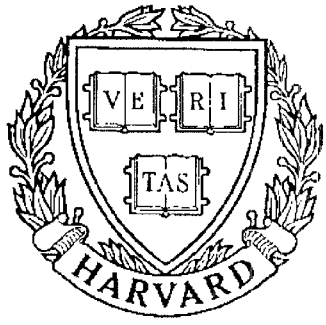
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Fast Error-Free Algorithms for Polynomial Matrix Computations

by J.S. Baras, D.C. MacEnany and R.L. Munach

TR 90-14

FAST ERROR-FREE ALGORITHMS FOR POLYNOMIAL MATRIX COMPUTATIONS

John S. Baras, David C. MacEnany and Robert L. Munach
Systems Research Center
University of Maryland
College Park, MD 20742

Abstract

Polynomial matrices over fields and rings provide a unifying framework for many control system design problems. These include dynamic compensator design, infinite dimensional systems, controllers for nonlinear systems, and even controllers for discrete event systems. An important obstacle for utilizing these powerful mathematical tools in practical applications has been the non-availability of efficient and fast algoritms to carry through the precise error-free computations required by these algebraic methods. Recently, with the advent of computer algebra this has become possible. In this paper we develop highly efficient, error-free algorithms, for most of the important computations needed in linear systems over fields or rings. We show that the structure of the underlying rings and modules is critical in designing such algorithms. We also discuss the importance of such algorithms for controller synthesis.

1. Introduction

The theory of polynomial matrices [8,21,23] plays a key role in the frequency-domain approach to the synthesis of multi-input/multi-output control and communication systems [13,24,25]. Examples include coprime factorizations of transfer function matrices, canonical realizations obtained from matrix fraction descriptions, design of feedback compensators and convolutional coders, and the analysis of quantization effects in linear systems. Typically, such problems abstract in a natural way to the need to solve systems of generalized Diophantine equations, e.g., the so-called Bezout equation [6,15,19,22]. These and other problems involving polynomial matrices require efficient polynomial matrix triangularization procedures [16], a result which is not surprising given the importance of

matrix triangularization techniques in numerical linear algebra. There, matrices with entries from a field can be triangularized using some form of Gaussian elimination. However, polynomial matrices have entries from a polynomial ring, an algebraic object for which Gaussian elimination is not defined. For matrices with entries from a polynomial ring which is Euclidean—the kind encountered most often in control theory applications—triangularization is accomplished instead by what is naturally referred to as Euclidean elimination. Unfortunately, the numerical stability and sensitivity issues of Euclidean elimination are not well understood and in practice floating-point arithmetic has yielded poor results. At present, a reliable numerical algorithm for the triangularization of polynomial matrices does not exist.

This paper presents algorithms for polynomial matrix triangularization which entirely circumvent the numerical sensitivity issues of floating-point methods through the use of exact, symbolic methods from computer algebra [5,14,20]. Often one encounters the comment that since in practical problems the numerical coefficients are not known precisely, computer algebra methods are at a disadvantage from a practical point of view. However this is a misconception. Whether we know the coefficients accurately or not is not the issue. The real issue in using these algebraic methods is to what extend we can perform the required computations within the accuracy of the model data. Existing floating point methods are poor, highly sensitive and often lead to large errors; essentially since they suffer from the same problems as computing zeroes of polynomials. The use of exact, error-free algorithms guarantees that all calculations are accurate to within the precision of the model data—the best that can be achieved. Furthermore one can calculate with such algorithms the exact sensitivities involved and therefore judge appropriately the confidence one should place on the results. Previous computer algebra algorithms for polynomial matrix problems appearing in control systems have been reported in [11]. Their performnce was very slow even on small size problems. We place emphasis on efficient algorithms to compute *exact* Hermite forms of polynomial matrices. The triangular, or more correctly, *trapezoidal* Hermite form is defined for any matrix with entries from a principal ideal ring [21,23]. Such matrices arise in many practical problems in communications and control. Here we shall focus on matrices having entries which are polynomials with rational coef-

ficients. An important aspect of the exact triangularization of such matrices involves the choice of arithmetic. We consider the tradeoffs between rational and integer arithmetic and choose the latter. This choice leads us to consider algorithms for the division of polynomials over a unique factorization domain (UFD). The standard algorithm for this task is well-known [3,4,7,18] and defined more generally for polynomials with coefficients from any *commutative ring with identity.* This algorithm is well-suited to the scalar problem of GCD computation of polynomials over UFDs since it avoids the computation of GCDs of the coefficients. In the context of polynomial matrix triangularization however, it becomes unavoidable to exploit the richer structure of the coefficient ring: the fact that GCDs are defined on a UFD. As a result we present an alternative to the standard algorithm specialized to polynomials over UFDs but enjoying a certain optimality property which is crucial to the efficiency of matrix triangularization procedures.

We have implemented algorithms to compute exact Hermite forms of polynomial matrices in the MACSYMA and *Mathematica* computer algebra languages. We have also written a suite of auxiliary programs which call on these triangularization procedures in order to perform the more high-level tasks arising in the frequency-domain approach to control system synthesis. We conducted simulations with MACSYMA code running on Texas Instruments Explorer II and give performance results for the triangularization of polynomial matrices. The algorithms presented here have time performance which is three orders of magnitude better than algorithms known earlier to control engineers.

2. Facts And Terminology of Polynomials and Polynomial Matrices

In this section we draw upon standard material from modern algebra [10,12]. Denote by $Q[s]$ the ring of polynomials in the *indeterminate* 's' with coefficients drawn from the field of rational numbers, Q. The subring $Z[s]$ of $Q[s]$ results when the polynomial coefficients are restricted to lie in Z, the ring of integers. The *leading coefficient* of a polynomial is the nonzero coefficient of its highest degree term. By convention, the leading coefficient of the zero polynomial is defined to be one, and its degree is taken to be $-\infty$. If the leading coefficient of a polynomial is one, then the polynomial is said to be *monic*. If $a(s)$ in $Z[s]$ is a polynomial of degree zero, then $a(s)$ is said to be a *unit* of $Q[s]$, i.e.,

$(a(s))^{-1}$ is in $Q[s]$. Obviously, the units of $Q[s]$ are precisely the units of Q, namely, $Q \setminus \{0\}$. A polynomial $a(s)$ in $Z[s]$ is called *primitive* if its coefficients are *relatively prime* in Z, i.e., if the greatest common divisor of its coefficients is a unit of Z, namely ± 1. For any $a(s)$ in $Z[s]$, there exists a non-zero scalar c_a in Z, unique up to its sign, and a primitive polynomial $p_a(s)$ in $Z[s]$, such that $a(s) = c_a \cdot p_a(s)$. The pair $(c_a, p_a(s))$ is called a *content-primitive factorization* of $a(s)$; with slight imprecision c_a is called *the* content of $a(s)$ and $p_a(s)$ its primitive (with respect to c_a). A collection of polynomials in $Z[s]$ having contents which are relatively prime is said to be *relatively primitive.* A polynomial $p(s)$ *divides* a polynomial $q(s)$, written $p(s)|q(s)$, if there exists $c(s)$ such that $p(s)\, c(s) = q(s)$. A *common divisor*, CD, $c(s)$ of $\{p_i(s) : i = 1, \ldots, n\}$ is a polynomial such that $c(s)|\{p_i(s)\}$. A *greatest common divisor*, GCD, $g(s)$ of $\{p_i(s)\}$ is a CD of $\{p_i(s)\}$ such that $c(s)|g(s)$ for any other CD, $c(s)$, of $\{p_i(s)\}$.

Denote by $M[Q[s]]$ the collection of $m \times n$ matrices with entries from $Q[s]$. Similarly, $M[Z[s]]$ will denote the subset of $M[Q[s]]$ when the entries are resticted to lie in $Z[s]$. We call $A(s)$ in $M[Q[s]]$ a *polynomial matrix*. Letting R denote either $Z[s]$ or $Q[s]$, $A(s)$ in $M[R]$ is said to be nonsingular if $A(s)$ is square ($m = n$) and $\det A(s)$ is not the zero polynomial. A nonsingular polynomial matrix, $U(s)$ in $M[R]$ having a determinant which is a unit of R is a *unit* of $M[R]$ and is said to be *unimodular* (with respect to $M[R]$). In this case therefore, $U(s)^{-1} = Adj\ U(s)/\det U(s)$ is itself a polynomial matrix and also unimodular with respect to $M[R]$. When the ring R also happens to be a field, the concepts of unimodularity and nonsingularity conincide, but keep in mind that in the general ring case it is unimodularity that coincides with the usual notion of invertibility.

A row of a polynomial matrix $A(s)$ in $M[Z[s]]$ is said to be *primitive* if its polynomial entries are relatively primitive; $A(s)$ is said to be *row (left) primitive*, if every row is primitive. For any $A(s)$ in $M[Z[s]]$, there exists a diagonal matrix C_A in $M[Z]$ and a row primitive matrix $P_A(s)$ in $M[Z[s]]$ such that $A(s) = C_A \cdot P_A(s)$. This we call a *left content-primitive factorization* of $A(s)$. The diagonal elements of C_A are called the *row contents* of the respective rows of $A(s)$. By analogy with the scalar case, a content-primitive factorization is obviously unique only up to the choice of the signs of the row contents.

Two polynomial matrices $A_1(s)$ and $A_2(s)$ in $M[Q[s]]$ (or polynomial vectors if $m = 1$

or $n = 1$) are said to be *linearly dependent* if there exists a coefficient of proportionality $a(s)/b(s)$ in the field of rational functions such that $A_1(s) = a(s)/b(s)A_2(s)$. Hence $A_1(s)$ and $A_2(s)$ are linearly dependent if there exist $a(s)$ and $b(s)$ in $Q[s]$ such that $b(s)A_1(s) - a(s)A_2(s) = 0$. This leads to the simple generalization that a finite collection of polynomial matrices is linearly dependent if and only if there exists a nontrivial linear combination of them employing only polynomial coefficients which equals the zero polynomial matrix. With this in mind, elementary row and column operations for polynomial matrices in $M[Q[s]]$ take the form:

1. interchange of any two rows;
2. addition to any row a polynomial multiple of any other row;
3. multiplication of a row by a nonzero rational.

Performing any of the above operations on an identity matrix results in an *elementary matrix* $E(s)$ in $M[Q[s]]$. Clearly, each such $E(s)$ is unimodular as is its inverse and also the product of any number of elementary matrices. Conversely, every unimodular matrix is a product of elementary matrices. Two polynomial matrices $A(s)$ and $B(s)$ are said to be *row equivalent* if each can be obtained from the other using a finite sequence of elementary row operations. In other words, they are related by left multiplication with a unimodular matrix $U(s)$ such that $B(s) = U(s)\ A(s)$ and $A(s) = U^{-1}(s)\ B(s)$.

Every $m \times n$ polynomial matrix $A(s)$ in $M[Q[s]]$ is row equivalent to an upper triangular form (or upper trapezoidal form if $m \neq n$). Therefore, it can be reduced by a sequence of elementary row operations to an upper triangular (trapezoidal) matrix $H_A(s)$ in $M[Q[s]]$. In fact, there exists a unimodular matrix $U(s)$ such that $U(s)\ A(s) = H_A(s)$ with $H_A(s)$ satisfying the following conditions:

1. Each entry below the diagonal is identically zero;
2. Each nonzero diagonal entry has degree greater than the entries above it;
3. Each diagonal entry is monic.

We say that $H_A(s)$ is a *column monic-Hermite form* of $A(s)$. To state conditions more precise than these concerning the structure of $H_A(s)$ requires the notion of the *rank* of a polynomial matrix which we omit. This omission is forgiveable in that rank should not enter into an exact triangularization algorithm. Indeed, exact rank information for

polynomial matrices is often obtained most efficiently through triangularization.

A *column integral-Hermite form* can be defined in terms of the column monic-Hermite form. Letting $H_A(s)$ denote a column monic-Hermite form for $A(s)$ in $M[Q[s]]$, multiply each row of $H_A(s)$ with the respectively smallest positive integer such that the matrix $H'_A(s)$ so obtained is in $M[Z[s]]$. Clearly, $H'_A(s)$ is row primitive and row equivalent to $A(s)$. Conversely, suppose that one is given $H'_A(s)$ satisfying conditions (1) and (2) above which is row primitive and row equivalent to $A(s)$. Divide each row of $H'_A(s)$ by the leading coefficient of the polynomial on the diagonal of the respective row and call the matrix so obtained $H_A(s)$. Then clearly there exists $U(s)$ unimodular such that $U(s)A(s) = H_A(s)$ and $H_A(s)$ is a monic-Hermite form of $A(s)$. This concept of column integral-Hermite form gives a triangular form in $M[Z[s]]$ for each matrix in $M[Q[s]]$. If $A(s)$ is nonsingular then it can be shown that its monic-Hermite form is unique and therefore its integral-Hermite form is also unique.

3. Triangularizing Polynomial Matrices

The upper triangularization of matrices with entries from a field using a sequence of *non-singular* elementary row operations plays a key role in the application of the theory of *vector spaces*. Likewise, the upper triangularization of matrices with entries from a ring using a sequence of *unimodular* elementary row operations plays a key role in the application of the theory of *vector modules*. Recall that the axioms defining a vector module (or *matrix module* or simply *module*) over a ring R are the same as those defining a linear vector space except that linear combinations employ scalar coefficients from a general ring R without the restriction that R also be a field. Thus a vector space over a field F, $V(F)$, is precisely the matrix module over the ring F, $M(F)$. We only deal with matrix modules over commutative rings (with identity) so there is no need to distinguish module handedness. We employ the notation $M[A](R)$ to denote the $m \times n$ matrix module over the ring R with matrix entries from the additive abelian group A; for the case $M[R](R)$ we have been writing $M[R]$. Using this notation we point out that $M[Q](Z)$ is a module but that $M[Z](Q)$ is not. We also warn the reader that $M[Z[s]]$ is not a submodule of $M[Q[s]]$; a module $M[B](R)$ is said to be a *submodule* of a module $M[A](R)$ if B is an

additive subgroup of A and for each b in B, rb is in B for all r in R. Thus, $M[Z[s]]$ is a submodule of $M[Q[s]](Z[s])$.

Computing triangular (trapezoidal) forms of matrices can be accomplished on any matrix module of the form $M[R]$ where R is an *integral domain*, i.e., a commutative ring with identity having no zero divisors [1,2,17]. However, this cannot in general be accomplished using only unimodular (i.e., invertible) operations. Nevertheless, the transformation to an upper triangular form using unimodular elementary row operations can be performed quite straightforwardly—in theory at least—on any matrix with entries from a type of integral domain called a Euclidean ring, for instance on a matrix from $M[Q[s]]$. The key feature that Euclidean rings enjoy is the Euclidean division property which we state for $Q[s]$. Given polynomials $a(s), b(s)$ in $Q[s]$ there exist two unique polynomials, the *quotient* $q(s)$ and the *remainder* $r(s)$, such that $b(s) = q(s)\ a(s) + r(s)$ *and* $\deg r(s) < \deg a(s)$. Often this property is stated under the assumption that $\deg a(s) \leq \deg b(s)$ since the proof for the case $\deg b(s) < \deg a(s)$ is trivial: choose $r(s) = a(s)$ and $q(s) \equiv 0$. The Euclidean division lemma for a general Euclidean ring R has a statement which differs from the above only in that polynomial degree, the *ring valuation* for $Q[s]$, is replaced by the ring valuation for R; for $R = Z$ (the case considered by Euclid) the ring valuation is integer absolute value. The fact that the inequality on the degrees of $a(s)$ and $r(s)$ is *strict* allows one to introduce a zero into a polynomial matrix using elementary operations. As an example, consider an attempt to introduce a zero into the $(2,1)$ position of a 2×2 polynomial matrix. Suppose one has found $q(s)$ and $r(s)$ as in the division lemma such that $b(s) = q(s)\ a(s) + r(s)$ and then computes,

$$\begin{pmatrix} 1 & 0 \\ -q(s) & 1 \end{pmatrix} \begin{pmatrix} a(s) & c(s) \\ b(s) & d(s) \end{pmatrix} = \begin{pmatrix} a(s) & c(s) \\ r(s) & d(s) - q(s)c(s) \end{pmatrix}$$

using an obviously unimodular pre-multiplication. Notice that the degree of the $(2,1)$ entry has been decreased by at least one. By interchanging the rows, the same procedure can now be repeated on the resulting matrix and iterated, each step reducing the degree of the $(2,1)$ entry in view of the strict degree inequality. This process consists entirely of unimodular premultiplications and clearly cannot continue indefinitely without yielding a

constant in the $(2,1)$ entry. If the constant is not zero, one final row exchange and the obvious elementary row operation will introduce a zero into the $(2,1)$ position. Moreover, one can show that the resulting polynomial in the $(1,1)$ position will be the GCD of the original polynomials $a(s)$ and $b(s)$. The use of this process to introduce zeroes into polynomial matrices we call *Euclidean elimination* by analogy with Gaussian elimination.

4. Integer vs Rational Arithmetic

The above example also serves to illustrate the fact that rational arithmetic is costly. For instance, to calculate the coefficients of polynomials of the form $d(s) - q(s)\ c(s)$ with c, d, q in $Q[s]$, one encounters the generic computation $\alpha + \beta\,\gamma$ with α, β, γ in Q. If these rationals are expressed as ratios of integers $\alpha = \frac{N^\alpha}{D^\alpha}$, $\beta = \frac{N^\beta}{D^\beta}$, $\gamma = \frac{N^\gamma}{D^\gamma}$, all reduced to lowest terms, then

$$\alpha + \gamma\ \delta = \frac{N^\alpha\ D^\beta\ D^\gamma + N^\beta\ N^\gamma\ D^\alpha}{D^\alpha\ D^\beta\ D^\gamma}.$$

This computation requires six integer multiplications, one integer addition and the calculation of a GCD. Although there are more efficient methods [17], it remains a fact that rational arithmetic is computationally expensive, due in large part to the need for GCD calculations. On the other hand, if it can be arranged so that α, β and γ are all integers, then the same computation obviously requires only two integer multiplications, one integer addition and no GCD calculation. Thus, our goal is to carry out matrix triangularization on $M[Q[s]]$ using only integer arithmetic. Clearly, by multiplying each row of any $A(s)$ in $M[Q[s]]$ by a large enough integer, the denominators of every coefficient of every entry of $A(s)$ can be cancelled and such a diagonal operation is certainly unimodular in $M[Q[s]]$. Again, this computation can be arranged more efficiently but because it involves a fixed overhead, assume for convenience that $A(s)$ is *given* in $M[Z[s]]$.

Unfortunately, this creates new difficulties because Euclidean elimination is not defined for $M[Z[s]]$ since $Z[s]$ is not a Euclidean ring. For instance, it is easy to see that the remainder of two polynomials in $Q[s]$ with integer coefficients has, in general, rational coefficients; consider the remainder of $2s$ after division by $3s-1$. In other words, Euclidean division is not defined for $Z[s]$. However, $Z[s]$ is an instance of a polynomial ring with

coefficients from a commutative ring with identity and for such a ring one has the *pseudo-division lemma*, a natural generalization of the Euclidean division lemma. Let C denote a commutative ring with identity. Given $a(s)$ and $b(s)$ in $C[s]$ with $\deg a(s) \leq \deg b(s)$ there exist two polynomials, the *pseudo-quotient* $q(s)$ and the *pseudo-remainder* $r(s)$, such that $L\,b(s) = q(s)\,a(s) + r(s)$ *and* $\deg r(s) < \deg a(s)$ where the *premultiplier* $L = a_0^{\deg b - \deg a + 1}$ with a_0 denoting the leading coefficient of $a(s)$. The pseudo-quotient and pseudo-remainder are unique if C is also an integral domain. The proof of the pseudo-division lemma yields a division procedure called *pseudo-division* which like Euclidean division enjoys the all-important *strict* degree reduction property; see [18] for the standard pseudo-division algorithm.

Let's consider an example in which we wish to pseudo-divide $b(s)$ by $a(s)$ where,

$$b(s) = s^8 + s^6 - 3s^4 - 3s^3 + 8s^2 + 2s - 5$$

and

$$a(s) = 3s^6 + 5s^4 - 4s^2 - 9s + 21.$$

Applying the standard pseudo-division algorithm one obtains,

$$27\,b(s) = (9s^2 - 6)\,a(s) + (-15s^4 + 3s^2 - 9),$$

i.e., $L = 3^{8-6+1} = 27$, $q(s) = 9s^2 - 6$ and $r(s) = -15s^4 + 3s^2 - 9$. This example appears in [18] as one step in the task of computing the GCD of $b(s)$ and $a(s)$. The next step is to divide out the content of $r(s)$ and then compute the GCD of $a(s)$ and $p_r(s)$ exploiting the fact that $\gcd(b(s), a(s)) = \gcd(a(s), p_r(s))$. The purpose of this content removal is to keep the size of the coefficients small for purposes of efficiency in succeeding calculations. However, consider the above computation in the context of a matrix triangularization—our 2×2 example will suffice:

$$\begin{pmatrix} 1 & 0 \\ -q(s) & L \end{pmatrix} \begin{pmatrix} a(s) & c(s) \\ b(s) & d(s) \end{pmatrix} = \begin{pmatrix} a(s) & c(s) \\ r(s) & L\,d(s) - q(s)c(s) \end{pmatrix}.$$

In this situation we see that we are not at liberty to blindly divide the entire second row by the content of $r(s)$ (or any integer for that matter) because it may introduce rational

coefficients in the $(2,2)$ entry and thereby ruin our attempt to maintain integer arithmetic. However, note that another solution to the above pseudo-division example is,

$$9\,b(s) = (3s^2 - 2)\,a(s) + (-5s^4 + s^2 - 3),$$

i.e., $L = 27$ is not necessarily the smallest premultiplier for which a "pseudo-quotient" and "pseudo-remainder" exist. Obviously, in the matrix case, "$L = 9$" yields better results than $L = 27$ since it yields smaller coefficients in the second row. Of course in this example the difference is negligible, however, if the size of the leading coefficient of $a(s)$ is large, the difference in computational burden can be quite substantial. Moreover, as we shall see below, keeping the size of all coefficients as small as possible is a primary goal.

5. Pseudo-division for Polynomials over a UFD

It is apparent that there are smaller (and larger) premultipliers, L, than the one defined in the pseudo-division lemma. Now the pseudo-division lemma is the best that one can do in general for polynomials over a commutative ring with identity. But be aware that the concept of 'smaller' referred to in the pseudo-division example is inherited from the fact that Z is also a *unique factorization domain* (UFD). Recall, a UFD is a commutative ring with identity which has no *zero divisors* and admits *prime factorizations*. Let U denote a UFD. One can think of u in U as being "smaller" than u' in U if u is a divisor of u'. For the problem of pseudo-division of polynomials $a(s)$, $b(s)$ in $U[s]$, what we seek is the smallest premultiplier L_* in U such that if there exist L in U and q, r in $U[s]$ satisfying,

$$L\,b(s) = q(s)\,a(s) + r(s) \qquad \text{and} \qquad \deg r(s) < \deg a(s),$$

then L_* divides L *and* $q_*(s)$, $r_*(s)$ in $U[s]$ exist such that,

$$L_*\,b(s) = q_*(s)\,a(s) + r_*(s) \qquad \text{and} \qquad \deg r_*(s) < \deg a(s).$$

A representation for L_* is easy to obtain. Let U' denote the *field of quotients* of U and consider U as embedded in U'. The field of quotients of a ring is defined for any ring which is an integral domain and thus is defined for any UFD; for example Q is the field

of quotients of Z. Consider $a(s)$, $b(s)$ in $U[s]$ and therefore in $U'[s]$ in the sense of the embedding. It is a fact that $U'[s]$ is a Euclidean ring since U' is a field and polynomial rings over fields are Euclidean. Therefore by the Euclidean division lemma there exist $\varrho(s)$, $\rho(s)$ in $U'[s]$ such that,

$$b(s) = \varrho(s)\, a(s) + \rho(s) \qquad \text{and} \qquad \deg \rho(s) < \deg a(s).$$

Now write $\varrho(s) = n(s)/D$ for $n(s)$ in $U[s]$ and D in U with D and the content of $n(s)$ relatively prime in U. Then,

$$D\, b(s) = n(s)\, a(s) + D\, \rho(s) \qquad \text{and} \qquad \deg D\, \rho(s) < \deg a(s).$$

Clearly, $D\, \rho(s)$ is in $U[s]$ so that one must conclude that L_*, $q_*(s)$, and $r_*(s)$ repectively differ from D, $n(s)$ and $D\, \rho(s)$ by at most a unit of U. Thus, we may choose $L_* = D$ and this suggests one method to compute L_* and then by uniqueness $q_*(s) = n(s)$ and $r_*(s) = D\, \rho(s)$. Letting $L = a_0^{\deg b - \deg a + 1}$ (assuming $\deg a(s) \leq \deg b(s)$) and making an appeal to the pseudo-division lemma it is not hard to show that $L_* = L/\gcd(L, c_q)$ where c_q denotes the content of the pseudo-quotient for the premultiplier L. Finally note that,

$$\gcd(L, c_q) = \gcd(\gcd(a_0, q_0), \gcd(a_0, q_1), \ldots, \gcd(a_0, q_{n-m})).$$

Although this gives us a representation for L_*, it does not lead to the most efficient computation for L_* due in large part to the fact that the coefficients of $q(s)$ are already too "big". Thus what we want is an efficient algorithm to compute L_*, $q_*(s)$ and $r_*(s)$. The algorithm given next computes the L_*, $q_*(s)$ and $r_*(s)$ as discussed above. It is a distinct improvement over the pseudo-division lemma given in [18] in that it computes with smaller numbers. It does so by exploiting the richer structure of polynomial rings with coefficients from a UFD but at the cost of both generality and GCD calculations. However, in the matrix problems we consider this cost is unavoidable.

Algorithm M - Pseudo-division of Polynomials over a UFD

Given two nonzero polynomials $b(s) = b_0 s^n + b_1 s^{n-1} + \cdots + b_n$ and $a(s) = a_0 s^m + a_1 s^{m-1} + \ldots + a_m$ in $U[s]$ with $m \leq n$, this algorithm computes the *smallest* L_*, pseudo-quotient $q_*(s)$, and pseudo-remainder $r_*(s)$ as discussed above. It computes L_*, $q_*(s)$, and

$r_*(s)$ *directly* by computing GCD's *on the fly*. This involves "smaller" numbers than first using the pseudo-division algorithm and then computing GCDs as outlined above. Bigger numbers cost more in GCD calculations and given the size of the integers encountered in polynomial matrix computations, e.g., easily greater than 1000 digits, this algorithm can save a substantial amount of time. For simpler notation we drop the 'asterisk' subscript in the algorithm's definition.

BEGIN:

$mnm \leftarrow \min(m, n-m)$

$g \leftarrow GCD(b_0, a_0)$

$l \leftarrow a_0/g$

$L \leftarrow l$

$b_0 \leftarrow b_0/g$

For $i = 1$ thru $n-m$ Do

For $j = i$ thru $n-m$ Do

$b_j \leftarrow b_j * l$

EndDo

For $j = 1$ thru $\min(mnm, n-m-i+1)$ Do

$b_{j+i-1} \leftarrow b_{j+i-1} - a_j * b_i$

EndDo

$g \leftarrow GCD(b_i, a_0)$

$l \leftarrow a_0/g$

$L \leftarrow L * d$

$b_i \leftarrow b_i/g$

EndDo

END

The algorithm terminates with the first $n-m+1$ coefficients of $b(s)$ overwritten according to $\{b_0, b_1, \ldots, b_{n-m}\} \leftarrow \{q_0, q_1, \ldots, q_{n-m}\}$ and the remaining coefficients over written according to $\{b_{n-m+1}, \ldots, b_n\} \leftarrow \{r_0, \ldots, r_{m-1}\}$.

Algorithm M - Proof of Correctness

The informal language description of *Algorithm M* basically implements the following recursion for $k = 0, 1, \ldots, n-m$,

$$\begin{aligned} b^{(-1)}(s) &= b(s); \\ g_k &= \gcd(a_0, b_0^{(k-1)}); \\ l_k &= a_0/g_k; \\ p_k &= b_0^{(k-1)}/g_k; \\ b^{(k)}(s) &= l_k\, b^{(k-1)}(s) - p_k\, a(s). \end{aligned}$$

Observe that $\deg b^{(k)}(s) < \deg b^{(k-1)}(s)$ for $k = 0, 1, \ldots, n-m$ because $l_k\, b_0^{(k-1)} = p_k\, a_0$ (where $b_0^{(k)}$ of course denotes the leading coefficient of $b^{(k)}$). Hence, $\deg b^{(n-m)}(s) < \deg a(s)$. From the algorithm's definition we see that $r_*(s) = b^{(n-m)}(s)$ and $L_* = \prod_{k=0}^{n-m} l_k$. Solving the recursion above we obtain

$$b^{(n-m)}(s) = L_* b(s) - (p_0 l_1 \cdots l_{n-m} s^{n-m} + \cdots + p_{n-m-1} l_{n-m} s + p_{n-m})\, a(s).$$

The algorithm therefore yields,

$$q_*(s) = \sum_{k=0}^{n-m} \left(\frac{L_*}{\prod_{i=0}^{k} l_i} \right) p_k\, s^{n-m-k},$$

whence,

$$L_*\, b(s) = q_*(s)\, a(s) + r_*(s) \qquad \text{and} \qquad \deg r_*(s) < \deg a(s),$$

with L_* in U and $q_*(s)$, $r_*(s)$ in $U[s]$. Thus the algorithm indeed computes a valid solution; next we show that it is optimal. Suppose there exist another L in U and $q(s)$, $r(s)$ in $U[s]$ such that,

$$L\, b(s) = q(s)\, a(s) + r(s) \qquad \text{and} \qquad \deg r(s) < \deg a(s).$$

Then by commutativity $L_*\, L\, b(s) = L\, L_*\, b(s)$ implies,

$$(L\, q_*(s) - L_*\, q(s))\, a(s) = L_*\, r(s) - L\, r_*(s).$$

Since there are no divisors of zero in a UFD this gives,

$$\deg(L\,q_*(s) - L_*\,q(s)) + \deg a(s) = \deg(L_*\,r(s) - L\,r_*(s)).$$

Since $\deg(L_*\,r(s) - L\,r_*(s)) \leq \max\{\deg r(s), \deg r_*(s)\} < \deg a(s)$ it must be true that,

$$\deg(L\,q_*(s) - L_*\,q(s)) = -\infty,$$

and therefore $L_*\,q(s) = L\,q_*(s)$. By equating coefficients we obtain,

$$L\left(\frac{L_*}{\prod_{i=0}^{k} l_i}\right) p_k = L_*\,q_k \qquad k = 0, 1, \ldots, n-m,$$

so that,

$$q_k \prod_{i=0}^{k} l_i = p_k\,L \qquad k = 0, 1, \ldots, n-m.$$

For $k = 0$ we get $l_0 q_0 = L p_0$ and therefore $l_0 | L p_0$. However, from the definition of the algorithm l_0 and p_0 are relatively prime in U, or *coprime*, and so in fact $l_0 | L$. For $k = 1$ we get $l_0 l_1 q_1 = L p_1$ and therefore $l_1 | (\frac{L}{l_0}) p_1$. Again, by construction l_1 and p_1 are coprime and so $l_1 | \frac{L}{l_0}$. In general we have l_k and p_k coprime and $l_k q_k = (\frac{L}{l_0 \cdots l_{k-1}}) p_k$ so that for $k = n - m$ we obtain,

$$l_{n-m} | \frac{L}{l_0 \cdots l_{n-m-1}}.$$

As a result $l_0 \cdots l_{n-m} | L$ and therefore $L_* | L$. **QED**

6. Pseudo-Euclidean Elimination

The introducton of a zero below the diagonal of a matrix $A(s)$ in $M[Z[s]]$ can now be performed using *Algorithm M*. This procedure we shall call *pseudo-Euclidean elimination* (PSEE) for obvious reasons. Consider triangularizing the matrix:

$$A(s) = \begin{pmatrix} 1 & s & s \\ 45s & -10s - 10 & 3s^2 + s + 10 \\ 7 - 5s & 6s^2 - 1 & 4s^2 - 10 \end{pmatrix}.$$

Pseudo-Euclidean elimination yields,

$$\begin{pmatrix} 7577325 & 0 & -1351755s^3 + 1373940s^2 - 5102550s - 7152750 \\ 0 & 89145 & -43605s^3 + 77103s^2 - 234189s - 35190 \\ 0 & 0 & 3706425s^4 - 5202000s^3 + 18532125s^2 + 15671025s + 7152750 \end{pmatrix}$$

and this illustrates the main disadvantage of triangularization on $M[Q[s]]$ performed over $M[Z[s]]$—the coefficient growth of the polynomials. As the number of rows and columns in the matrix increases, this coefficient growth continues unabated and begins to erode the advantage of using integer arithmetic. One approach to handle this new source of coefficient growth is to remove the content of the current row after each pseudo-Euclidean division step. It is better to remove the row content as soon as possible in this way rather than waiting due to the cost of computing GCDs of large integers, nevertheless, we illustrate row content removal for the current example. Factoring the above matrix into a left content-primitive form $C_A\ H'_A(s)$ yields,

$$\begin{pmatrix} 765 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 65025 \end{pmatrix} \begin{pmatrix} 9905 & 0 & -1767s^3 + 1796s^2 - 6670s - 9350 \\ 0 & 9905 & -4845s^3 + 8567s^2 - 26021s - 3910 \\ 0 & 0 & 57s^4 - 80s^3 + 285s^2 + 241s + 110 \end{pmatrix}.$$

The superfluous left content of this matrix can then be discarded since this is equivalent to multiplying $A(s)$ by C_A^{-1} thereby keeping the coeffcient size to a minimum. We emphasize that C_A is unimodular with respect to $M[Q[s]]$ but not with respect to $M[Z[s]]$. We stress that up to the signs of the entries across the rows $H'_A(s)$ is the same matrix which would have resulted had we employed row content removal after each pseudo-Euclidean division step and that this is the more efficient strategy. Note that the above polynomial matrix $H'_A(s)$ is nonsingular and in column integral-Hermite form and that therefore the unique

column monic-Hermite form of $A(s)$ is obtained directly from $H'_A(s)$ as,

$$H_A(s) = \begin{pmatrix} 1 & 0 & -\frac{1767s^3}{9905} + \frac{1796s^2}{9905} - \frac{1334s}{1981} - \frac{1870}{1981} \\ 0 & 1 & -\frac{969s^3}{1981} + \frac{8567s^2}{9905} - \frac{26021s}{9905} - \frac{782}{1981} \\ 0 & 0 & s^4 - \frac{80s^3}{57} + 5s^2 + \frac{241s}{57} + \frac{110}{57} \end{pmatrix}.$$

We see that PSEE provides an efficient triangularization procedure for $M[Z[s]]$ but, strictly speaking, PSEE modified with content factorization is *not* a valid triangularization procedure for $M[Z[s]]$ because content removal is not a unimodular operation in $M[Z[s]]$. On the other hand, augmenting PSEE with content factorization *is* a unimodular operation for $M[Q[s]]$ and yields an efficient triangularization procedure for $M[Q[s]]$ by avoiding rational arithmetic while maintaining integers of the smallest possible magnitude throughout an elimination. An analogous remark can be made about any pair of modules of the form $M[U[s]]$ and $M[U'[s]]$ as discussed above.

7. Algorithms for Triangularizing Polynomial Matrices

Algorithm T – Column-Oriented Triangularization of Polynomial Matrices

Given an $N \times N$ nonsingular matrix $A \in M[Z[s]]$, this algorithm overwrites A with a triangular form obtained by a sequence of unimodular, elementary row operations. It avoids rational arithmetic by using pseudo-division as defined in *Algorithm M* in order to achieve maximum computational efficiency with minimum coefficient growth. In addition, it further inhibits coefficient growth by factoring out the row content after each pseudo-Euclidean division step. This algorithm operates in a column oriented fashion by successively zeroing out the entries in each column below the diagonal. This is shown pictorially below.

$$\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix} \rightarrow \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix} \cdots \rightarrow \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{pmatrix}.$$

Assume there exists a pre-defined function

$$MinimumDegreeIndex(A, k) := \arg\min\{\deg A_{1,k}, \ldots, \deg A_{N,k}\}$$

which returns the index of the row of A whose k^{th} entry is a non-zero polynomial of lowest degree among the rows $\{k, k+1, \ldots, N\}$. If $A_{k,k}(s) = A_{k+1,k}(s) = A_{N,k}(s) \equiv 0$, then it returns $-\infty$, the degree of the zero polynomial by our convention.

BEGIN:

For k = 1 thru N-1 Do

$index \leftarrow MinimumDegreeIndex(A, k)$

If $index \neq -\infty$ Then

$A_{k,\cdot} \leftrightarrow A_{index,\cdot}$ (exchange rows k and $index$)

For $n = k+1$ thru N Do (zero out all entries in column k below $A_{k,k}$)

EndlessLoop

$num \leftarrow pseudo - quotient(A_{n,k}, A_{k,k})$

$denom \leftarrow pseudo - remainder(A_{n,k}, A_{k,k})$

$A_{n,.} \leftarrow denom * A_{n,.} - num * A_{k,.}$

$A_{n,.} \leftarrow A_{n,.}/GCD\{content(A_{n,1}), \ldots, content(A_{n,N})\}$

If $A_{n,k} \equiv 0$ then exit EndlessLoop

$A_{n,.} \leftrightarrow A_{k,.}$

End EndlessLoop

EndDo

EndIf

EndDo

End

It is emphasized that this is a 'paper' algorithm. In fact, the working code based on the above is more efficient and can handle singular (we have not defined the notion of *rank* of a polynomial matrix here) and non-square matrices and the entries can initially belong to $Q[s]$. However, these programming considerations just complicate matters and obscure the basic operation.

Algorithm P – Principal Minor-Oriented Triangularization of Polynomial Matrices

This algorithm is similar to the one above except it performs the zeroing process in a leading principal minor oriented fashion so that the algorithm consists of $N-1$ stages where the $k \times k$ leading principal submatrix is in a triangular form by the end of the k^{th} stage. Furthermore, the algorithm employs an additional substage which reduces the degrees of the polynomial entries above the diagonal *on the fly* using pseudo-division as in *Algorithm M*. The order in which the degrees are reduced is important and is based upon notions from [16] for triangularizing matrices in $M[Z]$. The order is shown pictorially below:

$$\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix} \rightarrow \begin{pmatrix} x & 1 & x & x & x \\ 0 & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix} \rightarrow \begin{pmatrix} x & 2 & 3 & x & x \\ 0 & x & 1 & x & x \\ 0 & 0 & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} x & 4 & 5 & 6 & x \\ 0 & x & 2 & 3 & x \\ 0 & 0 & x & 1 & x \\ 0 & 0 & 0 & x & x \\ x & x & x & x & x \end{pmatrix} \cdots \longrightarrow \begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{pmatrix}$$

The output matrix is in column integral-Hermite form, not simply triangularized as in *Algorithm T*, but with the entries above the diagonal of degree less than the diagonal entry. Clearly, the monic column Hermite form is easily obtained by left multiplication with the appropriate diagonal matrix of rational numbers, a unimodular matrix with respect to $M[Q[s]]$.

BEGIN:

For $k = 2$ thru N Do

For $n = 1$ thru $k-1$ Do (triangularize the $k \times k^{th}$ leading principal minor)

If $\deg A_{n,n} > \deg A_{k,n}$ Then $A_{k,.} \leftrightarrow A_{n,.}$

EndlessLoop

$num \leftarrow pseudoquotient(A_{k,n}, A_{n,n})$

$denom \leftarrow pseudoremainder(A_{k,n}, A_{n,n})$

$A_{k,.} \leftarrow denom * A_{k,.} - num * A_{n,.}$

$A_{k,.} \leftarrow A_{k,.}/GCD\{content(A_{k,1}), \ldots, content(A_{k,N})\}$

If $A_{k,n} \neq 0$ then $A_{k,.} \leftrightarrow A_{n,.}$ else Exit EndlessLoop

End EndlessLoop

End Do

For $i = -1$ thru $-k+1$ step -1 Do (reduce degs of abv diag polys in $k \times k^{th}$ minor)

For $j = i+1$ thru 0 Do

If $\deg A_{k+i,k+j} \geq \deg A_{k+j,k+j}$ Then

$num \leftarrow pseudoquotient(A_{k+i,k+j}, A_{kk+j,kk+j})$

$denom \leftarrow pseudoremainder(A_{k+i,k+j}, A_{kk+j,kk+j})$

$A_{k+i,.} \leftarrow denom * A_{k+i,.} - num * A_{k+j,.}$

$A_{k+i,.} \leftarrow A_{k+i,.}/GCD\{content(A_{k+i,1}), \ldots, content(A_{k+i,N})\}$

EndIf

EndDo

EndDo

EndDo

End

We close this section by noting that in both *Algorithm T* and *Algorithm P* each pseudo-Euclidean division step affects the entire row *and* the row content is removed after each division step . Alternatively, one could solve a scalar Bezout identity for each zero to be introduced using pseudo-division techniques and then perform a single elementary row operation followed by a the row content is removedrow content removal. However, the single row content of the latter method will be much larger than any of the "elementary" row contents computed by *Algorithm T* or *Algorithm P*. This makes the alternative method much less attractive than at first glance in light of the fact that computing the many "small" row contents is more efficient than computing the single "large" row content.

8. Simulation Results

Simulations were performed to determine the average time required to triangularize a square polynomial matrix and the maximum coefficient length of the output matrix using both *Algorithm T* and *Algorithm P* (see attached graphs). The maximum coefficient length is the number of digits of the largest (in absolute value) coefficient appearing in any polynomial entry of the output matrix. Each matrix had polynomial entries with randomly generated integer coefficients uniform on $[-99, 99]$. Runs were parameterized by the dimension of the matrix, which ranged from 2 to 16 and the maximum degree of its polynomial entries, chosen uniformly on $[0, \text{degreemax}]$, as degreemax ranged from 1 to 6.

These simulations were conducted on a Texas Intruments Explorer II with 16 mb of physical memory and 128 mb of virtual memory running at 40 MHz using the MACSYMA version of our algorithms. The graphs represent the results of the simulations averaged over 5 runs. The results indicate that *Algorithm T* was moderately faster than *Algorithm P* in triangularizing matrices up to 9×9. At that point *Algorithm T* was still faster for triangularizing matrices with lower degree polynomials, but slower in the higher degree

polynomials. This can be attributed to the fact that *Algorithm P* requires less memory during computations due to its substage which reduces the degrees of the polynomials above the diagonal on the fly. Therefore costly garbage collections, a technique of freeing dynamically allocated memory, are reduced.

It appears that both of these algorithms run close to exponential time. The slopes of the semi-log plots of the timings increase slightly with increasing polynomial degree. The maximum coefficient length was approximately the same for each algorithm and the coefficient growth appears to be sub-exponential with increasing matrix dimension. A 16×16 matrix with degree 6 polynomials is the largest that has been attempted with *Algorithm P*. It required 40 hours to triangularize with the resulting matrix having a maximum coefficient length of 2115 digits.

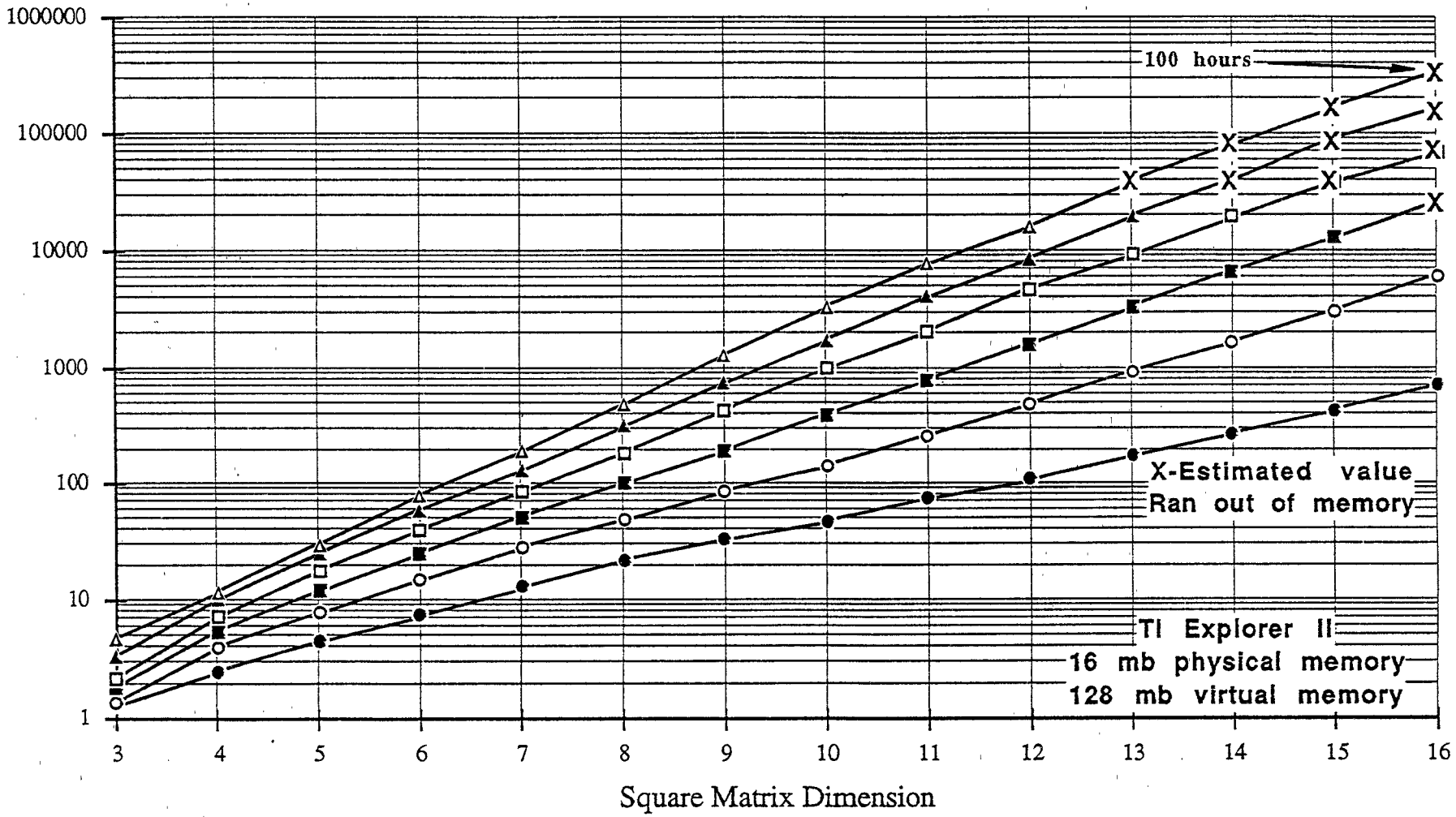
Although *Algorithm T* was faster than *Algorithm P* on the smaller matrices, it did not have the overhead of putting the matrix into a canonic form in the process; *Algorithm P* transforms the input matrix into the canonic integral-Hermite form as described earlier. The output matrix of *Algorithm T* therefore requires the application of an auxilliary algorithm to reduce the degree of the polynomial entries above the diagonal in order to put it in strict integral-Hermite form. Of course this is not necessary if one is only interested in rank information.

If one keeps in mind the fact that our simulation results were run on full, random matrices, which tend to yield worst-case performance, then these simulations indicate that our algorithms in their current state are ideally suited for problems in which $\max\{m, n\} \leq 9$. Such problems include many practical control system designs, textbook problems in a classroom/lab environment, and empirical error analyses involved in research for alternative apporaches to the machine computation of triangular forms of polynomial matrices based on other arithmetics such as floating-point or residue arithmetic [9]. For larger problems, our code can be modified in various ways to yield approximate results in much less time while providing some degree of error control. For instance, after the integer coefficients have reached a certain prespecified maximum size, the triangularization can be interrupted momentarily and the matrix $A(s)$ in $M[Z[s]]$ at its current state of triangularization can be converted to an *associated* matrix $A'(s)$ in $M[Q[s]]$ by premultiplication

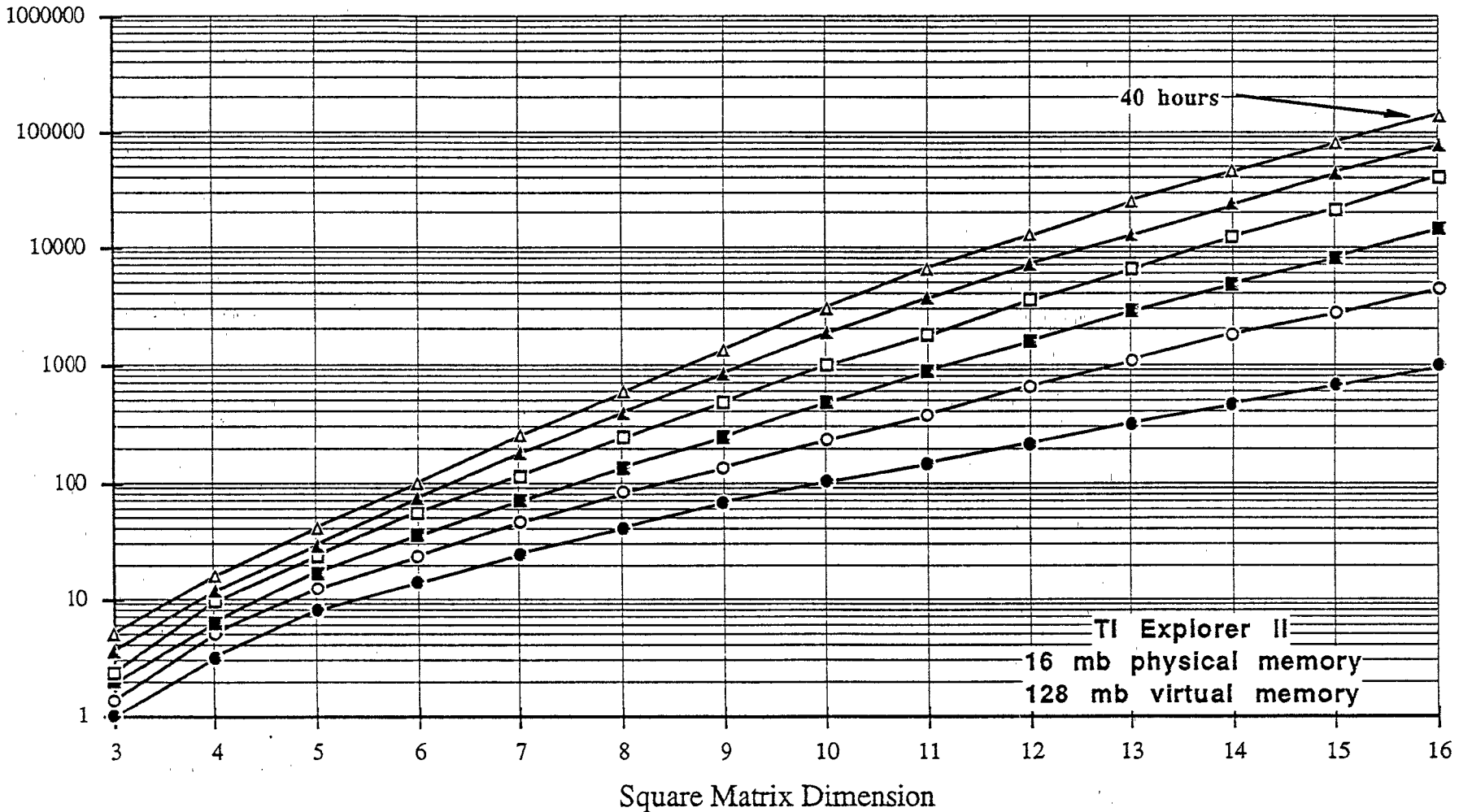
with a diagonal matrix in $M[Q]$. The matrix $A'(s)$ can then be "floated" to any desired decimal precision and then re-expressed as a matrix in $M[Q[s]]$ and finally converted back to $M[Z[s]]$ to continue the triangularization. An *ad hoc* technique such as this is certainly approximate but if done properly can yield better results than the *ad hoc* floating-point techniques currently used. Refinements of this idea for Algorithm P with error bounds and simulation results will be appear elsewhere.

We also compared our Hermite algorithm to the built-in Hermite algorithm included with the Scratchpad II and Maple computer algebra packages. On a 5×5 example generated randomly as above our code ran over 100 times faster.

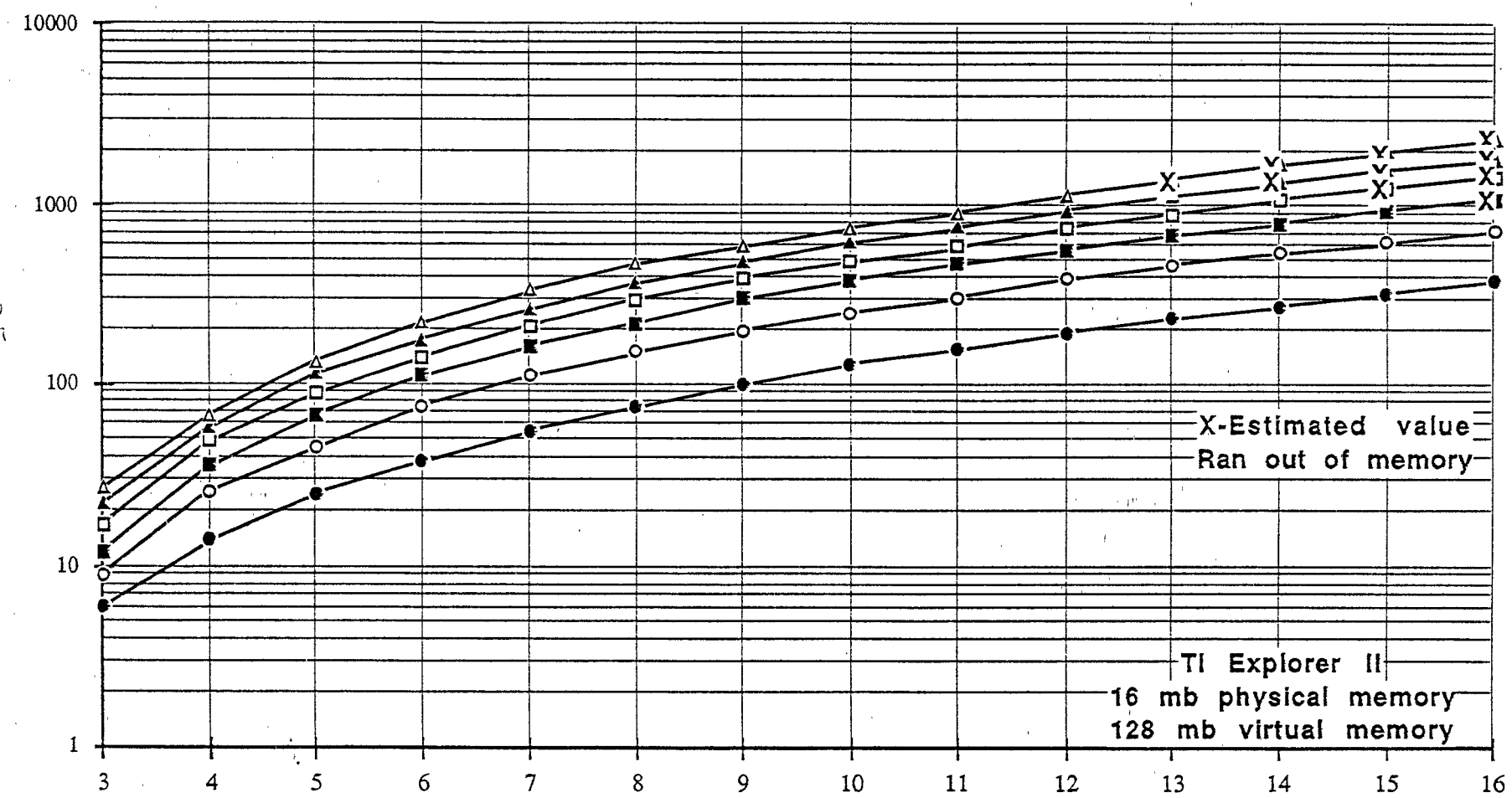
Time to Triangularize (sec) - Column Oriented Algorithm
Polynomial Degrees 1 thru 6



Time to Triangularize (sec) - Minor Oriented Algorithm
Polynomial Degrees 1 thru 6

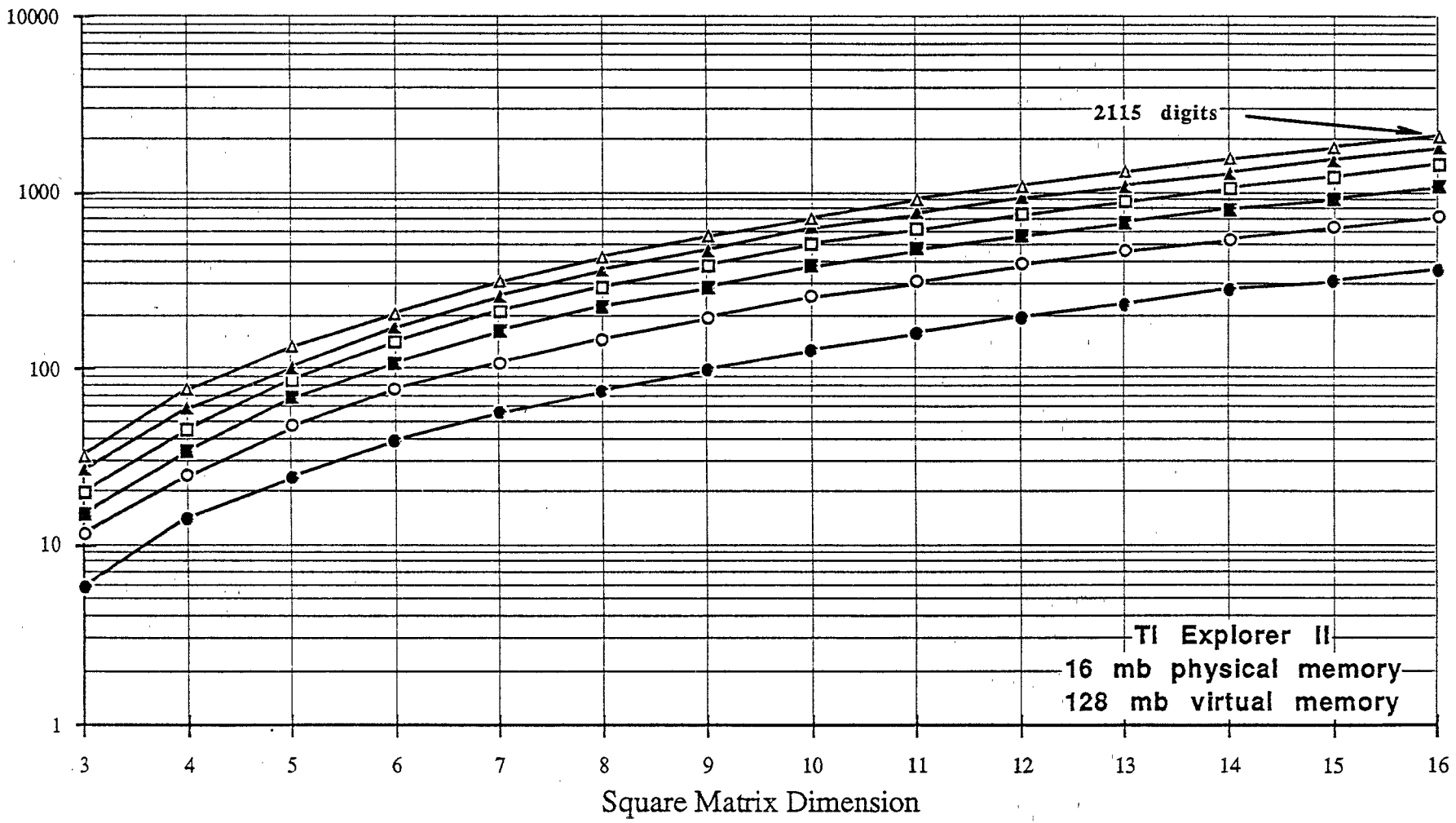


Maximum Coefficient Length (# of digits) - Column Oriented Algorithm
Polynomial Degrees 1 thru 6



Square Matrix Dimension

Maximum Coefficient Length (# of digits) - Minor Oriented Algorithm
Polynomial Degrees 1 thru 6



9. Summary of Functions

MACSYMA is a Lisp based system for performing formal, symbolic calculations using both error-free and arbitrary artihmetic. Since it is Lisp based, MACSYMA runs the fastest on *Lisp machines* - computers which have Lisp instructions hard-coded into their microprocessors, such as the Texas Instruments Explorer II. *Mathematica* is a relatively new computer algebra system written in 'C' which has capabilities similar to MACSYMA. We have been running *Mathematica* on a NeXT machine. If an input matrix has polynomials with decimal or floating-point coefficients, the coefficients must first be re-expressed as ratios of coprime integers—both MACSYMA and Mathematica have efficient, built-in functions to do this.

The following is a summary of the *high-level* auxiliary programs which we have to date implemented in MACSYMA and *Mathematica*. They perform most of the common, high-level tasks arising in the frequency-domain approach to control system synthesis.

- $RightMatrixFraction(H(s))$ — Computes a right matrix fraction description of the transfer function matrix $H(s)$, i.e., computes the matrices $N(s), D(s)$ such that $H(s) = N(s)\ D(s)^{-1}$. The LeftMatrixFraction description is analogously computed.

- $Bezout(N(s), D(s))$ — Finds the homogenous and particular solutions to the Bezout equation, i.e., finds polynomial matrices $X_p(s), Y_p(s), X_h(s), Y_h(s)$ such that $X_p(s)\ X(s) + Y_p(s)\ N(s) = I$ and $X_h(s)\ X(s) + Y_h(s)\ N(s) = 0$. Used for designing feedback compensators in the frequency domain.

- $ColumnReduce(D(s))$ — Column reduces the polynomial matrix $D(s)$, i.e., multiplies $D(s)$ by an appropriate unimodular matrix such that the matrix of leading coefficients of its entries is nonsingular. RowReduce is analogously computed.

- $Controller(H(s))$ — Finds a controller form realization of the transfer function matrix $H(s)$. Controllability, Observer and Observability realizations are analogously computed.

- $Hermite(N(s))$ — Finds the canonic column Hermite form of the polynomial matrix $N(s)$.

- *RightCoprime*$(N(s), D(s))$ — Determines the greatest common right divisor of the polynomial matrices $N(s)$ and $D(s)$. If it is not unimodular, it is *factored out* of both matrices making them right coprime. Used for finding *minimal* realizations. LeftCoprime is analogously computed.

- *Smith*$(N(s))$ — Finds the Smith form of the polynomial matrix $N(s)$. This is a canonic, diagonal form of a polynomial matrix. *SmithMcMillan*$(H(s))$ — Finds the Smith-McMillan form of the transfer function matrix $H(s)$. This is a canonic, rational, diagonal form of a matrix whose entries are ratios of polynomials.

Acknowledgements: This research was supported in part by NSF grant NSF CDR-8803012 under the Engineering Research Centers Program, and AFOSR University Research Initiative grant 87-0073. David MacEnany was partially supported by an IBM Fellowship.

9. References and Bibliography

[1] Bareiss, E.H. "Computational Solutions of Matrix Problems over an Integral Domain" *J. Inst. Maths Applics* V10, 69-104, 1972

[2] Bareiss, E.H. "Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination" *Math. Comp.* V22, 565-578, 1968

[3] Brown, W.S. "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors" *J. ACM* V18 (4) 478-504 Oct 71

[4] Brown, W.S. and J.F. Traub "On Euclid's Algorithm and the Theory of Subresultants" *J. ACM* V18 (4) 505-514 Oct 71

[5] Buchberger, B. & G.E. Collins et al (eds.) *Computer Algebra: Symbolic and Algebraic Computation* Wein: Springer,1982

[6] Chou, T.J. and G.E. Collins "Algorithms for the Solution of Systems of Linear Diophantine Equations" *Siam. J. Comp.* V11 (4) 687-708 Nov 82

[7] Collins, G.E. "Subresultants and Reduced Polynomial Remainder Sequences" *J. ACM* V14 (1) 128-142 Jan 67

[8] Gantmakher, F.R. *Theory of Matrices* New York: Chelsea, 1959

[9] Gregory, R.T. and E.V. Krishnamurthy *Methods and Applications of Error-Free Computation* Berlin: Springer, 1984

[10] Hartley, B. and T.O. Hawkes *Rings, Modules and Linear Algebra* London: Chapman and Hall, 1970

[11] Holmberg, U.,"Some MACSYMA Functions for Analysis of Multivariable Linear Systems", Technical Report CODEN:LUTFD2/(TFRT-7333)/1-040/(1986), Department of Automatic Control, Lund Institute of Technology, October 1986.

[12] Hungerford, T.W. *Algebra* Berlin: Springer, 1974

[13] Kailath, T. *Linear Systems* Englewood Cliffs: Prentice-Hall, 1980

[14] Kaltofen, E. & S.M. Watt (eds.) *Computers and Mathematics* Berlin: Springer, 1989

[15] Kannan, R. "Solving Systems of Linear Equations over Polynomials" Report CMU-CS-83-165, Dept. of Comp. Sci., Carnegie-Mellon University, Pittsburgh, 1983

[16] Kannan, R. and A. Bachem "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix" *Siam. J. Comp.* V8 (4) 499-507 Nov 79

[17] Keng, H.L. *Introduction to Number Theory* Berlin: Springer, 1982

[18] Knuth, D.E. *The Art of Computer Programming, Vol. 2* Reading, Mass.:Addison Wesley,1981

[19] Krishnamurthy, E.V. *Error-Free Polynomial Matrix Computations* Berlin: Springer, 1985

[20] Lipson, J.D., *Elements of Algebra and Algebraic Computing* Reading: Addison-Wesley, 1981

[21] MacDuffee, C.C. *The Theory of Matrices* New York: Chelsea, 1950

[22] McClellan, M.T. "The Exact Solution of Systems of Linear Equations with Polynomial Coefficients" *J. ACM* V20 (4) 563-588 Oct 73

[23] Newman, M. *Integral Matrices* New York: Academic Press, 1972

[24] Vidyasagar, M., *Control System Synthesis* Cambridge: MIT Press, 1985

[25] Wolovich, W.A., *Linear Multivariable Systems* Berlin: Springer, 1974