Journal of Symbolic Computation

Journal of Symbolic Computation 43 (2008) 359–376

Approximate Factorization of Multivariate Polynomials Using Singular Value Decomposition

Erich Kaltofen[a], John P. May[b,1], Zhengfeng Yang[c], Lihong Zhi[c]

[a]*Dept. of Mathematics, North Carolina State University, Raleigh, North Carolina, 27695-8205, USA*
[b]*School of Computer Science, University of Waterloo, Waterloo, Ontario Canada*
[c]*Key Lab of Mathematics Mechanization, AMSS, Beijing 100080, China*

Submitted 13 January 2006; revised received 18 April 2007; accepted 19 November 2007

Abstract

We describe the design, implementation and experimental evaluation of new algorithms for computing the approximate factorization of multivariate polynomials with complex coefficients that contain numerical noise. Our algorithms are based on a generalization of the differential forms introduced by W. Ruppert and S. Gao to many variables, and use singular value decomposition or structured total least squares approximation and Gauss-Newton optimization to numerically compute the approximate multivariate factors. We demonstrate on a large set of benchmark polynomials that our algorithms efficiently yield approximate factorizations within the coefficient noise even when the relative error in the input is substantial (10^{-3}).

Key words: multivariate polynomial factorization, approximate factorization, singular value decomposition, numerical algebra, Gauss-Newton optimization

1. Introduction

When the scalars in the inputs to a symbolic computation are given as floating point numbers, often with added noise that may come as the result of a preceding numerical computation or a physical measurement, the desired singular properties of the problem formulations can be lost. We shall consider the problem of factoring a multivariate polynomial into its complex factors. Let $f(x_1,\dots,x_n) \in \mathbb{Q}(\mathbf{i})[x_1,\dots,x_n]$ be irreducible over $\mathbb{C}$, where irreducibility is caused by

Email addresses: `kaltofen@math.ncsu.edu` (Erich Kaltofen), `jpmay@uwaterloo.ca` (John P. May), `zyang@mmrc.iss.ac.cn` (Zhengfeng Yang), `lzhi@mmrc.iss.ac.cn` (Lihong Zhi).

[1] Tel.: +1 519 747 2373; fax: +1 519 747 5284.

doi:10.1016/j.jsc.2007.11.005

perturbations on the coefficients of f. By $f^{[\min]}$ we denote a factorizable polynomial over $\mathbb{C}$ with $\deg(f^{[\min]}) \leq \deg(f)$ such that $\|f - f^{[\min]}\|_2$ is minimized, that is, $f^{[\min]}$ is a nearest reducible polynomial. We present new algorithms that can find a factorization $\widetilde{f} = f_1 \cdot f_2 \cdots f_r$ in $\mathbb{C}[x_1, \ldots, x_n]$ with $\deg(\widetilde{f}) \leq \deg(f)$ such that $\|\widetilde{f} - f^{[\min]}\|_2$ is small.

In (Kaltofen and May, 2003, Example 2) it was discovered that $f^{[\min]}$ is dependent on the degree notion. Our bounds such as $\deg(\widetilde{f}) \leq \deg(f)$ limit the degrees in the individual variables, that is $\deg_{x_i}(\tilde{f}) \leq \deg_{x_i}(f)$ for $i = 1, \ldots, n$. One of the authors, L. Zhi, had in Nov. 2002 considered to apply S. Gao's exact polynomial factoring algorithm (see Gao, 2003) for numerical coefficients. Independently, in the conclusion of (Kaltofen and May, 2003) we have suggested that structural minimal deformations that achieve the necessary rank deficiency of the Ruppert matrices arising in S. Gao's (2003) algorithm yield an approximate factorization algorithm. In (Gao et al., 2004) we have jointly designed and implemented a hybrid symbolic-numeric variant of Gao's bivariate polynomial factoring algorithm. Our algorithm computes an unstructured singular value decomposition followed by a newly designed approximate bivariate greatest common divisor algorithm. Here we present a multivariate generalization and, following (Zeng and Dayton, 2004), we introduce Gauss-Newton post-iteration, which can significantly improve the accuracy of the approximate factorization. As an alternative, one can use a structured total least squares deformation (Park et al., 1999; Lemmerling et al., 2000) of the Ruppert matrices, which we demonstrate to be a feasible approach.

We present experimental evidence that our new approach improves the approximate factorizations of (Gao et al., 2004). The difficulty of a satisfying numerical analysis of any of our algorithms are the notions of “near” and “small”. Our experiments show that our algorithms perform well even for polynomials with a relatively large irreducibility radius (Nagasaka, 2002; Kaltofen and May, 2003; Nagasaka, 2005).

There is an extensive literature on the problem of factoring multivariate polynomials over the real or complex numbers. In (Kaltofen, 1985) one of the first polynomial-time algorithms is given for input polynomials with exact rational or algebraic number coefficients, and the problem of approximate factorization is already discussed there (Kaltofen, 1985, section 6). Approximate factorization algorithms suppose that the input coefficients are perturbed and consequently, the input polynomial is irreducible over $\mathbb{C}$ under an exact interpretation of its coefficients. However, if the input polynomial is near its factorizable counterpart, say within machine floating point precision, one can attempt to run exact methods with floating point arithmetic, such as Hensel lifting, computing zero-sum relations of power series roots, or interpolating the irreducible factors as curves. The work reported in (Sasaki et al., 1991, 1992; Galligo and Watt, 1997; Huang et al., 2000; Sasaki, 2001; Galligo and Rupprecht, 2001; Corless et al., 2001, 2002; Galligo and Rupprecht, 2002; Rupprecht, 2004; Sommese et al., 2004) studies recovery of approximate factorization from the numerical intermediate results. For significant noise, which is the setting we study, those methods can suffer from stability problems. For instance, the approximate zero sums are now far from zero. A somewhat related topic are algorithms that obtain the exact factorization of an exact input polynomial by use of floating point arithmetic in a practically efficient way (Chèze, 2004).

A different line of methods bounds from below the distance from the input polynomial to the nearest factorizable polynomial, that is, the irreducibility radius (Nagasaka, 2002; Kaltofen and May, 2003; Nagasaka, 2005). Not only do such bounds help in declaring inputs numerically irreducible, they also provide insight in the quality of a computed approximate factorization.

No polynomial-time algorithm is known for computing the nearest factorizable polynomial $f^{[\min]}$, which is open problem 1 in (Kaltofen, 2000). In (Hitz et al., 1999) a polynomial-time algorithm is given for computing the nearest polynomial with a complex factor of constant degree. In practice, that algorithm is much slower than any of the numerical solutions—and the same may be expected of a future solution to the open problem—but for polynomials of degree 2 or 3 one can obtain an actual optimal answer with which one can further gauge the output of the fast but non-optimal numerical procedures.

With the algorithms presented in this paper, we have successfully computed improved approximate factorizations of all benchmark examples presented in the literature, including those with significant irreducibility radii introduced in (Gao et al., 2004).

2. Approximate Multivariate Polynomial Division and GCD

Our algorithms require, as a substep, the computation of approximate multivariate greatest common divisors of complex polynomials. Several of the algorithms available for approximate GCD further require an algorithm to compute approximate multivariate polynomial division, which we shall discuss first.

2.1. Approximate Polynomial Division

The simplest interesting problem in approximate polynomial algebra seems to be the problem of polynomial "exact" division. Multiplication by a given polynomial is a linear operation so we can represent multiplication of polynomials of total degree d by a given f as $C^{[d]}(f)$, the convolution matrix associated with f and d. For instance,

$$\overrightarrow{(a_2x+a_1y+a_0)\cdot(b_2x+b_1y+b_0)} = C^{[2]}(a_2x+a_1y+a_0)\cdot\begin{bmatrix} b_2\\ b_1\\ b_0\end{bmatrix}$$

$$= \begin{bmatrix} a_2 & 0 & 0\\ a_1 & a_2 & 0\\ 0 & a_1 & 0\\ a_0 & 0 & a_2\\ 0 & a_0 & a_1\\ 0 & 0 & a_0\end{bmatrix}\cdot\begin{bmatrix} b_2\\ b_1\\ b_0\end{bmatrix}.$$

Note that the convolution matrix can be formed for other notions of degree (or polynomials with a given support), but for simplicity of discussion we will use total degree in our descriptions in this section. The results carry over to all degree notions.

If we are given polynomials f and g with $\mathrm{tdeg}(g) \geq \mathrm{tdeg}(f)$ such that f does not divide g exactly then we want to apply a perturbation so that f does divide g. If we fix the coefficients of f then $\tilde{g}$, the closest polynomial to g that f divides, can be found by solving the least squares problem:

$$\min_{\mathrm{tdeg}(q)=\mathrm{tdeg}(g)-\mathrm{tdeg}(f)} \|fq-g\|_2. \tag{1}$$

We can write $\tilde{g}$ exactly in terms of a convolution matrix,

$$\tilde{g} = C^{[\mathrm{tdeg}(q)]}(f)\,(C^{[\mathrm{tdeg}(q)]}(f))^{\dagger}\,g$$

(where we are being intentionally sloppy about the distinction between g as a polynomial and g as a vector of its coefficients). In the univariate case, the coefficient matrix for the least squares problem (1) has a small displacement rank and the arising system can be solved efficiently (Zhi, 2003).

One of the shortcomings of this approach is, although it does solve the approximation problem completely, it does not allow for f to vary as well (or instead of) g. However, it is very easy to implement, and the results it provides seem good enough for our purposes.

If one wished to allow perturbations of the coefficients of both f and g, then the division problem becomes a total least squares (TLS) problem (perturbations are allowed in the right-hand side vector as well as the entries of the matrix). In fact, since the matrix $C^{[d]}(f)$ has a very specific structure, approximate division becomes a structured total least squares (STLS) problem.

2.2. *Approximate Polynomial GCD*

For completeness we restate the algorithm in (Gao et al., 2004). A very similar multivariate approximate GCD algorithm was proposed independently in (Zeng and Dayton, 2004) but a pre-specified tolerance ε is required there. In addition, a Gauss-Newton iteration step is introduced to improve the GCD further. In practice just a few steps of iteration can improve the backward error by at least an order of magnitude and so it is usually worth the extra computation, especially when the g and h started quite close to a pair with a non-trivial GCD. For example, if g and h are nearly machine precision distance from a pair with an exact GCD, the approximate GCD computed from the SVD method is generally limited to about half of the machine precision, while Gauss-Newton iteration can usually improve the result to exact within full machine precision.

Algorithm 1 (AMVGCD: Approximate Multivariate GCD).

```
INPUT: $g$ and $h$ in $\mathbb{C}[x_1,\ldots,x_n]$
OUTPUT: $d$, a non-constant approximate GCD of $g$ and $h$

(1) Determine $k$, the degree of the approximate GCD of $g$ and $h$, in one of the two ways below:
    (a) Form $S = \mathrm{S}_1(g,h)$, the matrix of the linear system $ug+vh=0$, where $g,h \in \mathbb{C}[x_1,\ldots,x_n]$
        with $\mathrm{tdeg}(u) < \mathrm{tdeg}(h)$ and $\mathrm{tdeg}(v) < \mathrm{tdeg}(g)$. Find the largest gap in the singular values of $S$ and infer the degree from the numerical rank of $S$.
    (b) Compute the degrees of the GCDs of several random univariate projections of $g$ and $h$
        by looking for the numerical rank of the corresponding univariate Sylvester matrices.
(2) Reform $S$ as $\mathrm{S}_k(g,h)$ that is, use $\mathrm{tdeg}(u) = \mathrm{tdeg}(h) - k$ and $\mathrm{tdeg}(v) = \mathrm{tdeg}(g) - k$ as the
    constraints on $u$ and $v$ in the linear system in the first step. This new $S$ will have a dimension 1 nullspace.
(3) Compute a basis for the nullspace of $S$ by computing the singular vector corresponding the
    smallest singular value of $S$. This vector gives a solution $[u,v]^T$.
(4) Find $d$, the approximate quotient of $h$ and $u$ (or $g$ and $v$); alternately minimize $\|h-d\,u\|_2^2 +$
    $\|g+d\,v\|_2^2$, using least squares.
```

If one wishes to specify a tolerance, then only the first step of the Algorithm 1 is affected. In that case, it is possible that the computation of the degree could yield $k=0$, in which case the method would return $d=1$, declaring g and h to be approximately relatively prime to the given tolerance.

We wish to add that there now exist viable alternatives to both the approximate division and approximate GCD problem based on structured total least squares approximation (Kaltofen et al., 2006).

3. The Factorization Algorithm and Experiments

In this section, we propose an approximate factoring algorithm for multivariate polynomials over $\mathbb{C}$. The algorithm is a generalization of the bivariate factoring algorithm in (Gao et al., 2004). In addition, we have incorporated Gauss-Newton post-refinement of the approximate factors. Much like the GCD algorithm presented above, our factoring algorithm relies on singular value decomposition. We have implemented our algorithm in Maple 10 and we present benchmark tests.

3.1. The Exact Factoring Algorithm

We briefly describe the multivariate generalization of the bivariate factoring algorithm in (Gao, 2003), more details can be found in (May, 2005).

Assume that f is non-constant and $\gcd(f, f_{x_1}) = 1$ where $f_{x_1} = \partial f/\partial x_1$, which makes f both square-free and with no factor in $\mathbb{C}[x_2, \ldots, x_n]$. Suppose that f factors as

$$f = f_1 f_2 \cdots f_r, \tag{2}$$

where $f_i \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ are distinct and irreducible over $\mathbb{C}$. Define

$$E_i = \frac{f}{f_i}\frac{\partial f_i}{\partial x_1} \in \mathbb{C}[x_1, x_2, \ldots, x_n]\ 1 \le i \le r. \tag{3}$$

Then

$$f_{x_1} = E_1 + E_2 + \cdots + E_r \text{ and } E_i E_j \equiv 0 \bmod f \text{ for all } i \ne j. \tag{4}$$

The following fact, for two variables stated first in (Ruppert, 1986), gives a test for irreducibility and is a key part of the factoring algorithm.

Fact 2. *Suppose $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ with multi-degree $(d_1, d_2, \ldots, d_n)$, i.e., $\deg_{x_i} f = d_i$. Then f is absolutely irreducible if and only if the equations*

$$\frac{\partial}{\partial x_i}\left(\frac{g}{f}\right) = \frac{\partial}{\partial x_1}\left(\frac{h_i}{f}\right),\ i = 2, \ldots, n \tag{5}$$

have no nonzero solution $g, h_2, \ldots, h_n \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ with

$$\left.\begin{aligned} \deg g &\le (d_1 - 2, d_2, \ldots, d_n), \\ \deg h_i &\le (d_1, d_2, \ldots, d_i - 1, \ldots, d_n), i = 2, \ldots, n. \end{aligned}\right\} \tag{6}$$

Since differentiation is linear over $\mathbb{C}$, the equations (5) give a linear system for the coefficients of g and h_i, whose coefficient matrix we call the Ruppert matrix $\mathrm{Rup}(f)$. The matrix $\mathrm{Rup}(f)$ of f is full rank if and only if f is absolutely irreducible. Using the criterion in Fact 2, (Kaltofen and May, 2003) provides some separation bounds for testing whether a numerical polynomial is absolutely irreducible, given a certain tolerance on its coefficients. When these bounds are small, one may suspect the polynomial f to be near a reducible polynomial. In the following, we explain how to use Fact 2 for factorization.

First, let us note that, similar to (Gao, 2003) and (Kaltofen and May, 2003), the degree conditions on g and the h_i are changed to:

$$\left.\begin{aligned} \deg g &\le (d_1-1, d_2, \ldots, d_n), \\ \deg h_i &\le (d_1, d_2, \ldots, d_i-1, \ldots, d_n), i = 2, \ldots, n, \end{aligned}\right\} \tag{7}$$

which allows for the solution $(f_{x_1}, f_{x_2}, \ldots, f_{x_n})$ even when f is irreducible. We use $\mathrm{Rup}_1(f)$ to denote the slightly larger coefficient matrix of (5) using the bounds (7).

Theorem 3. *Let $f \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ be a non-constant polynomial of multi-degree $(d_1, d_2, \ldots, d_n)$ with $\gcd(f, f_{x_1}) = 1$. Define*

$$G = \{g \in \mathbb{C}[x_1, x_2, \ldots, x_n] : (5) \text{ and } (7) \text{ hold for some } h_2, \ldots, h_n \in \mathbb{C}[x_1, x_2, \ldots, x_n]\} \tag{8}$$

Suppose f has the factorization into irreducible polynomials as in (2). Then G is a vector space over $\mathbb{C}$ of dimension r and each $g \in G$ is of the form $g = \sum_{i=1}^{r} \lambda_i E_i$ where $\lambda_i \in \mathbb{C}$.

The proof that follows is a direct multivariate generalization of Gao (2003, Theorem 2.3). First, for any $g_k = E_k = \frac{f}{f_k}\frac{\partial f_k}{\partial x_1}$, let $h_{k,i} = \frac{f}{f_k}\frac{\partial f_k}{\partial x_i}$, for $i = 2, \ldots, n$. Then $(g_k, h_{k,2}, \ldots, h_{k,n})$ satisfies (7) and

$$\frac{\partial}{\partial x_i}\left(\frac{g_k}{f}\right) = \frac{\partial}{\partial x_i}\left(\frac{1}{f_k}\frac{\partial f_k}{\partial x_1}\right) = \frac{\partial}{\partial x_1}\left(\frac{1}{f_k}\frac{\partial f_k}{\partial x_i}\right) = \frac{\partial}{\partial x_1}\left(\frac{h_{k,i}}{f}\right),$$

for $i = 2, \ldots, n$. So $E_1, \ldots, E_r \in G$. Since $E_1, \ldots, E_r$ satisfy (4), they are linearly independent over $\mathbb{C}$. Hence $\dim_{\mathbb{C}} G \ge r$.

Let $g \in G$ with $h_2, \ldots, h_n \in \mathbb{C}[x_1, x_2, \ldots, x_n]$ satisfying (5) and (7). We need to show that g is a linear combination of $E_1, \ldots, E_r$ over $\mathbb{C}$. Since $\gcd(f, f_{x_1}) = 1$, f has no repeated roots in the algebraic closure of $\mathbb{C}(x_2, \ldots, x_n)$. Let

$$f = u_{d_1}\prod_{i=1}^{d_1}(x_1 - c_i), c_i \in \overline{\mathbb{C}(x_2, \ldots, x_n)}.$$

Due to $\deg_{x_1} g < \deg_{x_1} f$ and $\deg_{x_1} h_j \le \deg_{x_1} f$, we have the partial fraction decompositions

$$\frac{g}{f} = \sum_{i=1}^{d_1}\frac{a_i}{x_1 - c_i}, \quad \frac{h_j}{f} = \sum_{i=1}^{d_1}\frac{b_{ji}}{x_1 - c_i} + h_j^*,$$

where $b_{ji} \in \overline{\mathbb{C}(x_2, \ldots, x_n)}$, $\deg_{x_1} h_j^* = 0$,

$$a_i = g(c_i, x_2, \ldots, x_n)/f_{x_1}(c_i, x_2, \ldots, x_n). \tag{9}$$

Since

$$\frac{\partial}{\partial x_j}\left(\frac{g}{f}\right) = \sum_{i=1}^{d_1}\left(\frac{1}{x_1 - c_i}\frac{\partial a_i}{\partial x_j} + \frac{a_i}{(x_1 - c_i)^2}\frac{\partial c_i}{\partial x_j}\right), \qquad \frac{\partial}{\partial x_1}\left(\frac{h_j}{f}\right) = \sum_{i=1}^{d_1}\frac{-b_{ji}}{(x_1 - c_i)^2}.$$

The equation (5) implies that $\frac{\partial a_i}{\partial x_j} = 0$. It follows that $a_i \in \mathbb{C}$ for $i = 1, \ldots, d_1$ If c_i and c_j are algebraic conjugates over $\overline{\mathbb{C}(x_2, \ldots, x_n)}$, then so are a_i and a_j because of (9); hence $a_i = a_j$ since they are in $\mathbb{C}$. Therefore a_i is constant for c_i in the same conjugate class which corresponds to an irreducible factor of f over $\mathbb{C}(x_2,\ldots,x_n)$. So as in (Gao, 2003), we have that

$$\frac{g}{f} = \sum_{i=1}^{r} \lambda_i \frac{1}{f_i}\frac{\partial f_i}{\partial x_1},$$

where $\lambda_i \in \mathbb{C}$. Therefore, each $g \in G$ is of the form $g = \sum_{i=1}^{r} \lambda_i E_i$ and $\dim_{\mathbb{C}} G = r$. □

Now we show how to extract the factors of f from the linear space G. This is the direct generalization of Gao (2003, Theorems 2.9 and 2.10):

Fact 4. *Suppose that $g_1,\ldots,g_r$ form a basis for G over $\mathbb{C}$. Select $s_i \in S \subset \mathbb{C}$ uniform randomly and independently for all $1 \le i \le r$, and let $g = \sum_{i=1}^{r} s_i g_i$. There is a unique $r \times r$ matrix $A = [a_{i,j}]$ over $\mathbb{C}$ such that*

$$g g_i \equiv \sum_{j=1}^{r} a_{i,j} g_j f_{x_1} \pmod{f} \quad \textit{in } \mathbb{C}(x_2,\ldots,x_n)[x_1]. \tag{10}$$

Furthermore, let $E_g(x) = \det(Ix - A)$, the characteristic polynomial of A. Then the probability that

$$f = \prod_{\lambda\in\mathbb{C}:E_g(\lambda)=0} \gcd(f, g - \lambda f_{x_1}) \tag{11}$$

gives a complete factorization of f over $\mathbb{C}$ is at least $1 - r(r-1)/(2|S|)$, where $|S|$ denotes the cardinality of S.

Again, the proof is nearly exactly the same as the one in (Gao, 2003).

It is possible to reduce Gao's degree conditions (7) to Ruppert's (6) in the factoring algorithms. For completeness, we state the corresponding theorem, whose proof reveals how to modify the factoring algorithms.

Theorem 5. *Let $\gcd(f, f_{x_1}) = 1$. Then $\mathrm{Rup}(f)$ has rank deficiency $r-1$, i.e., the dimension of the nullspace of $\mathrm{Rup}(f)$ is $r-1$, where r is the number of irreducible factors of f over $\mathbb{C}$.*

The proof is based on Ruppert's original arguments (Ruppert, 1999, Section 3). Clearly, any solution $g, h_2,\ldots,h_n$ of Theorem 3 that satisfies the stricter bound (6) corresponds to a vector in the nullspace of $\mathrm{Rup}(f)$. For $k = 2,\ldots,r$ the polynomials

$$\underbrace{\deg_{x_1}(f_k)g_1 - \deg_{x_1}(f_1)g_k}_{\hat{g}_k}, \deg_{x_1}(f_k)h_{1,2} - \deg_{x_1}(f_1)h_{k,2}, \\ \ldots, \deg_{x_1}(f_k)h_{1,n} - \deg_{x_1}(f_1)h_{k,n},$$

where g_k and $h_{k,i}$ are as in the proof of Theorem 3, are such solutions, i.e., $\deg_{x_1}(\hat{g}_k) \le d_1 - 2$, because the leading coefficients in the variable x_1 of $\deg_{x_1}(f_k)g_1 = \deg_{x_1}(f_k)\frac{f}{f_1}\frac{\partial f_1}{\partial x_1}$ and $\deg_{x_1}(f_1)g_k = \deg_{x_1}(f_1)\frac{f}{f_k}\frac{\partial f_k}{\partial x_1}$ cancel. The corresponding coefficients vectors are linearly independent, so the rank deficiency of $\mathrm{Rup}(f)$ is at least $r-1$. There cannot be an additional linearly independent null vector, because otherwise all null vectors of $\mathrm{Rup}_1(f)$ would be spanned by those polynomial coefficient vectors, but the solution $g = \partial f/\partial x_1$, $h_2 = \partial f/\partial x_2, \ldots, h_n = \partial f/\partial x_n$ for $\mathrm{Rup}_1(f)$ is not in that span, since the degree in x_1 of g is too high. □

The following paragraph is omitted from the journal version.

Now suppose we have computed polynomials $v_2, \ldots, v_r$ whose coefficient vectors form a basis for the g-component of the nullspace of $\mathrm{Rup}(f)$. By basis transformation we have $v_j = \sum_k \mu_{j,k}\hat{g}_k$ for $j = 2, \ldots, r$, where $[\mu_{j,k}] \in \mathbb{C}^{(r-1)\times(r-1)}$ is non-singular. Therefore

$$\begin{aligned}\sum_j s_j v_j &= \sum_k (\sum_j s_j \mu_{j,k})\hat{g}_k \\ &= \underbrace{\left(\sum_k \deg_{x_1}(f_k)\sum_j s_j \mu_{j,k}\right)}_{\lambda_1(s_2,\ldots,s_k)} g_1 - \sum_k \underbrace{\left(\deg_{x_1}(f_1)\sum_j s_j \mu_{j,k}\right)}_{\lambda_k(s_2,\ldots,s_k)} g_k\end{aligned}$$

For symbolic s_j we have $\prod_{i<l}(\lambda_i(s_2,\ldots,s_k) - \lambda_l(s_2,\ldots,s_k)) \neq 0$, because the columns of $[\mu_{j,k}]$ are linearly independent. Note that the coefficient of s_j constitutes the j-th row of those columns. By Zippel/Schwartz $\sum_j s_j v_j$ has distinct λ_i with probability $(r-1)(r-2)/(2|B|)$.

End of material omitted from the journal version. © *authors.*

3.2. The Numerical Factoring Algorithm

In order to apply the factorization algorithm given in (Gao, 2003) and its multivariate generalization, given above, to approximate polynomials we must be able to solve the following problems:

(1) compute the approximate GCDs of multivariate polynomials: $\gcd(f, g - \lambda_i f_{x_1})$,
(2) reduce the polynomial f so that $\gcd(f, f_{x_1}) = 1$ approximately,
(3) determine the numerical dimension of G, and
(4) compute an E_g that has no cluster of roots.

For the first problem, the previous section discusses robust algorithms to compute the approximate GCDs of multivariate polynomials. The second problem is also handled by way of the approximate GCD; we can compute the approximate GCD of f and f_{x_1}. Then with an approximate division, $f/\gcd(f, f_{x_1})$, we may, heuristically, reduce to the case where $\gcd(f, f_{x_1}) = 1$ approximately. Details on this approach follow below in Section 3.4.

To solve the third problem we can determine the numerical dimension of G by the SVD of the matrix $\mathrm{Rup}_1(f)$. Let σ_i be the i^{th} singular value of $\mathrm{Rup}_1(f)$. If a tolerance ε is given, then the numerical dimension of G is the r such that

$$\cdots \geq \sigma_{r+2} \geq \sigma_{r+1} > \varepsilon \geq \sigma_r \geq \cdots \geq \sigma_1.$$

However, if we do not know the relative error in the coefficients of f, it is difficult to provide a tolerance ε that is consistent with the error in the data. If we have no tolerance given, we infer a tolerance from the largest gap in the singular values. That is, we choose $\varepsilon = \sigma_r$ so that σ_{r+1}/σ_r is as large as possible. As in (Kaltofen and May, 2003), the singular value σ_r bounds from below the distance from f to a polynomial $\tilde{f}$ that has r absolutely irreducible factors:

$$\min_{\substack{\deg \tilde{f} = (d_1,\ldots,d_n) \\ \dim \mathrm{Nullspace}(\mathrm{Rup}_1(\tilde{f})) = r}} \|\mathrm{Rup}_1(f) - \mathrm{Rup}_1(\tilde{f})\|_2 \geq \sigma_r.$$

This inferred tolerance σ_r can also be used as an input tolerance to the approximate multivariate GCDs at the end of the factorization algorithm.

Remark 6. Ideally, we could apply a structure preserving low rank approximation (SPLRA) as in (Park et al., 1999) to obtain a matrix $\tilde{R}$ which is closest to $\mathrm{Rup}_1(f)$ and has rank deficiency r. Since $\tilde{R}$ preserves structure of $\mathrm{Rup}_1(f)$, it corresponds to the Ruppert matrix of a polynomial $\tilde{f}$ which is the nearest polynomial that has exactly r absolutely irreducible factors. However, so far our experiments with applying various heuristics for SPLRA to this problem have had mixed results. So we leave details of research in this direction to be reported in future papers. In the following, we still use the SVD of the $\mathrm{Rup}_1(f)$ to find a nearby rank deficient matrix.

For the fourth problem, suppose we have obtained approximate basis $g_1,\ldots,g_r$ of G from the singular vectors corresponding to the last r singular values of $\mathrm{Rup}_1(f)$. It is easy to see that $\|\mathrm{Rup}_1(f)g_i\|_2 \le \sigma_i \le \sigma_r$. So the g_is form an approximate basis for G with tolerance σ_r. Following construction of the matrix A_g as described in Fact 4, we find a random element of G by choosing $s_1,\ldots,s_r \in S \subset \mathbb{C}$ uniform randomly, letting $g = \sum_{i=1}^{r} s_i g_i$ and substituting arbitrary values of $\alpha_i \in \mathbb{C}$ for x_i with the property that $f(x_1,\alpha_1,\ldots,\alpha_{n-1})$ remains square-free. The matrix A_g can be formed in the following manner: first reduce the polynomials gg_i and $g_j f_{x_1}$ modulo f (evaluated at $x_k = \alpha_k$) for $1 \le i, j \le r$ by using approximate division of univariate polynomials (see Section 2.1) then solve the least squares problem:

$$\min \|\mathrm{rem}(gg_i - (a_{i,1}g_1 f_{x_1} + \cdots + a_{i,r}g_r f_{x_1}),\ f)\|_2$$

to find the value of unknown elements $a_{i,j}$. Let $E_g(\lambda) = \det(I\lambda - A)$, the characteristic polynomial of A_g. We compute all the numerical roots $\lambda_1,\ldots,\lambda_r$ of the univariate polynomial E_g over $\mathbb{C}$ as the eigenvalues of A_g, and find the smallest distance between these roots:

$$\mathit{min\_dist}(g) = \min\{|\lambda_i - \lambda_j|,\ \ 1 \le i < j \le r\}.$$

If the distance is small then numerically E_g has a cluster of roots, and we should choose another set of s_is and try to find a separable E_g. In practice, since Fact 4 says g should give a separable E_g with high probability, we compute a number of random gs and keep the g with the largest $\mathit{min\_dist}(g)$.

In Gao's exact algorithm the absolutely irreducible factors are obtained from g by computing GCDs over algebraic extension fields given by the irreducible factors of E_g. In our case, all the roots of E_g are given as numerical values in $\mathbb{C}$. Hence there is no need to deal with field extensions, and we can compute directly in $\mathbb{C}$. We compute the multivariate approximate GCDs $\tilde{f}_i = \gcd(f, g - \lambda_i f_{x_1})$ according to the method in Section 2 for each numerical root λ_i of E_g and we obtain a proper approximate factorization of f over $\mathbb{C}$: $f \approx \prod_{i=1}^{r} \tilde{f}_i$.

Once we have computed an approximate factorization, there are a number of ways to improve it. First, we can compute a scaling c that minimizes the backward error of the approximate factorization:

$$\min_{c\in\mathbb{C}} \|f - c\prod_{i=1}^{r} \tilde{f}_i\|_2 / \|f\|_2.$$

The factorization can be improved further by solving a minimization problem (for example the one in (Huang et al., 2000)) or by setting up a minimization problem to which we can apply Gauss-Newton iteration, similar to what was done to refine the approximate GCD in

(Zeng and Dayton, 2004). First note that the optimization version of the approximate factorization problem is finding a least squares solution to the non-linear system of the form $F(v_1,\ldots,v_r) = f$ where $v_i \in \mathbb{C}[x_1,\ldots,x_n]$ and

$$F(v_1,\ldots,v_r) = \left[\, C^{[\mathrm{tdeg}(v_2\cdots v_r)]}(v_1)\cdots C^{[\mathrm{tdeg}(v_r)]}(v_{r-1})\, v_r \,\right].$$

Here $C^{[k]}(v)$ denotes the matrix of the linear map multiplication with polynomials of total degree k as described in subsection 2.1. Clearly there is a solution when $f = f_1\cdots f_r$ and $v_i = f_i$; otherwise we will solve

$$\min_{v_1,\ldots,v_r} \|F(v_1,\ldots,v_r) - f\|_2.$$

There exists such a minimum at one or more of the points where

$$(DF(v_1,\ldots,v_r))^H F(v_1,\ldots,v_r) = 0$$

(DF denotes the Jacobian of F). When formulated this way, it is easy to see that we can apply Gauss-Newton iteration to attempt to find the solution. That is, given an initial $[v_1^0, v_2^0,\ldots,v_r^0]$ we refine with the update

$$[v_1^{i+1},\ldots,v_r^{i+1}] = [v_1^i,\ldots,v_r^i] - (DF(v_1^i,\ldots,v_r^i))^{\dagger} F(v_1^i,\ldots,v_r^i).$$

Given the description of F above, the product rule gives that the Jacobian of F is a block matrix of the form:

$$DF(v_1,\ldots,v_r) = [C^{[\mathrm{tdeg}(v_1)]}(v_2 v_3\cdots v_r)\, C^{[\mathrm{tdeg}(v_2)]}(v_1 v_3\cdots v_r)\ldots C^{[\mathrm{tdeg}(v_r)]}(v_1 v_2\cdots v_{r-1})]$$

which has full rank (so long as not all the v_i's are 0) since every matrix $C^{[k]}(v)$ has full rank (so long as $v \neq 0$).

As with any type of Newton method, if the initial input $[v_1^0, v_2^0,\ldots,v_r^0]$ is close enough and DF is not rank deficient at the least squares solution, then the iteration will converge to the least squares solution according to Kelley (1999, Theorem 2.4.1):

Fact 7. *Let $w_0 = [v_1^0,\ldots,v_r^0]$ be the initial point and $w_\star = [v_1^\star,\ldots,v_r^\star]$ be a local minimum for F. If DF is full rank then there exist $K > 0$ and $\delta > 0$ so that if $\|w_0 - w_\star\| < \delta$ then the error of the Gauss-Newton iteration update at step k (e_k) satisfies:*

$$\|e_k\|_2 < K(\|e_{k-1}\|_2^2 + \|F(w_\star) - f\|_2\, \|e_{k-1}\|_2).$$

Although, as done with the GCD in Gao et al. (2004), it is possible to bound the distance of the output of the SVD method for factorization from the closest approximate factorization, that bound is quite large (exponentially large in the degree). We need a much tighter bound in order to prove something about when the output of the SVD method will be within the basin of attraction of the global minimum. Finally, it is worth mentioning that Fact 7 implies that Gauss-Newton iteration converges at a quadratic rate if the nearby local minimum is an exact factorization of f. Otherwise, iteration converges at a linear rate that is inversely proportional to the error of the factorization at the global minimum ($\|F(w_\star) - f\|_2$). In practice, the iteration converges in very few steps (≈ 7 for most polynomials tested).

3.3. Algorithm

Algorithm 8 (AFMP: Approx. Factoring Multivariate Polynomials).

INPUT: A polynomial $f \in \mathbb{C}[x_1,\ldots,x_n]$ such that f and f_{x_1} are approximately relatively prime, that is f is approximately square-free and has no approximate factors in $\mathbb{C}[x_2,\ldots,x_n]$ (see Section 3.4 below).

OUTPUT: A list of approximate factors f_i and an optimal scaling c.

Let S be a finite set $S \subset \mathbb{C}$ with $|S| \geq \mathrm{tdeg}(f)^2$. In our implementation $S = \{k/B \mid -B \leq k \leq B\}$ for a size parameter B.

```
(1) Compute approximate nullspace solutions:
    (a) Form the matrix $\mathrm{Rup}_1(f)$;
    (b) Compute the singular value decomposition of the Ruppert matrix, and find the last $\mathrm{tdeg}(f)+1$ singular values $\sigma_i$;
    (c) Find the biggest gap in the singular values and decide the numerical dimension $r$ of $G$, assuming $r \geq 2$;
    (d) Form a basis $g_1,\ldots,g_r$ of $G$ from the last $r$ right singular vectors of $\mathrm{Rup}_1(f)$.
(2) Compute an $E_g$ with well spaced roots:
    (a) Evaluate at randomly selected values for the variables $x_i = \alpha_i$ that do not change the degree or the square-free property of $f$;
    (b) For $k$ from 1 to $K$ do ($K = 4$ seems to work well in practice)
        (i) Pick $s_{i,k} \in S$ randomly, and set $\bar{g}_k = \sum_{i=1}^{r} s_{i,k}\, g_i$
        (ii) Compute $a_{i,j,k}$ that minimize the norm of the univariate remainder:
             $$\min \|\mathrm{rem}(\bar{g}_k g_i - \sum_{j=1}^{r} a_{i,j,k} g_j f_{x_1},\ f)\|_2;$$
        (iii) Let $E_{\bar{g}_k}(x) = \det(Ix - A)$, where $[A_{\bar{g}_k}]_{i,j} = [a_{i,j,k}]$. Compute the numerical roots $\lambda_{i,k}, 1 \leq i \leq r$ of $E_{\bar{g}_k}$ (the numerical eigenvalues of $A_{\bar{g}_k}$) and set $min\_dist_k = \min_{1 \leq i < j \leq r}\{|\lambda_{i,k} - \lambda_{j,k}|\}$;
    (c) Let $g = \bar{g}_k$ where $min\_dist_k$ is maximal.
(3) Compute factors via approximate GCDs:
        Compute $f_i = \gcd(f, g - \lambda_i f_{x_1})$ over $\mathbb{C}[x_1,\ldots,x_n]$ for $1 \leq i \leq r$.
(4) Solve optimizations to refine the factorization:
    (a) Apply Gauss-Newton iteration to improve the approximate factors;
    (b) Compute $\min_{c \in \mathbb{C}} \|f - c\prod_{i=1}^{r} f_i\|_2 / \|f\|_2$.
```

Remark 9. Two aspects need to be clarified about the approximate GCD computation in step 3. First, in order to obtain the degree of the multivariate GCD, it is fastest to project to univariate problems using a tolerance ε in Step 1 of Algorithm AMVGCD in Section 2.2. The use of σ_r as the tolerance for the approximate GCD is not accurate due to the large norm of the projected univariate polynomials, and must be increased (e.g. multiplied by the ratio of the norm of the projected polynomial to the norm of the original polynomial) to obtain suitable GCDs. Second, it should also be noted that Gauss-Newton refinement will not be used on the approximate GCD computations performed in the AFMP algorithm; it will only be used on the factorization. Experiments seem to indicate that using refinement on the approximate GCD computations leads to better pre-refinement factorizations, but that the Gauss-Newton iterations converge to the same factorization as when refinement was not used in the GCD computations.

Remark 10. It is clear that output polynomials cannot be guaranteed to be approximately irreducible. For example, in the case that the input does not lie near a factorizable polynomial then the approximate GCDs may place a factor near a reducible polynomial. One may, of course, always achieve approximate irreducibility certification by applying the test given in (Kaltofen and May, 2003) (generalized to multivariate polynomials in (May, 2005)) to the produced factors and apply the algorithm again if necessary.

Remark 11. The shape of the matrix $\mathrm{Rup}_1(f)$ depends on the degree of f in each x_i. Thus, Algorithm 8 will find an approximate factorization of f which will have the same (or smaller) degree in each x_i but one which may have higher total degree than f. If one wishes to find an approximate factorization with the same total degree as f one can use a structured version of the Ruppert matrix that depends on the total degree of f. For more details see (May, 2005, Section 3.2).

In the examples and implementation below, the total degree preserving version of the Ruppert matrix has been used. Product of the approximate factors found will always have total degree less than or equal to the the total degree of the original polynomial. Note that it is possible that the nearest polynomial that factors may have higher total degree. See (Kaltofen and May, 2003) for details.

Remark 12. For the approximate algorithm, our choice of S is based on experimental success. In fact, we worked with $B = 10$ with good results, requiring re-selection of $\bar{g}_2, \ldots$ in Step (2.b.i) rarely.

3.4. Multiple Factors

In the case that f is quite close to a polynomial that is not square-free, our factorization algorithm does not work well. This is related to the fact that the exact algorithm does not work at all on polynomials with repeated factors. In that case $\mathrm{Rup}_1(f)$ has many extraneous null vectors that do not correlate with factors (at least, not in the same way). When an irreducible polynomial is near a repeated factor polynomial, the approximate null vectors and numerical rank of $\mathrm{Rup}_1(f)$ exhibit some of these same problems. Another, similar but lesser problem is the removal of approximate factors in $\mathbb{C}[x_2, \ldots, x_n]$, that essentially amounts to a multivariate approximate GCD computation of several polynomials (Kaltofen et al., 2006).

One method to deal with the non-square-free case is to compute f_{sqfr}, the approximate quotient of f and the approximate GCD of f and f_{x_1} (Gao et al., 2004). Then compute the distinct approximate factors of $f_{\mathrm{sqfr}} \approx f_1 \cdots f_r$ using our algorithm. Finally, determine powers for each factor by looking for gaps in the sequence $\alpha_{i,j} = \sigma_1(S_1(f_i, \partial_{x_1,j} f))$.

We can only definitively call f approximately square-free if all of the nearest polynomials that factor are square-free. We cannot compute the nearest polynomial that factors, but we can bound the distance to the nearest polynomial that factors using the singular values of $\mathrm{Rup}_1(f)$ as in (Kaltofen and May, 2003), and similarly bound the distance to the nearest polynomial that is not square-free using the singular values of $S_1(f, f_{x_1})$. If the two bounds are very close we have to compute the factorization both ways and use the one with smaller backwards error.

In (Zeng and Dayton, 2004) a different method is proposed, that is based entirely on multivariate approximate GCDs and that generalizes the univariate algorithm in (Zeng, 2003). Experimentally, the two approaches seem to work similarly well (compare the example 14 from the table below to the ASFF example in (Zeng and Dayton, 2004)).

3.5. A Factoring Example

We illustrate our algorithm by factoring the following noisy polynomial (from (Kaltofen, 2000)) over the complex numbers:

$$f := 81x^4 + 72x^2y^2 + 0.002x^2z^2 - 648x^2 + 16y^4 + 0.001y^2z^2 \\ - 288y^2 + 1296 - 648.003z^4 - 0.007z^2.$$

The above polynomial is obtained by multiplying

$$(9x^2 + 4y^2 + 25.45596z^2 - 36)(9x^2 + 4y^2 - 25.45578z^2 - 36),$$

and rounding to three decimal places. Since $\deg f = (4, 4, 4)$, the Ruppert matrix (with respect to total degree) is 168×60. The last several singular values of the matrix are:

$$\cdots, 198.661, 145.253, 0.868 \times 10^{-10}, 0.431 \times 10^{-12}.$$

Starting from the second smallest singular value, the biggest gap is

$$145.253/0.868 \times 10^{-10} = 0.167 \times 10^{13}.$$

So $r = 2$ and f is supposed to be close to a polynomial having two irreducible factors. A basis for G computed from the last two right singular vectors is:

$$g_1 = 0.000151524784x^3 - 0.000606279x + 0.000067324xy^2 + 0.233157269xz^2,$$
$$g_2 = 0.108724346x^3 - 0.434897383x + 0.048321932xy^2 - 0.000322604xz^2.$$

Take a random linear combination of $g = g_1 + g_2$ and set $y = 2$ and $z = -1$. $A = [a_{i,j}]$ can be computed as

$$\begin{bmatrix} 0.000336771 & 0.000192632 \\ 0.000335102 & 0.000335302 \end{bmatrix}$$

Two eigenvalues of the matrix A are $\lambda_1 = 0.000590107$, $\lambda_2 = 0.000081966$. Computing $f_i = \gcd(f, g - \lambda_i f_x)$, $i = 1, 2$, we obtain two factors of f:

$$f_1 = 406.598x^2 + 180.710y^2 - 1150.03z^2 - 1626.39,$$
$$f_2 = 406.596x^2 + 180.709y^2 + 1150.04z^2 - 1626.39.$$

By finding an optimal scaling factor $c = 0.00048996$ that minimizes $\|f - c f_1 f_2\|_2$ we find the factorization:

$$\sqrt{c} f_1 = 9.000015518x^2 + 4.000009094y^2 - 25.45583592z^2 - 36.00004257,$$
$$\sqrt{c} f_2 = 8.999984448x^2 + 3.999990906y^2 + 25.45597017z^2 - 35.99995743,$$

that has a backward error $\|f - c f_1 f_2\|_2 / \|f\|_2 = 4.67 \times 10^{-13}$. We apply Gauss-Newton iteration and find an improved factorization (rounded to 10 decimal places):

$$f_1 = 9.000015552x^2 + 4.000009094y^2 - 25.455835924z^2 - 36.000042565,$$
$$f_2 = 8.999984448x^2 + 3.999990906y^2 + 25.455970172z^2 - 35.99995743,$$

that has backward error $\|f - f_1 f_2\|_2 / \|f\|_2 = 3.23 \times 10^{-14}$

3.6. Implementation and Experiments

The AFMP algorithm and its variants have been implemented in Maple and tests are reported in (Gao et al., 2004). There, Gauss-Newton iteration was not used to improve approximate GCDs or the final factorization. Here we report the results of repeating the experiments with iterative improvement in Table 1. Timings are given for some well known and some randomly generated examples run on a Pentium 4 at 2.0 GHz for $Digits = 14$ in Maple 10 under Windows. Here *coeff. error* indicates the noise imposed on the input, namely the relative 2-norm coefficient error to the original product of polynomials. Both *backward error* and *bkwd. err. w/ iter.* are relative errors, namely $\|f - \prod_i \tilde{f}_i\|_2 / \|f\|_2$. Please note that some incorrectly stated backward errors in (Gao et al., 2004) have been corrected here. The *time* is that for the entire factorization in seconds of a single run; the timings on a given example can vary significantly (up-to a factor of 4) depending on the random choices made in the algorithm; the Gauss-Newton iteration is generally less than 10% of the total time. The column *iters* is the number of Gauss-Newton iterations that were run before convergence – further iteration did not improve the factorization. Notice that the number of iterations increases as the backward error of the solution found increases (as discussed in the paragraph following Fact 7). The column *impr.* indicates the factor by which the backwards error was improved by iterative refinement.

Our experiments seem to indicate that refinement will tend improve to backward error by about one order of magnitude over the results originally achieved in (Gao et al., 2004). The improvement can be quite a bit more pronounced if the original polynomial was within machine precision of being factorizable. In example 9, the factorization found before refinement had backward error worse than 2.37e-1, the backward error of the trivially factorizable polynomial $f(x,y) - f(0,y)$, while that is beaten slightly after refinement. As can be seen by the number of iterations, when $\|noise\| \approx 10^{-1}$ it is still very difficult to get good results.

One can also compute the forward error of each factorization, by which we mean the relative 2-norm coefficient vector distance of a computed approximate factor to the nearest originally chosen factor, before noise was added to the product. For the examples our implementation produced forward errors that are of the same magnitude of the stated backward errors, with the exception of Example 9 where the degrees of the produced approximate factors are 4 and 5, hence the forward error is, in some sense, infinite.

In Table 1:

- Example 1 is from (Nagasaka, 2002) where an approximate factorization with backward error 0.000753084 is also given (this smaller backward error is possible because the perturbed polynomial has increased total degree and thus is not comparable to the error given in Table 1 where total degree is preserved[2]);
- Examples 2 and 3 are from (Sasaki, 2001); Sasaki's algorithm takes 430ms and 2080ms on a SPARC 5 (CPU: microSPARC Π, 70 MHz) and produced backward errors of 10^{-9} and 10^{-5}, respectively;
- Example 4 is from (Corless et al., 2001); the backward error for their approximate factorization is reported as 0.47×10^{-4}, compared to our backward error 2.38×10^{-9} (no timings were reported);
- Example 5 is from (Corless et al., 2002), which is the factorization of an exact polynomial of degree 9 (here their and our backward errors are about the same; no timings were reported);

[2] Note added April 8, 2008: In [Kaltofen, Li, Yang, Zhi 2008, Proc. ISSAC 2008] we have proven that the backward error here for the total degree is optimal.

Table 1
Algorithm performance on benchmarks

Ex.	deg(f_i)	Coeff. error	Backward error	Bkwd err. w/ iter	Time(s)	Iters	Impr.
1	2,3	10^{-2}	1.08e–2	1.02e–3	7.764	7	10.6×
2	5,5	10^{-13}	1.07e–12	1.18e–13	6.813	2	9.0×
3	10,10	10^{-7}	9.95e–7	2.87e–7	157.09	3	3.4×
4	7,8	10^{-9}	1.94e–8	2.38e–9	50.222	16	8.2×
5	3,3,3	0	1.24e–13	6.44e–14	19.517	1	2.4×
6	6,6,10	10^{-5}	1.47e–4	7.24e–6	1329.4	4	20.3×
7	9,7	10^{-4}	2.18e–4	7.07e–5	74.157	4	3.1×
8	4,4,4,4,4	10^{-5}	3.34e–3	8.56e–6	5345.5	4	390.6×
9	3,3,3	10^{-1}	8.03e–1	1.06e–1	33.062	16	7.6×
10	12,7,5	10^{-5}	3.16e–4	8.02e–6	1766.7	4	39.4×
11	12,7,5	10^{-5}	7.77e–5	7.66e–6	2737.6	4	10.2×
12	12,7,5	10^{-3}	5.82e–3	7.66e–4	4288.7	6	7.6×
13	$5,(5)^2$	10^{-5}	6.84e–5	6.52e–6	46.751	3	10.5×
14	$(5)^3,3,(2)^4$	10^{-10}	2.60e–8	3.93e–9	136.39	2	6.6×
15	5,5	10^{-5}	1.55e–5	7.91e–6	559.30	3	2.0×
15a	2,2	10^{-5}	4.62e–13	3.23e–14	2.871	2	14.4×
15b	2,3	10^{-2}	7.44e–4	3.78e–4	6.687	4	2.0×
16	18,18	10^{-6}	4.50e–6	6.65e–7	5945.9	3	6.8×
17	18,18	10^{-6}	4.03e–6	6.61e–7	10348.	3	6.1×
18	6,6	10^{-7}	2.97e–7	5.10e–8	31.829	2	3.8×

- Examples 6 to 13 and 15 to 17 were constructed by choosing factors with random integer coefficients in the range $-5 \le c \le 5$ and then adding a perturbation; for noise we choose a relative tolerance 10^{-e}, then randomly choose a polynomial that has the same degree as the product, 25% as many terms (5% for Example 10 and 99% for Example 17) and coefficients in $[-10^e, 10^e]$; finally, we scale the perturbation so that the relative error is 10^{-e};
- Examples 10, 11 and 12 approximately factorize the same polynomial with perturbations of different noise level and sparseness;
- Example 13 has repeated factors denoted with exponents in the degrees column; it should be noted that the improved factorization found by Gauss-Newton iteration still has a squared factor even though the refinement iteration does not treat the identical factors differently than non-identical factors.
- Example 14 is Zeng's ASFF example in (Zeng and Dayton, 2004); it has non-trivial content and repeated factors. The forward errors of the factors we compute are about 10^{-8}, similar to Zeng's forward error.
- Example 15, 15a, and 15b are polynomials in three variables; 15a is from (Kaltofen, 2000)

and is worked in detail in Section 3.5; Example 15b is from (Huang et al., 2000) where the backward error for their approximate factorization after refinement is reported as 5.72e-4;

- Example 18 is a polynomial with complex coefficients, where the real and imaginary parts of the coefficients of the factors were chosen random integers in $[-5,5]$. Noise was added to the real and imaginary parts of all terms.

The implementation reported in (Gao et al., 2004) also successfully found the approximate factors of four examples, provided by Jan Verschelde, which arise in the engineering of Stewart-Gough platforms (see (Sommese et al., 2004)). The input polynomials in 2 and 3 variables of degree 12 have small absolute coefficient error, 10^{-16}, and have approximate factors of multiplicities 1, 3 and 5. The trivariate approximate factors were computed via sparse numerical interpolation using the techniques of (Giesbrecht et al., 2004, 2006), (which is possible in this example because the forward error in the approximate factor coefficients is near machine precision). The running times, no more than 200 seconds with a backward error of no more than $7.62\cdot 10^{-9}$, appear much faster than what (Sommese et al., 2004) report for their solution, though this is part due to the advantage gained by using the sparse interpolation code reported in (Giesbrecht et al., 2004, 2006).

The Maple implementation and benchmark runs can be found online at
http://www.math.ncsu.edu/~kaltofen/software/appfac/paper07_mws/
or http://www.mmrc.iss.ac.cn/~lzhi/Research/hybrid/appfac/.

4. Concluding Remarks

Wolfang Ruppert's differential forms (Ruppert, 1986, 1999) not only lead to a new exact factorization algorithm (Gao, 2003), they yield a formulation as a nearest structured singular matrix problem in the approximate setting. That setting then allows the application of several methods from numerical analysis, such as singular value decomposition (SVD), which has been already applied in the area of hybrid symbolic/numeric algorithms in (Corless et al., 1995; Emiris et al., 1997; Gianni et al., 1998; Zeng, 2003). Here we have shown that the SVD-based approach followed by Gauss-Newton iteration can efficiently produce approximate factorization, which on our reversely engineered benchmark examples have a backward error as near as the introduced coefficient noise. Recently, structured least norm algorithms (Park et al., 1999; Lemmerling et al., 2000) have been successfully applied to hybrid symbolic/numeric algorithms (Kaltofen et al., 2007; Botting et al., 2005; Kaltofen et al., 2006) and we can report that they are a viable alternative to the SVD/Gauss-Newton approach.

For polynomials with many variables, the arising structured totals least norm problems have a very high dimension. One approach, already mentioned in section 3.6, is to use sparse numerical interplolation (Giesbrecht et al., 2006) of the bi- or tri-variate factor images. Another is the use of fast structured solvers analogous to the theory of Toeplitz-like matrices (Pan, 2001; Olshevsky, 2003). For the univariate approximate GCD problem, results are reported in (Zhi, 2003; Li et al., 2005). We hope to develop displacement operators for generalized Sylvester matrices and the Ruppert matrices in the near future.

Acknowledgements: We thank Jan Verschelde for files of the examples in (Sommese et al., 2004) and Wen-shin Lee for help with sparse interpolation and the Maple code to (Giesbrecht et al., 2004, 2006). The first and second authors were supported in part by the National Science Foundation under Grants CCF-0305314, OISE-0456285 and CCF-0514585. The third and fourth authors were supported by NKBRPC (2004CB318000) and the Chinese National Natural Science Foundation under Grant 10401035.

References

Botting, B., Giesbrecht, M., May, J., 2005. Using Riemannian SVD for problems in approximate algebra. In: Wang and Zhi (2005), pp. 209–219, distributed at the Workshop in Xi'an, China, July 19–21.

Chèze, G., 2004. Absolute polynomial factorization in two variables and the knapsack problem. In: Gutierrez (2004), pp. 87–94.

Corless, R. M., Galligo, A., Kotsireas, I. S., Watt, S. M., 2002. A geometric-numeric algorithm for absolute factorization of multivariate polynomials. In: Mora (2002), pp. 37–45.

Corless, R. M., Gianni, P. M., Trager, B. M., Watt, S. M., 1995. The singular value decomposition for polynomial systems. In: Levelt, A. H. M. (Ed.), Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC'95. ACM Press, New York, N. Y., pp. 96–103.

Corless, R. M., Giesbrecht, M. W., van Hoeij, M., Kotsireas, I. S., Watt, S. M., 2001. Towards factoring bivariate approximate polynomials. In: Mourrain (2001), pp. 85–92.

Dumas, J.-G. (Ed.), 2006. ISSAC MMVI Proc. 2006 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y.

Emiris, I. Z., Galligo, A., Lombardi, H., May 1997. Certified approximate univariate GCDs. J. Pure Applied Algebra 117 & 118, 229–251, special Issue on Algorithms for Algebra.

Galligo, A., Rupprecht, D., 2001. Semi-numerical determination of irreducible branches of a reduced space curve. In: Mourrain (2001), pp. 137–142.

Galligo, A., Rupprecht, D., 2002. Irreducible decomposition of curves. J. Symbolic Comput. 33 (5), 661–677.

Galligo, A., Watt, S., 1997. A numerical absolute primality test for bivariate polynomials. In: Küchlin, W. (Ed.), ISSAC 97 Proc. 1997 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y., pp. 217–224.

Gao, S., 2003. Factoring multivariate polynomials via partial differential equations. Math. Comput. 72 (242), 801–822.

Gao, S., Kaltofen, E., May, J. P., Yang, Z., Zhi, L., 2004. Approximate factorization of multivariate polynomials via differential equations. In: Gutierrez (2004), pp. 167–174, ACM SIGSAM's ISSAC 2004 Distinguished Student Author Award (May and Yang).

Gianni, P., Seppälä, M., Silhol, R., Trager, B., 1998. Riemann surfaces, plane algebraic curves and their period matrices. J. Symbolic Comput. 26 (6), 789–803.

Giesbrecht, M., Labahn, G., Lee, W., 2004. Symbolic-numeric sparse interpolation of multivariate polynomials (extended abstract). In: Proc. Ninth Rhine Workshop on Computer Algebra (RWCA'04), University of Nijmegen, the Netherlands. pp. 127–139.

Giesbrecht, M., Labahn, G., Lee, W., 2006. Symbolic-numeric sparse interpolation of multivariate polynomials. In: Dumas (2006), pp. 116–123.

Gutierrez, J. (Ed.), 2004. ISSAC 2004 Proc. 2004 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y.

Hitz, M. A., Kaltofen, E., Lakshman Y. N., 1999. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In: Dooley, S. (Ed.), Proc. 1999 Internat. Symp. Symbolic Algebraic Comput. (ISSAC'99). ACM Press, New York, N. Y., pp. 205–212.

Huang, Y., Stetter, H. J., Wu, W., Zhi, L., 2000. Pseudofactors of multivariate polynomials. In: Traverso, C. (Ed.), Internat. Symp. Symbolic Algebraic Comput. ISSAC 2000 Proc. 2000 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y., pp. 161–168.

Kaltofen, E., 1985. Fast parallel absolute irreducibility testing. J. Symbolic Comput. 1 (1), 57–67, misprint corrections: *J. Symbolic Comput.* vol. 9, p. 320 (1989).

Kaltofen, E., 2000. Challenges of symbolic computation my favorite open problems. J. Symbolic Comput. 29 (6), 891–919, with an additional open problem by R. M. Corless and D. J. Jeffrey.

Kaltofen, E., May, J., 2003. On approximate irreducibility of polynomials in several variables. In: Sendra (2003), pp. 161–168.

Kaltofen, E., Yang, Z., Zhi, L., 2006. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In: Dumas (2006), pp. 169–176, full version, 21 pages. Submitted, December 2006.

Kaltofen, E., Yang, Z., Zhi, L., 2007. Structured low rank approximation of a Sylvester matrix. In: Wang, D., Zhi, L. (Eds.), Symbolic-Numeric Computation. Trends in Mathematics. Birkhäuser Verlag, Basel, Switzerland, pp. 69–83, preliminary version in Wang and Zhi (2005), pp. 188–201.

Kelley, C. T., 1999. Iterative Methods for Optimization. No. 18 in Frontiers in Applied Mathematics. SIAM, Philadelphia.
URL http://www.siam.org/books/textbooks/fr18_book.pdf

Lemmerling, P., Mastronardi, N., Van Huffel, S., 2000. Fast algorithm for solving the Hankel/Toeplitz Structured Total Least Squares problem. Numerical Algorithms 23, 371–392.

Li, B., Yang, Z., Zhi, L., 2005. Fast low rank approximation of a Sylvester matrix by structured total least norm. J. JSSAC (Japan Society for Symbolic and Algebraic Computation) 11 (3,4), 165–174.

May, J., 2005. Approximate factorization of polynomials in many variables and other problems in approximate algebra via singular value decomposition methods. Ph.D. thesis, North Carolina State University.

Mora, T. (Ed.), 2002. ISSAC 2002 Proc. 2002 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y.

Mourrain, B. (Ed.), 2001. ISSAC 2001 Proc. 2001 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y.

Nagasaka, K., 2002. Towards certified irreducibility testing of bivariate approximate polynomials. In: Mora (2002), pp. 192–199.

Nagasaka, K., 2005. Towards more accurate separation bounds of empirical polynomials II. In: Ganzha, V. G., Mayr, E. W., Vorozhtsov, E. V. (Eds.), Computer Algebra in Scientific Computing, 8th International Workshop, CASC 2005, Kalamata, Greece, September 12-16, 2005, Proceedings. Vol. 3718 of Lecture Notes Comput. Sci. Springer Verlag, Heidelberg, Germany, pp. 318–329.

Olshevsky, V., 2003. Pivoting on structured matrices and rational tangential interpolation. In: Olshevsky, V. (Ed.), Fast Algorithms for Structured Matrices: Theory and Applications. No. CONM/323 in Contemporary Mathematics. AMS, pp. 1–75.

Pan, V. Y., 2001. Structured Matrices and Polynomials: Unified Superfast Algorithms. Birkhäuser, Boston.

Park, H., Zhang, L., Rosen, J. B., 1999. Low rank approximation of a Hankel matrix by structured total least norm. BIT 39 (4), 757–779.

Ruppert, W. M., 1986. Reduzibilität ebener Kurven. J. reine angew. Math. 369, 167–191.

Ruppert, W. M., 1999. Reducibility of polynomials $f(x, y)$ modulo p. J. Number Theory 77, 62–70.

Rupprecht, D., 2004. Semi-numerical absolute factorization of polynomials with integer coefficients. J. Symbolic Comput. 37 (5), 557–574.

Sasaki, T., 2001. Approximate multivariate polynomial factorization based on zero-sum relations. In: Mourrain (2001), pp. 284–291.

Sasaki, T., Saito, T., Hilano, T., Oct. 1992. Analysis of approximate factorization algorithm I. Japan J. of Industrial and Applied Mathem. 9 (3), 351–368.

Sasaki, T., Suzuki, M., Kolář, M., Sasaki, M., Oct. 1991. Approximate factorization of multivariate polynomials and absolute irreducibility testing. Japan J. of Industrial and Applied Mathem. 8 (3), 357–375.

Sendra, J. R. (Ed.), 2003. ISSAC 2003 Proc. 2003 Internat. Symp. Symbolic Algebraic Comput. ACM Press, New York, N. Y.

Sommese, A. J., Verschelde, J., Wampler, C. W., 2004. Numerical factorization of multivariate complex polynomials. Theoretical Comput. Sci. 315 (2–3), 651–669, special issue on Algebraic and Numerical Algorithms.

Wang, D., Zhi, L. (Eds.), 2005. Internat. Workshop on Symbolic-Numeric Comput. SNC 2005 Proc. Distributed at the Workshop in Xi'an, China, July 19–21.

Zeng, Z., 2003. A method computing multiple roots of inexact polynomials. In: Sendra (2003), pp. 266–272.

Zeng, Z., Dayton, B. H., 2004. The approximate GCD of inexact polynomials part II: a multivariate algorithm. In: Gutierrez (2004), pp. 320–327.

Zhi, L., 2003. Displacement structure in computing approximate GCD of univariate polynomials. In: Li, Z., Sit, W. (Eds.), Proc. Sixth Asian Symposium on Computer Mathematics (ASCM 2003). Vol. 10 of

Lecture Notes Series on Computing. World Scientific, Singapore, pp. 288–298.