcup_{i=1}^{n+1}(A_i + B_i)\right), \\ & [g_1, \ldots, g_{n+1}] \mapsto [g_1, \ldots, g_{n+1}]M = \left[\textstyle\sum_{i=1}^{n+1} g_i f_i\right] \end{aligned} \tag{5.2}$$

is surjective for generic coefficients of the f_i and, moreover, the dimension of the domain of μ is at least as large as the dimension of the range, in other words, $\sum_i |B_i| \geq |\cup_{i=1}^{n+1}(A_i + B_i)|$. Then the matrix M is the transpose of the matrix of μ, has entries in K and has at least as many rows as columns. If, furthermore, $|B_i| \geq \deg_{f_i} R$ for $i = 1, \ldots, n+1$, then this is a sparse resultant, *or* Newton, matrix *for the system $f_1, \ldots, f_{n+1}$.*

Observe that the coefficients of f_i are generic, or symbolic, and those of g_i are immaterial. Each entry of M is either zero or equal to a coefficient of some f_i. The rows of M are indexed by the points in B_i, so that the row corresponding to $b \in B_i$ expresses the polynomial $x^b f_i$. The columns of M are indexed by the points in $\cup_i(A_i + B_i)$, which is precisely the support of $\sum_i g_i f_i$. In short, Newton matrices are constructed in the same way as Sylvester and Macaulay matrices [vdW50] and correspond, therefore, to the transpose of the linear transformation above.

Keeping with the philosophy of this paper, we store a Newton matrix by storing only the B_i and f_i, $i = 1, \ldots, n+1$, hence using $O(cn)$ space, where c denotes the number of matrix columns and bounds the cardinality of any B_i. This space bound relies on the hypothesis that a multi-index representing an integer exponent vector or, equivalently, a monomial takes $O(1)$ space. This is assumed in the rest of the paper and is justified by the observation that, typically, the list of n maximum degrees in any variable (denoted d) can be stored in constant amount of space.

The following well known theorem is the basis for computing nontrivial multiples of the resultant [CE00, vdW50].

THEOREM 5.2. *Consider any maximal nonzero minor (determinant of a maximal submatrix) D of a Newton matrix M. Then D is a nontrivial multiple of the sparse resultant R.*

PROOF. If there is a common zero $\xi \in (\overline{K}^*)^n$ for $f_1, \ldots, f_{n+1}$, then it is a common zero for all polynomials in the range of μ. Consequently, this range cannot contain any monomial, because the monomial value at ξ cannot be zero. Therefore, μ is not surjective, i.e., every maximal minor of the matrix M vanishes on the coefficient specializations for which there exists a solution in $(\overline{K}^*)^n$. Consider D and R as polynomials in the coefficients of input polynomials f_i and compare the two sets (or algebraic varieties) in the space of these coefficients on which D and R vanish. According to the above argument, D vanishes on the zero set (or variety) of R and, since R is irreducible, Hilbert's Nullstellensatz [vdW50] implies that R must divide D. (This holds for every maximal minor.) Furthermore, the hypothesis $D \neq 0$ implies that this is a nontrivial multiple. □

Our approach to demonstrating the matrix structure proceeds by studying its properties concerning *multiplication* by a vector, namely the fact that this operation has complexity substantially lower than quadratic in the matrix dimension. This essentially amounts to revealing the quasi-Toeplitz structure of M, which reduces both pre- and post-multiplication by a vector to polynomial multiplication. In [MP97] the quasi-Toeplitz and quasi-Hankel structure of all types of resultant matrices is formalized, including Bézout and Dixon matrices. The question remains how to perform this multiplication. Straightforward application of theorem 4.7 is non-optimal. An improvement is possible by the approach of [MP97, prop. 24] in the case of premultiplication. Yet, by modifying algorithm 4.5 we obtain a further improvement, whereas postmultiplication is immediately reduced to premultiplication due to the following powerful general theorem.

THEOREM 5.3. (TELLEGEN'S THEOREM)[BCS97, thm. 13.20] *Let M be an $a \times c$ matrix with no zero rows and $L(A)$ denote the complexity of postmultiplying any rectangular matrix A by a vector of appropriate dimension. Then $L(M) = L(M^T) + c - a$.*

We now consider *premultiplication* of M by an a-dimensional vector. Polynomial f will capture the structure of M. If M is viewed as an $(n+1) \times 1$ block matrix, the blocks would be of Toeplitz type in the univariate (Sylvester matrix) case. If the polynomial associated to an $a \times c$ Toeplitz matrix with (i, j)-th entry t_{j-i} is $f = \sum_{k=1-a}^{c-1} t_k x^k$, then premultiplication of the matrix by a row vector $[v_1, \ldots, v_a]$ is expressed by polynomial multiplication fg, where $g = \sum_{i=1}^{a} v_i x^i$. So f must account for the various blocks and, in the multivariate case, take into consideration the fact that the structure is not exactly Toeplitz, but rather *quasi-Toeplitz*. The first issue is addressed by introducing a new variable x_{n+1} in order to index the k-th block, $k = 1, \ldots, n+1$, by x_{n+1}^k. The second issue is addressed by indexing the rows by the monomials in B_i and the columns by the monomials in $\cup_{i=1}^{n+1}(A_i + B_i)$.

In the case of premultiplication of M by a row vector the polynomial associated to the quasi-Toeplitz matrix is

$$f = \sum_{i=1}^{n+1} x_{n+1}^{-i} f_i(x).$$

Any row vector can be decomposed into subvectors of length $|B_i|$, $i = 1, \ldots, n+1$, and the respective entries can be thought of as the coefficients of a polynomial g_i with support B_i, just as in expression (5.2). The polynomial associated to the vector contains monomials of the type $x_{n+1}^i x^b$, where $b \in B_i$, $i = 1, \ldots, n+1$. Hence, the polynomial of the input vector is

$$g = \sum_{i=1}^{n+1} x_{n+1}^i g_i(x), \quad \text{where} \quad g_i = \sum_{b \in B_i} g_{ib} x^b,$$

and g_{ib} is an appropriate element of the given vector. Then the premultiplication by any vector is reduced to computing $\sum_i g_i f_i$. The latter has support equal to the set of all column monomials and represents the constant coefficient of fg regarded as a univariate polynomial in x_{n+1}.

$$\sum_{i=1}^{n+1} f_i(x) g_i(x) = \sum_{q \in \cup_i (A_i + B_i)} x^q \sum_{i=1}^{n+1} \sum_{e \in A_i : q-e \in B_i} c_{ie}\, g_{i(q-e)},$$

where $f_i = \sum_{e \in A_i} x^e c_{ie}$.

EXAMPLE 4.6 (CONTINUED) Let $f_1 = c_0 + c_1 x_1 + c_2 x_1 x_2$, with ordered support $A_1 = ((0,0), (1,0), (1,1))$, let $B_1 = ((0,0), (1,0))$ and consider the subsequence $((0,0), (1,0), (1,1), (2,0), (2,1))$ of S. For an arbitrary row vector $[s_0, s_1, \ldots]$, the first entries can be

thought of as the coefficients of polynomial $g_1 = s_0 + s_1 x_1$. Then, premultiplication by this vector starts as follows:

$$\begin{array}{c} \begin{array}{cccc} 1 & x_1 & & \end{array} \\ \left[\begin{array}{cccc} s_0 & s_1 & \cdots & \end{array}\right] \end{array} \left[\begin{array}{ccccc} 1 & x_1 & x_1x_2 & x_1^2 & x_1^2x_2 \\ c_0 & c_1 & c_2 & 0 & 0 \\ 0 & c_0 & 0 & c_1 & c_2 \\ & & \vdots & & \end{array}\right] \begin{array}{c} f_1 \\ x_1 f_1 \\ \vdots \end{array} =$$

$$= \begin{array}{ccccccc} 1 & & x_1 & & x_1x_2 & x_1^2 & x_1^2x_2 \\ \left[\begin{array}{c} \end{array}\right. & s_0c_0 & s_0c_1 + s_1c_0 & s_0c_2 & s_1c_1 & s_1c_2 & \left.\begin{array}{c} \end{array}\right] + \cdots . \end{array}$$

To the right of the matrix, we mark the polynomials filling in the rows, and above the matrix and the vectors, we show the monomials indexing the columns or entries, respectively.

We can now describe a modification of algorithm 4.5 for computing the interesting part of fg, whose complexity stays within the same asymptotic bounds. Suppose $S = \{m_1, \ldots, m_s\} \subset \mathbf{Z}^n$ is given, such that $\cup_i (A_i + B_i) \subseteq S$.

1Pick primes $p_1, \ldots, p_n, p_{n+1}$ and compute the values $w_k = \prod_{i=1}^{n} p_i^{m_{ki}}$, $k = 1, \ldots, s$. This takes $O^*(sn + n\sqrt{d})$ ops and $O(sn)$ storage by lemma 4.3, where d is the maximum degree of f_i, g_i in any variable.

2Let $A_i' = \{(e, -i) : e \in A_i\} \subset \mathbf{Z}^{n+1}$. Evaluate all A_i' monomials, for $i = 1, \ldots, n+1$, by multiplying the appropriate w_k by p_{n+1}^{-i}. This takes $O^*(s)$ per polynomial and total space $O(sn)$ to compute all necessary values. The new values define Vandermonde matrix V_f expressing evaluation of f, and whose dimension is $|\mathrm{supp}(f)| \leq sn$, since $\mathrm{supp}(f) = \cup_i A_i'$ and $A_i \subset S$. Construct the coefficient vector c_f and compute the evaluation vector $l_f = V_f^T c_f$. Analogously proceed for the polynomial g. Then multiply pointwise the two evaluation vectors in order to obtain l_{fg}. All operations have complexity $O^*(sn)$ by the proof of theorem 4.7.

3Let W be the Vandermonde matrix defined by the evaluation of the support monomials of f and g computed above; its dimension is $(2n+1)s$. Solve $W^T c_{fg} = l_{fg}$ for the coefficient vector c_{fg} of $\sum_{i=1}^{n+1} f_i g_i$ and return the subvector corresponding to the constant monomials with respect to x_{n+1}. This step has complexity $O^*(sn)$.

The algorithm can clear denominators by multiplying all monomials by x_{n+1}^{n-1}, then returning the coefficient of x_{n+1}^{2n} in the product polynomial. Now an immediate corollary of theorem 4.7 is the following.

COROLLARY 5.4. *Consider polynomials $f_i, g_i \in P$, $i = 1, \ldots, n+1$. Let $A_i, B_i \subset \mathbf{Z}^n$ be the respective supports and let $S \subset \mathbf{Z}^n$ be such that $\cup_i (A_i + B_i) \subseteq S$. Then computing $\sum_{i=1}^{n+1} f_i g_i$ has time complexity $O^*(sn + n\sqrt{d})$ and space complexity $O(sn)$, where d is the maximum degree of f_i, g_i in any variable. This implies that premultiplication of M by a row vector can be performed within these complexity bounds.*

The second important property is that *postmultiplication* of M by a c-dimensional vector can also be performed substantially faster than the straightforward quadratic method. This is a corollary of Tellegen's theorem 5.3.

COROLLARY 5.5. *Consider Newton matrix M defined by n-variate polynomials f_i with supports A_i and by support sets $B_i \subset \mathbf{Z}^n$, $i = 1, \ldots, n+1$. Let s and d be as above. Then computing the product Mv for some column vector v has time and space complexity in $O^*(sn + n\sqrt{d})$ by Tellegen's theorem 5.3.*

S can be taken to be precisely $\cup_i(A_i + B_i)$, the set of the monomials indexing the columns of M, hence $c = |S|$. Typically, the exact computation of support S is expensive, so we can bound it by the integer lattice points lying in the Minkowski sum of the Newton polytopes of the f_i. In the dense context, $s = |S|$ was bounded simply as a function of the degrees, thus yielding a quite loose bound. This development culminates with the following result.

THEOREM 5.6. *Let M be an $a \times c$ Newton matrix of the transformation of (5.2) where $a \geq c$, and let v be a $1 \times a$ vector, both with constant entries. Then computing the vector vM takes $O^*(cn + n\sqrt{d})$ ops and $O(cn)$ storage space, where d is the maximum degree of $f_1, \cdots, f_{n+1}$ in any one variable. Computing vector Mv, where v is a $c \times 1$ column vector with constant entries, has time and space complexity in $O^*(cn + n\sqrt{d})$.*

An improvement of practical interest is possible when multiplication of M by a row or column vector must be repeated several times, as in computing the rank of M. Namely, the first steps of the above algorithms, which evaluate the supports and vector l_f, may be performed only once.

An interesting extension for polynomial system solving is when the matrix entries are univariate polynomials in an indeterminate, other than the variables eliminated by the resultant [Emi96, vdW50]. This means that in the course of performing the computations above, a typical vector by which M is multiplied has entries that are polynomials in this indeterminate. This would increase the time complexity by an additional quasi-linear factor in the maximum degree of the input polynomials in this indeterminate.

6. Incremental matrix construction

In this section we sketch the incremental algorithm for constructing a Newton matrix, proposed in [EC95], and reduce its time complexity by one order of magnitude; the original algorithm had cubic complexity in the matrix dimension. The incremental construction yields the smallest Newton matrices among all existing algorithms and, moreover, constructs optimal matrices in several cases, including all cases where optimal matrices provably exist. An implementation is available and experiments have shown that the matrix dimension is typically within a factor of three of the optimal.

The matrix is constructed by adding integer points to the candidate sets B_i, until a Newton matrix is found. For every intermediate candidate matrix with at least as many rows as columns, the algorithm tests whether it has full rank. To formalize, let Q_i denote the Newton polytope of f_i and define Minkowski sums $Q_{-i} = \sum_{j \neq i} Q_j$, $i = 1, \ldots, n+1$, and $Q = \sum_j Q_j$. Then the set of row monomials is the disjoint union of sets $B_i \subset Q_{-i} \cap \mathbf{Z}^n$. The set of column monomials always lies in Q and, at any stage of the algorithm, it is defined to be $\cup_i(A_i + B_i)$ for the B_i at this stage. The algorithm linearly orders all points in each Q_{-i}, so that there is a well-defined rule for incrementing the sets B_i for $i = 1, \ldots, n+1$. As the B_i are incremented, the algorithm constructs successively larger matrices until a Newton matrix is found. Instead of using generic coefficients for the f_i, in practice we use random integer values.

Initially B_i contains the optimal number of points, namely $\deg_{f_i} R$, given by identity (5.1), $i = 1, \ldots, n+1$. The number of incremental steps is bounded by the final number of rows, because every step adds at least one point to some B_i. In practice, every step adds more than one point; in this regard, computing the matrix rank provides useful information. The matrix obtained at each step is characterized by the same structure as the Newton matrix. The idea is, therefore, to exploit the structure of the rectangular matrix in order to accelerate each rank test.

LEMMA 6.1. *Let polynomial system $f_1, \ldots, f_{n+1} \in P$ and let M be an $a \times c$ matrix constructed in the course of the incremental algorithm, with numeric entries, such that $a \geq c$. Computing the numerical rank of M within some given tolerance requires $O^*(c^2 n + cn\sqrt{d})$ ops and $O^*(cn + n\sqrt{d})$ storage space.*

PROOF. The proof follows from theorem 3.2 if we apply theorem 5.6 to bound the cost of a vector-by-matrix multiplication. □

To obtain a regular square Newton matrix from a full-rank rectangular incremental matrix M we rely on an observation of [EP96] which, itself, uses the probabilistic construction in [Pan96b, fact 7.2]; see, alternatively, [EP96, lem. 3.1]. The complexity of this step is dominated by the matrix construction.

LEMMA 6.2. [EP96, prop. 5.4] *Let an $a \times c$ matrix M be of full rank and assume that L is a unit $a \times a$ lower triangular Toeplitz matrix, with its $a-1$ subdiagonal entries randomly chosen from a fixed finite set T. Let W be the $c \times c$ trailing principal submatrix of LM. Then, W is nonsingular and $\det W$ is a multiple of the sparse resultant R with a probability at least $1 - c/|T|$.*

The following theorem gives an *output-sensitive* upper bound on the worst-case complexity of the incremental algorithm for computing a Newton matrix. In the rest of this section, we ignore the cost of computing the monomial set indexing the columns of the Newton matrix. For the sake of simplicity, we make the *hypothesis* that $a = O^*(cn)$; this is a reasonable assumption, based on our experience.

THEOREM 6.3. *Assume that the given $n+1$ polynomials in n variables have numeric coefficients and let t be the number of rank tests required by the incremental algorithm of [EC95] in order to construct M. Assume that the maximum degree in any variable is $d = O(c^2)$ and that we are given some numeric tolerance ϵ. Then, using the numerical algorithm 3.1 with complexity bounded by theorem 3.2 yields an overall time complexity in $O^*(c^2nt)$ and space complexity in $O^*(cn)$.*

The previous time complexity bound was $O(a^2c)$ from [EC95, lem. 7.2] and the space complexity was $O(ac)$. These bounds follow from the fact that the algorithm tests the nonsingularity of several matrix candidates, by applying an incremental version of LU-decomposition, which is performed in place. Table 2 shows the various parameters in examples studied in [EC95, Emi97] using our implementation in `C`, publicly available at `http://www-sop.inria.fr/galaad/logiciels/emiris/soft_alg.html`. This implementation relies on LAPACK procedures for the numerical operations [ABB+95]. The first three examples are multihomogeneous systems with three groups of two, one and one variables respectively, where the corresponding degrees are given in the table and the following two are different expressions of the cyclic 6-root problem; see [EC95] for details. The last example is the Stewart platform from parallel robot kinematics; see [Emi97].

Table 2. Performance of the incremental algorithm

type	n	$\deg R$	c	a	t
$(2,1,1;2,1,1)$	4	240	260	260	5
$(2,1,1;2,2,1)$	4	480	592	690	43
$(2,1,1;2,2,2)$	4	960	1120	1200	≤ 49
original cyclic	6	290	849	1457	180
simplified cyclic	5	66	102	121	18
Stewart platform	6	214	405	526	68

Clearly, an important issue concerns a formal bound on the number of singularity tests t.

LEMMA 6.4. *Consider the incremental algorithm described above and suppose that a valid Newton matrix is defined by point sets B_i, $i = 1, \ldots, n+1$. If any or all of the B_i are incremented (by following the ordering on the respective set $Q_{-i} \cap \mathbf{Z}^n$), then the new matrix is again a valid Newton matrix.*

This lemma suggests the following heuristic rule to minimize t: At every incremental step, for a fixed D, the algorithm adds at least D new rows, by appending as many points to the corresponding sets B_i. Let a_1 denote the number of rows in the first full-rank matrix encountered by the algorithm, and let $a_0 < a_1$ be the number of rows in the last (hence largest) rejected candidate matrix. The algorithm tries to optimize the number of rows by performing a *binary search in the heuristic range* $(a_0, a_1]$. Hence, the total number of tests is roughly $a_1/D + \log D$. We applied the new algorithm for D roughly equal to $\deg R$ to the 1st, 4th and 5th inputs in table 2 and obtained a matrix with the same number of columns after 7, 15 and 5 tests respectively. Our experiments showed that a_1 increases at most at the same rate as $\deg R$; this justifies the following assumption.

COROLLARY 6.5. *In the context of theorem 6.3, assume that the number of rows a in a Newton matrix constructed by the incremental algorithm is bounded by a constant multiple of* $\deg R$. *Then, with the binary search in the heuristic range just described, the time complexity of the algorithm for finding this matrix becomes $O^*(c^2 n)$.*

There are two main reasons for constructing Newton matrices. The first is for solving systems of nonlinear polynomial equations. We have examined the phase of matrix construction, which is comparatively costly. Once this is over, certain matrix operations are applied to simplify the linear algebra problem and, eventually, obtain a multiplication table; this is a matrix for which we have to compute eigenvalues and eigenvectors [EC95, Emi97, BMP00]. This is an important question for which we refer the reader to [PC99, PCZ98] on recent results. The method of [BMP00] exploits structure or sparsity in computing selected eigenvalues and eigenvectors of the multiplication table obtained from the Newton matrix as a Schur complement. On the other hand, [MRP00] proposes iterative methods based on structured matrices for computing all real roots and all roots in a given box or disc. Further results exploiting matrix structure are desirable.

The second major application is in computing the exact sparse resultant polynomial, which divides the determinant of the Newton matrix. In this context, the coefficients are typically polynomials in a single variable, denoted u. This may be the same situation as in the u-resultant approach [vdW50] or when u has been chosen among the input variables to be "hidden" in the coefficient field [Emi96]. In both cases, the first question is to compute $\det M(u)$ as a univariate polynomial; the rest of the problem is considered in the next section.

COROLLARY 6.6. *We are given an $a \times c$ Newton matrix M with univariate entries of degree d. Under the hypotheses of theorem 6.3, there exists a Las Vegas algorithm to compute the (univariate) determinant by using $O^*(c^3 nd)$ ops and $O^*(cn + cd)$ storage space.*

PROOF. This can be achieved by the well-known evaluation-interpolation technique. The determinant degree in u is bounded by cd, the number of evaluations is $1 + cd$, and one determinant of the specialized matrix requires $O^*(c^2 n)$ ops. The latter bound follows from theorem 5.6 and the Las Vegas algorithm of [Wie86, thm. 1] as extended in [KP91, lem. 2]; see alternatively [EP96, thm. 3.2] or [EP97, thm. 3.2]. The needed space is $O^*(cn)$ in addition to $O(cd)$ needed for storing the determinant values and interpolating from them to the polynomial coefficients. □

For the u-resultant construction, $d = 1$ and the number of columns containing u

equals the degree of resultant R in the coefficients of the u-polynomial. If the latter is f_{n+1}, then the time complexity becomes $O^*(c^2 n \deg_{f_{n+1}} R)$.

7. Sparse resultant computation

This section focuses on computing the sparse resultant from a set of Newton matrices, when all input coefficients are given specific numeric values. These are either exact or known to some limited precision. Moreover, it is straightforward to extend our algorithms to the case where some polynomial coefficients remain indeterminate or are expressed in terms of parameters, just as at the end of the previous section. Exploiting the matrix structure enables us to decrease the overall complexity by a factor proportional to the square root of matrix size.

We shall require an additional property for the Newton matrices used. Associate each matrix with one of the given polynomials f_i, so that the number of rows of M containing multiples of f_i is precisely $\deg_{f_i} R$, hence the degree of $\det M$ in the coefficients of f_i equals the corresponding degree of the resultant. This property can be guaranteed in the case of the incremental algorithm if we fix set B_i to its initial size [EC95], and is also satisfied in the case of the subdivision-based algorithm of [CE00]. Thus, either algorithm can be used in the discussion that follows.

The naive way to compute R as the Greatest Common Divisor (GCD) of $n+1$ determinants is known not to work for arbitrary coefficient specializations [Zip93]. For this, two probabilistic methods have been proposed by Canny and Emiris; detailed complexity and error analysis can be found in [CE00]. Another source of randomization that we do not explicitly determine is the application of the Las Vegas algorithm of corollary 6.6.

Let M_i be the Newton matrix associated to f_i, for $1 \leq i \leq n+1$. Recall that the f_i have indeterminate coefficients and let g_i be the specialization of f_i and h_i be a random polynomial with the same support. Denote by $D_i^{(j)}$, $0 \leq j \leq n+1$ the determinant of matrix M_i for the system obtained after specializing $f_k \mapsto g_k + \epsilon h_k$, for $k \leq j$ and $f_k \mapsto h_k$, for $k > j$, where ϵ is an indeterminate that will go to zero.

The *division method* determines the resultant of the system $g_i + \epsilon h_i$ (within a scalar factor) by

$$R(g_i + \epsilon h_i) = \frac{D_{n+1}^{(n+1)}}{D_{n+1}^{(n)}} \cdots \frac{D_1^{(1)}}{D_1^{(0)}}.$$

The desired resultant $R(g_i)$ can then be obtained by setting $\epsilon = 0$, provided that the choice of h_i is sufficiently generic. This is equivalent to requiring that all $D_i^{(j)}$, as polynomials in ϵ, have full degree. If the coefficients of each h_i is distributed uniformly with β bits, then the probability that this requirement fails is bounded by $(n+2)^2 \deg R / 2^\beta$ [CE00]. Note that R may be computed by using less than $n+1$ matrix determinants, if at least one of them happens to have the same degree as R in the coefficients of more than one polynomial.

THEOREM 7.1. *Suppose that we have already computed all (and at most $n+1$) necessary Newton matrices for polynomial system $g_1, \ldots, g_{n+1} \in P$, with matrix size $c \times c$. The sparse resultant of the specialized system can be computed by the division method (a Las Vegas algorithm) in $O^*(c^2 n^2 \deg R)$ ops, using $O^*(cn)$ additional storage space, where $\deg R$ denotes the total degree of the sparse resultant in the input coefficients.*

PROOF. The evaluation-interpolation scheme is used with $1 + \deg R$ different values for ϵ, since the degree of $R(g_i + \epsilon h_i)$ in ϵ is bounded by $\deg R$. The dominant complexity is that of evaluating the $2n$ determinants. Since $\deg R \leq c$, the storage for the interpolation phase is in $O^*(cn)$ by setting $d = 1$ in corollary 6.6. □

Observe that only the constant term of $R(g_i + \epsilon h_i)$ is needed. The previous time complexity bound was $O^*(M(c) \deg R)$, where $M(c)$ is the arithmetic complexity of

a $c \times c$ matrix multiplication. Theoretically, $M(c) = O(c^{2.38})$ but in practice it is in $O(c^{2.81})$.

The following method uses only two Newton matrix determinants by distinguishing an exponent vector $a \in A_1$ and imposing a related technical constrain on B_1 (for details, see [CE00]). The first determinant, denoted D_1, is $D_1^{(n+1)}$ under the above notation. The second, denoted D_1', is the determinant of M_1 for specialized system $f_1 \mapsto x_1^a + \epsilon h_1$, $f_i \mapsto g_i + \epsilon h_i$, for $i \geq 2$. Then, the *GCD method* computes

$$R(g_i + \epsilon h_i) = \frac{D_1}{\gcd(D_1, D_1')},$$

and the desired resultant is again obtained by setting $\epsilon = 0$.

THEOREM 7.2. *With the above notation, the sparse resultant of $g_1, \ldots, g_{n+1} \in P$ can be computed by the GCD method in $O^*(c^3 n)$ ops, using $O^*(cn)$ total space.*

PROOF. The dominant step is the computation of D_1, D_1' as univariate polynomials in ϵ, with degree bounded by c. By the evaluation-interpolation scheme, this takes $O^*(c^3 n)$ ops and $O^*(cn)$ storage. Computing the GCD, then evaluating the fraction and, lastly, interpolating to the least significant coefficient of $R(g_i + \epsilon h_i)$, all have dominated complexities. □

The previous time complexity bound was $O^*(M(c)c)$. Note that the univariate GCD computation can be reduced to a branch-free computation of a subresultant because the degree of the GCD, which is precisely the extraneous factor in D_1, is known in advance. Moreover, this computation can be enhanced by probabilistic interpolation techniques [Zip93, ch. 15].

Both the division and the GCD method are readily extended to computing the sparse resultant polynomial, if the coefficients are specialized to functions of one or more parameters. This covers also the case of the u-resultant.

8. Conclusion

Most complexity bounds rely on the efficiency of FFT, i.e., its quasi-linear time complexity and linear space complexity. Yet, it is known that the latter algorithm is truly advantageous only for rather large inputs, due to the relatively high overhead constant hidden in the $O()$ notation. Our methods can be adapted to other basic algorithms for polynomial multiplication of intermediate speed, namely the classical algorithm and the so-called Karatsuba's method [KO63], which may be preferable for inputs of moderate size [Ber01]. Karatsuba's multiplication algorithm has linear space complexity and time complexity $O(k^{\lg 3})$ for k-degree polynomials, where lg denotes the logarithm in base 2. See table 1 for some ramifications.

Our results contribute in the direction of developing numerical nonlinear algebra. Resultant matrices reduce polynomial system solving in the zero-dimensional case to a linear algebra problem, including an eigenvalue/eigenvector computation. We have pointed out recent advances, though further results exploiting matrix structure are desirable.

We may try to combine other ways of exploiting structure and sparsity, in particular since the large number of zero entries usually constitute the great majority. One example is by applying nested dissection [LRT79]. Last but not least, we would like to use information between successive rank tests since every rejected candidate is a submatrix of the next.

Acknowledgements

We thank Erich Kaltofen for pointing out the references on Tellegen's theorem and Henryk Woźniakowski for furnishing us with the reprints of [KW92, KW94].

The first author was partially supported by European ESPRIT project FRISCO (LTR 21.024). The second author was supported by NSF Grants CCR 9020690 and CCR 9625344, and PSC-CUNY Awards Nos. 666327 and 667340. Work on this paper was partially conducted while the second author was on sabbatical at INRIA Sophia-Antipolis.

References

ABB+95 E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 2nd edition, 1995.

Ber01D.J. Bernstein. Multidigit multiplication for mathematicians. *Adv. Appl. Math.*, 2001. To appear.

BCL82 B. Buchberger, G.E. Collins, and R. Loos, editors. *Computer Algebra: Symbolic and Algebraic Computation*. Springer, Wien, 2nd edition, 1982.

BCS97 P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*. Springer, Berlin, 1997.

BMP00 D. Bondyfalat, B. Mourrain, and V.Y. Pan. Computation of a specified root of a polynomial system of equations using eigenvectors. *Lin. Alg. & Appl.* 319:193–209, 2000.

BP94 D. Bini and V.Y. Pan. *Polynomial and Matrix Computations*, volume 1: Fundamental Algorithms. Birkhäuser, Boston, 1994.

CE00 J.F. Canny and I.Z. Emiris. A subdivision-based algorithm for the sparse resultant. *J. ACM*, 47(3):417–451, 2000.

CKL89 J.F. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, pages 121–128, 1989.

CP93 J. Canny and P. Pedersen. An algorithm for the Newton resultant. Technical Report 1394, Comp. Science Dept., Cornell University, 1993.

EC95 I.Z. Emiris and J.F. Canny. Efficient incremental algorithms for the sparse resultant and the mixed volume. *J. Symbolic Computation*, 20(2):117–149, August 1995.

Emi96 I.Z. Emiris. On the complexity of sparse elimination. *J. Complexity*, 12:134–166, 1996.

Emi97 I.Z. Emiris. A general solver based on sparse resultants: Numerical issues and kinematic applications. Technical Report 3110, INRIA Sophia-Antipolis, France, January 1997.

EP96 I.Z. Emiris and V.Y. Pan. Techniques for exploiting structure in matrix formulae of the sparse resultant. *Calcolo, Spec. Issue on Toeplitz Matrices: Structure, Algorithms and Applications*, 33 (3–4):353–369, 1996.

EP97 I.Z. Emiris and V.Y. Pan. The structure of sparse resultant matrices. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, Maui, Hawaii, pages 189–196, July 1997.

Ewa96 G. Ewald. Combinatorial convexity and algebraic geometry. Springer, New York, 1996.

GV96 G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 1996.

KL88 E. Kaltofen and Y. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *Proc. ACM Intern. Symp. on Symbolic and Algebraic Computation*, volume 358 of *Lect. Notes in Comp. Science*, pages 467–474. Springer, 1988.

KO63 A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Dokl.*, 7:595–596, 1963.

KP91 E. Kaltofen and V.Y. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. 3rd Ann. ACM Symp. on Parallel Algorithms and Architectures*, pages 180–191, New York, 1991. ACM Press.

KS91 E. Kaltofen and B.D. Saunders. On Wiedemann's method for solving sparse linear systems. In *Proc. Intern. Symp. Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Lect. Notes in Comp. Science*, volume 536, pages 29–38. Springer, 1991.

KW92 J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the

power and Lanczos algorithms with a random start. *SIAM J. Matrix Anal. Appl.*, 13(4):1094–1122, 1992.

KW94 J. Kuczyński and H. Woźniakowski. Probabilistic bounds on the extremal eigenvalues and condition number by the Lanczos algorithm. *SIAM J. Matrix Anal. Appl.*, 15(2):672–692, 1994.

Laz81 D. Lazard. Résolution des systèmes d'équations algébriques. *Theor. Comp. Science*, 15:77–110, 1981.

LRT79 R.J. Lipton, D. Rose, and R.E. Tarjan, Generalized Nested Dissection, *SIAM J. Numer. Anal.,* 16(2):346–358, 1979.

Man94 D. Manocha. Solving systems of polynomial equations. *IEEE Comp. Graphics and Appl., Special Issue on Solid Modeling*, pages 46–55, 1994.

MP97 B. Mourrain and V.Y. Pan. Solving special polynomial systems by using structured matrices and algebraic residues. In F. Cucker and M. Shub, editors, *Proc. Workshop on Foundations of Computational Mathematics*, pages 287–304. Springer, 1997.

MP98 B. Mourrain and V.Y. Pan. Asymptotic acceleration of solving polynomial systems. In *Proc. ACM Symp. Theory of Comput.*, pages 488–496. ACM Press, New York, 1998.

MP00 B. Mourrain and V.Y. Pan. Multivariate polynomials, duality and structured matrices. *J. Complexity*, 16(1):110–180, 2000.

MRP00 B. Mourrain, V.Y. Pan, and O. Ruatta. Asymptotic acceleration of solving multivariate polynomial systems of equations. Submitted for publication, 2000.

Pan96a V.Y. Pan. Numerical computation of a polynomial GCD and extensions. Technical Report 2969, INRIA, Sophia-Antipolis, France, August 1996.

Pan96b V.Y. Pan. Parallel computation of polynomial GCD and some related parallel computations over abstract fields. *Theor. Comp. Science*, 162(2):173–223, 1996.

Pan97 V.Y. Pan. Faster solution of the key equation for decoding the BCH error-correcting codes. In *Proc. ACM Symp. Theory of Comp.*, pages 168–175. ACM Press, 1997.

Pan01 V.Y. Pan. *Structured matrices and polynomials: Unified superfast algorithms*, Birkhäuser, Boston, 2001.

Par80 B.N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, 1980.

PC99 V.Y. Pan and Z. Chen. The complexity of matrix eigenproblem. In *Proc. ACM Symp. Theory of Comput.*, pages 507–516. ACM Press, New York, 1999.

PCZ98 V.Y. Pan, Z.Q. Chen, and A. Zheng. The complexity of algebraic eigenproblem. Preprint 071, MSRI, Berkeley, 1998.

Stu94 B. Sturmfels. On the Newton polytope of the resultant. *J. of Algebr. Combinatorics*, 3:207–236, 1994.

vdW50 B.L. van der Waerden. *Modern Algebra.* F. Ungar Publishing Co., New York, 3rd edition, 1950.

Wie86 D.H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986.

Zip93 R. Zippel. *Effective Polynomial Computation.* Kluwer Academic Publishers, Boston, 1993.