Towards Factoring Bivariate Approximate Polynomials*

Robert M. Corless† Mark W. Giesbrecht† Mark van Hoeij‡ Ilias S. Kotsireas†
Stephen M. Watt†

†The Ontario Research Centre for Computer Algebra
University of Western Ontario
London ON, N6A 5B7 Canada

‡Department of Mathematics
Florida State University
Tallahassee, FL 32306-4510, USA

ABSTRACT

A new algorithm is presented for factoring bivariate approximate polynomials over $\mathbb{C}[x, y]$. Given a particular polynomial, the method constructs a nearby composite polynomial, if one exists, and its irreducible factors. Subject to a conjecture, the time to produce the factors is polynomial in the degree of the problem. This method has been implemented in Maple, and has been demonstrated to be efficient and numerically robust.

1. INTRODUCTION

Over the past several years symbolic-numeric algorithms for polynomials have been studied by a number of authors. Methods have been presented to compute greatest common divisors of approximate polynomials [2, 7, 18], compute functional decompositions [8], test primality [10], find zeros of multivariate systems [6, 7] and solve other problems. An important problem of this family which has not been sufficiently treated is the factoring of approximate polynomials. Earlier methods for factoring multivariate approximate polynomials are beset with exponential complexity: the paper [10] considers point combinations, an exponential search, while the paper [13] uses an optimization method exponential in the degree of the factor recovered. While exponential methods may be the best in practice for problems of bounded size, we are led to seek more efficient algorithms.

The paper [14], while more efficient than previous treatments (and more importantly, more numerically stable), is still of complexity exponential in the degree of the input.

The papers [15, 23, 21, 22, 20] discuss another interesting algorithm based on zero-sum identities of power-series solutions of $f(x, y) = 0$. In [22] this algorithm is shown to be quite general, also being applicable to the problem of factoring over algebraic number fields and algebraic function fields. This algorithm is also numerically stable and of polynomial complexity.

In this paper we present a polynomial-time method to factor approximate bivariate polynomials. The bivariate case captures essential issues of the more general problem of factoring multivariate polynomials. As in [7], we adopt a backward error analysis point of view.

Problem:
Given $f \in \mathbb{C}[x, y]$ of total degree n, and $\varepsilon \in \mathbb{R}$, find g and $h \in \mathbb{C}[x, y]$ such that $g \times h = f + \Delta f$, where $\deg g, \deg h > 0, ||\Delta f|| < \varepsilon$ and $|| \cdot ||$ denotes an appropriate polynomial norm. Here we use the 2-norm.

This is related to an open problem first posed by Kaltofen in 1992 [16] and posed as a challenge at ECCAD in 1998 (now published in [17]). We do not completely solve the problem, however: the method of this paper are guaranteed to find factors of an approximate polynomial only if the given polynomial is "sufficiently close" to being factorable. We do not give a *priori* bounds for what "sufficiently close" means.

Further, we assume that f is "approximately square-free", that is, all polynomials in a neighbourhood of f are square-free. This can be quickly determined in practice by taking a random complex floating-point value for x and using the fast algorithm of [2] to certify that the approximate GCD of $f(x, y)$ and $f_y(x, y)$ is 1. If this is not the case, then an algorithm to remove the multivariate approximate GCD such as described in [7] must be used.

The principal contributions of this paper are:

- A method to compute points on the Riemann surface of the factor of $f + \Delta f$ identified by an initial point. These points can then be used to recover the coefficients of the factor by the numerical implicitization method of [9].
- A perturbation analysis of the conditioning of the Riemann surfaces of factors of bivariate polynomials.

*Supported in part by the Natural Sciences and Engineering Research Council of Canada, the Ontario Research and Development Challenge Fund, the ESPRIT Long Term Research project FRISCO, and Waterloo Maple Inc.

ISSAC 2001, 7/01, Ontario, Canada
©2001 ACM 1-58113-417-7/ 01/ 0007 $5.00

- A fast method for bivariate approximate polynomial division.

Together, these contributions give us an efficient method to construct factors of bivariate approximate polynomials.

The factoring method is based on two observations: first, given a polynomial and a point on the Riemann surface of one of its components, it is possible to integrate locally to determine sufficient information to reconstruct the Riemann surface of the component in a stable fashion; and second, if an irreducible polynomial f is near a composite polynomial $f+\Delta f$, then the well-conditioned parts of the Riemann surface for f will be close to the union of the Riemann surfaces for each factor of $f + \Delta f$.

This leads to an algorithm with the following outline:

1. [Initialization] Compute a point (x_0, y_0) on $f(x, y) = 0$, well-conditioned in the sense that $||\nabla f||$ is not close to zero. Choosing a random x_0 will do this with high probability.

2. [Parameterization] Use a numerical continuation method to follow the Riemann surface away from (x_0, y_0) to discover a well-conditioned region $\mathcal{R}$.

3. [Implicitization] For each $r = 1, ..., n-1$ (or up to $n/2$ if investigations of several roots are done in parallel) attempt to construct a candidate factor g of degree r approximating f on $\mathcal{R}$. Using the methods of [9], this amounts to finding an approximate null vector of a numerical matrix. If there is no such vector, then there is no factor of total degree r. If there is more than one such null vector, then we have a basis for the *implicit ideal*, and a factor can be reconstructed from this.

4. [Division] When a suitable g is found, perform an approximate division to find an $h \in \mathbb{C}[x, y]$ yielding $f + \Delta f$ which factors as g times h.

5. [Refinement] As in [8].

The remainder of the paper is organized as follows. Section 2 presents the algorithm in the setting of exact polynomials, and then elaborates the issues which arise in the approximate setting. Section 3 sketches theorems justifying the algorithm. Section 4 sketches the complexity of the algorithm. Section 5 discusses the Maple implementations of the algorithm, and gives examples of its use.

2. THE RIEMANN SURFACE APPROXIMATE FACTORING ALGORITHM

In this section we first present our algorithm in the setting of exact polynomials. We then elaborate on the issues which arise when the coefficients of our polynomials are appoximate.

The factoring proceeds as follows. Suppose that $f(x, y) = g(x, y)h(x, y)$.

First, a random point (x_0, y_0) on the variety of $f(x, y) = 0$ is found. Say, without loss of generality, that this point is on the Riemann surface for $g(x, y) = 0$.

Next, we use a continuation method to find points on a path $(x(s), y(s))$ such that for every $0 \le s \le s_1$, $f(x(s), y(s)) = 0$, and moreover that we do not go through a singular point, where $\nabla f = 0$. This last restriction is because if $f(x, y) = g(x, y)h(x, y)$, then a short computation shows that $\nabla f = [0, 0]$ at any point (x, y) such that $g(x, y) = h(x, y) = 0$. By avoiding such points, then, we remain on the Riemann surface of the factor that we started on, namely g.

Next, we sample enough points on the Riemann surface for g to recover its coefficients by interpolation, as in [9].

Computation of h is by bivariate division.

In the case of approximate polynomials that are close to polynomials that are exactly factorable the following differences from the exact case arise.

- Exact singularity of f, that is $\nabla f = 0$, is no longer necessary on a path from the Riemann surface of one "factor" to the other.

- Riemann surfaces of a given degree may often be locally approximated surprisingly well by Riemann surfaces of a lower degree. This means that we must try to get a "global fit" to the Riemann surface of a factor.

- The linear system (null vector problem) to find a factor may no longer have an exact solution—we may need to find an approximate null vector.

- The computed points will not be exactly on the Riemann surface for $f(x, y) = 0$, but instead near to it.

These considerations, and others, are addressed in the algorithm that follows.

2.1 Initialization

We choose a random x_0, and numerically solve the univariate problem $f(x_0, y) = 0$ for one such y. Call this solution y_0. It will in general provide only a pseudozero, with $f(x_0, y_0) = \varrho$ for some small residual ϱ. Since the point x_0 was random, then with probability 1 we are away from a singularity of f and therefore within $O(\varrho)$ of a true root. By choosing the precision we work with we may take ϱ as small as we like. Note that this is independent of any inherent errors in the data (that is, coefficients of f).

2.2 Numerical parameterization of $f(x, y) = 0$

For us, a *path* or parameterization P (of the variety $f(x, y) = 0$) is the range of a smooth (C^2) function $s : \mathbb{R} \to P \in \mathbb{C} \times \mathbb{C}$ such that $(x(s), y(s)) \in P$ if and only if $f(x(s), y(s)) = 0$. We interpret algorithms to find paths as methods for solving the differential equations

$$\frac{d}{ds} f(x(s), y(s)) = 0 \qquad (1)$$

starting at some given initial point $x(0) = x_0$, $y(0) = y_0$, and proceeding in a prescribed direction, which may depend on x and y.

This is equivalent to

$$f_x\dot{x} + f_y\dot{y} = 0 \tag{2}$$

(using $\dot{x}$ to mean dx/ds).

In practice, we follow the real and imaginary parts of x and y separately. Write $x = x_r + ix_i$ and $y = y_r + iy_i$ and

$$f(x,y) = u(x_r, x_i, y_r, y_i) + iv(x_r, x_i, y_r, y_i)\,, \tag{3}$$

and remember that $f(x,y)$ is separately analytic in each of x and y. Then equation (1) becomes

$$\begin{aligned} u_{x_r}\dot{x}_r + u_{x_i}\dot{x}_i + u_{y_r}\dot{y}_r + u_{y_i}\dot{y}_i &= 0 \\ v_{x_r}\dot{x}_r + v_{x_i}\dot{x}_i + v_{y_r}\dot{y}_r + v_{y_i}\dot{y}_i &= 0\,, \end{aligned} \tag{4}$$

or, using the Cauchy-Riemann conditions for analyticity (in this notation $u_{x_r} = v_{x_i}$ and $u_{x_i} = -v_{x_r}$ and analogously for y), we deduce that every solution solves an equation of the following form, using $\dot{w} = F(w;\nu)$, where we define $w = [x_r, x_i, y_r, y_i]^T$:

$$\begin{bmatrix} \dot{x}_r \\ \dot{x}_i \\ \dot{y}_r \\ \dot{y}_i \end{bmatrix} = F(w;\nu) = \alpha \begin{bmatrix} u_{y_r} & u_{y_i} \\ -u_{y_i} & u_{y_r} \\ -u_{x_r} & -u_{x_i} \\ u_{x_i} & -u_{x_r} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix}, \tag{5}$$

for some scalar $\alpha(s)$ and unit vector $[\nu_1(s), \nu_2(s)]$. In complex form, this is equivalent to $\dot{x} = \alpha\exp(i\theta)f_y$, and $\dot{y} = -\alpha\exp(i\theta)f_x$, where $[\nu_1, \nu_2] = [\cos\theta, \sin\theta]$.

For geometric reasons we choose α so as to ensure that s is arc length: $\alpha = (|f_x|^2 + |f_y|^2)^{-1/2} = (u_{x_r}^2 + u_{x_i}^2 + u_{y_r}^2 + u_{y_i}^2)^{-1/2}$. We shall describe below how to choose the unit vector ν (depending on x and y) in a manner that preserves an essential property, namely that the path remains on the variety of the same factor of f as the initial point.

For such few curves as can be analytically parameterized, symbolic solution of these differential equations may be possible. However, in general, we must use numerical methods.

In principle, we can use any numerical method to solve equations (5). Natural candidates are geometric integration methods on manifolds, or Projected Implicit Runge-Kutta methods [1], which are appropriate for differential-algebraic equations (adding back in the constraint $f(x,y) = 0$ makes these differential equations into a DAE), or Taylor Series Methods for high accuracy, or (of especial interest in this context) validated interval arithmetic methods such as are described in [3]. These latter methods provide a computer assisted proof that the computed points are accurate.

It is important to note that, in principle, these differential equations can be solved to arbitrary accuracy, and can locate singular points (path crossings, where $\alpha = \infty$) also to arbitrary accuracy.

We choose instead to use the NODES package in Maple for solving differential equations numerically, followed by a Newton refinement (to the desired precision) of selected points [24]. For our purposes, note that the package uses hardware floats in Maple, and is thus efficient. Its heuristics for local error stepsize control enable an accurate solution. Failure of the heuristics, while theoretically possible, is unlikely.

A procedure `poly2proc` was written to generate a procedure that describes the differential equations (5). This procedure takes as input the approximate polynomial to be factored. Its output is a Maple procedure suitable to be used by the NODES package, but it could equally well generate FORTRAN or C code for use with other numerical integrators.

The end result of this computation is a *numerical parameterization* of a segment of the variety $f(x,y) = 0$. In detail, what NODES produces is a piecewise polynomial approximation to $(x(s), y(s))$ over the range of integration. This piecewise polynomial parameterization can then be refined at any point using Newton iteration on $f(x,y) = 0$. By piecing together several of these path segments, we build up a numerical parameterization of a patch of the Riemann surface defined by $f(x,y) = 0$.

Keeping on the same component.
Differentiating α^{-2} and using (5) gives $\dot{\alpha} = -2\alpha^3(A\nu_1 + B\nu_2)$ where A and B are polynomials in derivatives of u. Choosing (ν_1, ν_2) proportional to (A, B) gives the "most negative" $\dot{\alpha}$. Choosing instead $(-B, A)$ gives $\dot{\alpha} = 0$.

In section 3.1 we analyze this strategy of integrating our differential equations in a direction such that $\dot{\alpha} \le 0$, and argue why this guarantees (if f is close enough to a factorable polynomial) that our computed points are all on the Riemann surface for the same approximate factor g of f. This is a key property of the method; by providing this guarantee, we avoid the potentially exponential cost of deciding which points are on each factor.

2.3 Newton refinement

Suppose $f(x_k, y_k) = \varepsilon \approx 0$ and we wish to find complex (x_{k+1}, y_{k+1}) closer to the Riemann surface defined by $f(x,y) = 0$. Linearization gives

$$0 \approx f(x_k, y_k) + f_x(x_k, y_k)\Delta x + f_y(x_k, y_k)\Delta y\,. \tag{6}$$

This equation has an infinite number of solutions. The asymptotically optimal (smallest) step $(\Delta x, \Delta y)$ will occur if we choose $(\Delta x, \Delta y)$ orthogonal to the surface $f(x,y) = \varepsilon$, and thus in the direction of the greatest variation of f. This is equivalent to imposing

$$(\Delta x, \Delta y) = \beta\,(\overline{f_x}(x_k, y_k), \overline{f_y}(x_k, y_k)) \tag{7}$$

for some scalar β. Using this in (6) gives $\beta = -\alpha^2 f(x_k, y_k)$ or

$$(\Delta x, \Delta y) = -\alpha^2(\overline{f_x}, \overline{f_y})\,f\,. \tag{8}$$

This gives us an explicit formula for a Newton refinement step.

2.4 Implicitization

We use the numerical implicitization procedure described in [9] to recover the coefficients of the factors of the polynomial from the sampled points on the Riemann surface for the

factor. The implicitization procedure runs as follows. First, a candidate set of monomials in the support of the factor is selected. We can impose sparsity on the factor at this point by choosing a restricted set of monomials. If we do not know the sparsity pattern, then we use the complete set of monomials of degree less than or equal to d. This gives $g(x,y)$ with support $\{1, x, y, x^2, xy, y^2, \ldots, y^d\}$. Next the sample points in the numerical parameterization are selected. Here we observe that for reliable implicitization, points from a substantial fraction of the Riemann surface must be selected (see section 3); in [11] *generic* points are used. Then a matrix M with rows labelled by the support is constructed. We take $1 \leq k \leq 2\binom{d+2}{2}$ or twice as many samples as strictly necessary. Then any approximate null vector of M corresponds to a polynomial in the implicit ideal. In practice we take the singular vector of M corresponding to the smallest singular value.

2.5 Approximate Polynomial Division

In this subsection we present an algorithm for the division of approximate polynomials: given $f, g \in \mathbb{C}[x,y]$, find $h \in \mathbb{C}[x,y]$ of total degree $\deg f - \deg g$ such that $\|f - gh\|$ is minimized. The proposed algorithm requires $O(t^2 n^2 \log n)$ floating point operations, where f and h have degrees n and t respectively. In other words, it is quadratic time in the dense representation of the input, which has $O(n^2)$ terms.

We formulate this as a structured least squares problem. Let f, g, h have respective total degrees n, r, t respectively. Then

$$\begin{aligned} f(x,y) &= \sum_{0\leq i\leq n} f_i(y)x^i, & f_i(y) &= \sum_{0\leq j\leq n-i} f_{ij}y^j \\ g(x,y) &= \sum_{0\leq i\leq r} g_i(y)x^i, & g_i(y) &= \sum_{0\leq j\leq r-i} g_{ij}y^j \\ h(x,y) &= \sum_{0\leq i\leq t} h_i(y)x^i, & h_i(y) &= \sum_{0\leq j\leq t-i} h_{ij}y^j . \end{aligned}$$

We are trying to find the h which minimizes $\|f - gh\|$. For $0 \leq i \leq r$, define

$$G_i^{(k)} = \begin{bmatrix} g_{i0} & & \\ \vdots & \ddots & \\ \vdots & & g_{i0} \\ g_{i,r-i} & & \\ & \ddots & \vdots \\ & & g_{i,r-i} \end{bmatrix} \in \mathbb{C}^{(r-i+1+k)\times(k+1)}$$

Clearly

$$G_i^{(k)} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_k \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{r-i+k} \end{pmatrix}$$

is equivalent to

$$g_i(y)(\alpha_0 + \cdots + \alpha_k y^k) = \beta_0 + \beta_1 y + \cdots + \beta_{r-i+k} y^{r-i+k}.$$

Multiplication by g is given by the block matrix

$$G = \begin{bmatrix} G_0^{(t)} & & & \\ \vdots & G_0^{(t-1)} & & \\ G_r^{(t)} & & \ddots & \\ & G_r^{(t-1)} & & G_0^{(0)} \\ & & \ddots & \vdots \\ & & & G_r^{(0)} \end{bmatrix} \in \mathbb{C}^{\frac{(n+1)(n+2)}{2} \times \frac{(t+1)(t+2)}{2}},$$

and

$$\begin{aligned} &G(h_{00}, \ldots, h_{0t}, h_{10}, \ldots, h_{1,t-1}, \ldots, h_{t0})^t \\ &= (f_{00}, \ldots, f_{0n}, f_{10}, \ldots, f_{1,n-1}, \ldots, f_{n0})^t \end{aligned} .$$

Note that the blocks in G are not of uniform size. The $h \in \mathbb{C}[x,y]$ which minimizes $\|f - gh\|$ corresponds to the least squares solution to $G\mathbf{h} = \mathbf{f}$. This immediately gives an $O(n^2 r^4)$ solution (i.e., about cubic in the input size) by simply solving the least squares problem using an appropriate numerical technique (see, e.g., [12]).

A faster solution is derived as follows. The desired $\mathbf{h}$ is the unique solution to the non-singular, Hermitian system $G^*G\mathbf{h} = G^*\mathbf{f}$. We can solve this quickly with a matrix-free method if we can multiply vectors by G and G^* quickly.

Note however that the condition number of G^*G is the square of the condition number of G, and this is known to make the algorithm less numerically stable than a QR factoring of G would be, but note that the sensitivity of the problem to changes in the data depends on the square of the condition number of G anyway [4]. In addition, the numerical difficulty in using G^*G to solve least-squares problems can be attributed to "killing small data", which may even be beneficial in our case. Thus, for this special application, we expect that the benefits of the speed of solution using G^*G will offset the cost of its lower stability.

Multiplication of a vector on the left by G can be done simply by multiplication by g in $\mathbb{C}[x,y]$ and hence can be accomplished with $O(n^2 \log n)$ floating point operations by means of a Fast Fourier Transform. More specifically, assume we wish to compute $\mathbf{q} = G\mathbf{p}$, where $\mathbf{p}$ is the vector of coefficients of some $p \in \mathbb{C}[x,y]$. Choose $\omega \in \mathbb{C}$ to be an $(n+1)^{\text{th}}$ primitive root of unity and compute the product $q_k(x) = g(x,\omega^k)p(x,\omega^k)$ for $k = 0 \ldots n$. The coefficients of $q(x,y) = g(x,y)p(x,y)$ can be recovered from $q_0, \ldots, q_k$ through interpolation. The total cost is $O(n^2 \log n)$. Moreover, since we are only evaluating and interpolating at roots of unity, the process is numerically stable.

Similarly, we can multiply by G^* with this same cost by noting that the coefficients of $G^*\mathbf{v}$ are the negative degree coefficients of $\overline{g}(1/x, 1/y)v(x,y)$, where $\overline{g}$ is the conjugate of g and $v(x,y)$ is the polynomial corresponding to the coefficient vector $\mathbf{v}$.

Given a fast way to evaluate G^*G at a vector, we can use a matrix-free iterative method such as Lanczos or Conjugate Gradient to solve the desired least squares problem with

$O(t^2)$ applications of G^*G, for a total cost of $O(t^2n^2 \log n)$ floating point operations.

3. JUSTIFICATION

3.1 Conditioning

We define the condition number α of a point on the Riemann surface, and show why this definition is useful.

With each point on the variety $f(x,y) = 0$, we may associate a number α, defined in the previous section. We show here why this defines a "condition number". Consider the Newton step given by (8). It follows that the size of the step is

$$\|(\Delta x, \Delta y)\| = \alpha^2(|\overline{f_x}|^2 + |\overline{f_y}|^2)^{1/2}|f| = \alpha|f| \,. \qquad (9)$$

We thus conclude that the Riemann surface for $f+\Delta f = 0$ is asymptotically at a distance $\alpha|\Delta f(x,y)|$ from the Riemann surface for $f = 0$, as $|\Delta f(x,y)| \to 0$. This shows that α is a condition number. Note that $\alpha = \infty$ at any place where $g(x,y) = h(x,y) = 0$ if $f = gh$.

3.2 Integration on one component only.

In this section, we look at what happens when a factorable polynomial is perturbed, from the point of view of condition number α. We argue that for small perturbations, for factors g and h that have simple contact, the strategy of integrating so that $\dot{\alpha} \leq 0$ guarantees that we stay on the Riemann surface of the factor which we started on.

Consider $f_\varepsilon(x,y) = f_0(x,y) + \varepsilon f_1(x,y)$, where $f_0(x,y) = g(x,y)h(x,y)$. Look at the points of intersection of the Riemann surfaces of g and h, namely the simultaneous zeros of $g(x,y) = h(x,y) = 0$. Generically, these will consist of a finite number of points, because f is assumed to be squarefree.. A straightforward computation shows that $f_x = g_x h + g h_x$ and $f_y = g_y h + g h_y$ and thus $f_x = f_y = 0$ at simultaneous zeros of g and h; therefore $\alpha = \infty$ at any such points. We also see that $\alpha = \infty$ at points where either (g, g_x, g_y) or (h, h_x, h_y) are each all zero.

If we now consider the perturbed polynomial $f_\varepsilon = gh + \varepsilon f_1$, then we have for $(\varepsilon > 0)$

$$\begin{aligned} f_{\varepsilon,x} &= g_x h + g h_x + \varepsilon f_{1,x} \\ f_{\varepsilon,y} &= g_y h + g h_y + \varepsilon f_{1,y} \end{aligned}$$

and hence at a common zero of g and h (which may not be on the variety of f_ε to be sure, but for small enough ε will be close) we have

$$\alpha_\varepsilon = \varepsilon^{-1} \left(\overline{f}_{1,x} f_{1,x} + \overline{f}_{1,y} f_{1,y} \right)^{-1/2} .$$

Thus we see that a nearby simple common zero of g and h induces a condition number of $O(\varepsilon^{-1})$ in a perturbed polynomial.

Notice that in order for a path to pass from one component $g(x,y) = 0$ of the Riemann surface to the other, $h(x,y) = 0$, it is necessary to pass through the simultaneous zero, in the factorable case. In the nearly factorable case, it is only necessary to pass through a region where α is "large". Since we do not *a priori* know ε, we must therefore avoid *any* increase in α in our integration. This is sufficient to guarantee that we remain on the same component, provided that ε is small enough.

Note that this is *not* a necessary condition, as it is entirely possible that singularities in g alone exhibit the same behaviour, in which case allowing α to increase is harmless and we would remain on g. This can be verified *a posteriori*, but not during the computation; we therefore exclude this possibility by requiring $\dot{\alpha} \leq 0$ along a path.

For a given problem with a non-infinitesimal perturbation, the condition number (which is based on linear infinitesimal perturbation theory) may not tell us where the bad region is. Therefore the strategy of this section only works for polynomials "close enough" to a factorable polynomial f_0. The fact that it is "close enough" can at present be verified only *a posteriori*, by computing the other factor h by approximate division (see section 2.5) and computing the residual $f - gh$.

4. EFFICIENCY CONSIDERATIONS

The cost of initialization is the cost of solving a degree n univariate polynomial. By [19] this has polynomial complexity.

Numerical solution of initial value problems by a method of order p that chooses stepsizes according to the usual algorithms can be modelled as one that equidistributes the stepsize according to $C_n h_n^p = \varepsilon$, where ε is the constant tolerance input by the user [5]. This leads to a mean stepsize related to a Hölder mean M of the pth derivative of the solution: $h_{\text{mean}} = (\varepsilon/M)^{1/p}$. The mean M is a characteristic of the Riemann surface that we are investigating, and thus depends on the coefficients of the bivariate approximate polynomial f, but because of the pth root this dependence is weak. Assuming that step rejections are rare, the cost per step of the method is essentially fixed (and usually measured in terms of the number of function evaluations, which here cost $O(n)$ floating-point arithmetic operations). The cost of integrating a path of fixed length L is therefore $O(nL(M/\varepsilon)^{1/p})$. The question of how big L should be remains open. We conjecture that an L large enough to ensure coverage of a substantial portion of the Riemann surface for the factor is polynomially bounded by the degree of the factor.

The parameterization algorithm has two phases: numerical integration 'downhill' in α, followed by a fixed number of integrations at constant α. The downhill integration continues until α is sufficiently close to zero. Inspection of the differential equations (5) shows that α decays at least exponentially along such a path, and therefore the number of steps taken is bounded. In the second phase, analysis shows that paths with constant α are closed (think of taking contours of a Riemann surface according to given function of x and y), and so it is useful to add 'event handling' to the code to detect when we return to the starting point. This means that the length of such paths is not fixed, but depends on the local characteristics of the Riemann surface. The length of such paths is, however, bounded for a given Riemann surface because the region of interest for integration contains all singularities and α goes to zero as x and y go to infinity. This bound depends polynomially on the degree of f. This follows from the above conjecture.

Therefore the cost of computing points accurate to $O(\varepsilon)$ on the Riemann surface for f is polynomial in the degree of f. Refinement of computed points to accuracy better than $O(\varepsilon)$ on the Riemann surface, by Newton's method, is likewise of polynomial cost, because each Newton step is of polynomial cost.

Finally, this means that detecting irreducibility is likewise of polynomial cost, though the methods of [10] are likely faster in practice.

5. EXAMPLES

5.1 A simple example

Consider the exactly factorable polynomial $p(x,y) = y^2 - x^4 = (y-x^2)(y+x^2)$ and perturb it slightly to get $\tilde{p}(x,y) = f = y^2 - x^4 + \epsilon x^2$, for $\epsilon = 0.01$. Starting from $\tilde{p}(x,y)$ we want to be able to recover a factor of the nearby exactly factorable polynomial $p(x,y)$. We will work with 15 digits of precision in Maple to use hardware floating point. First we choose a random (complex) point x_0 and we solve the (quadratic) equation $f(x_0,y) = 0$. We get a point (x_0,y_0) on the curve defined by f. Then we integrate the differential equations (5) (which are generated automatically by `poly2proc`) in an interval where the condition number remains small. We integrate the characteristic differential equations for the Riemann surface of $f(x,y) = 0$, going downhill in α. This produces a set of points on the Riemann surface of $f(x,y) = 0$. A dense factor of degree 2 in x,y has a support of six monomials. We thus choose to sample 12 points to approximate this factor. We use the SVD to find an approximate null vector. This null vector gives us the coefficients of the approximate factor $g(x,y)$, which we here normalize by dividing by the biggest coefficient in absolute value and keep the first 5 significant digits, because the next smallest singular value is 5 orders of magnitude larger than the smallest. We have that

$$g(x,y) \;=\; 1.0000y + 1.0000x^2 - .0050433.$$

We now perform the approximate division of $f(x,y)$ by $g(x,y)$ to find an $h(x,y)$ such that $f \approx gh$. The support of h will be $[1,x,x^2,y]$. Solving the linear system in the unknown coefficients of h, we get

$$h(x,y) \;=\; 1.0000y - 1.0000x^2 + 0.0049999.$$

The change in f necessary to make these factors exact is $O(\epsilon^2)$. We have that $\|f-gh\|/\|f\| = .47 \times 10^{-4}$.

5.2 From Kaltofen's challenge problems

Consider the polynomial $p(x,y,z) =$

$$81\,x^4 + 16\,y^4 - 648\,z^4 + 72\,x^2y^2 - 648\,x^2 - 288\,y^2 + 129$$

which is irreducible over C, but factorizes in an extension of C containing $\sqrt{2}$. We perturb slightly the original polynomial $p(x,y,z)$,

$$\tilde{p}(x,y,z) = 81\,x^4 + 16\,y^4 - 648.001\,z^4 + 72\,x^2y^2 + 0.002\,x^2z^2 \\ +0.001\,y^2z^2 - 648\,x^2 - 288\,y^2 - 0.007\,z^2 + 1296$$

and then we project, taking for example $f(x,y) = \tilde{p}(x,y,1) =$

$$81\,x^4 + 16\,y^4 + 647.992 + 72\,x^2y^2 - 647.998\,x^2 - 287.999\,y^2.$$

We choose $x_0 = 0.1$ and solve numerically the fourth degree equation $f(x_0,y) = 0$. We select the root

$$y_0 = -1.6166362044176.$$

As before, we integrate the differential equations (5). An interval where the condition number behaves nicely is $[0,4]$ in this case. The numerical solution obtained is used to interpolate a factor of total degree 2. The approximate factor g we obtain this way is:

$$g(x,y) = 1.0 - 0.37936085353\,y^2 - 0.85356238581\,x^2.$$

The corresponding h factor that we obtain after solving the linear system is:

$$h(x,y) = 647.991999864 - 42.176201951\,y^2 - 94.89640272\,x^2$$

This gives $\|f-gh\|/\|f\| = .25 \times 10^{-9}$.

5.3 A higher-degree example

This example demonstrates the factorization of a randomly chosen composite polynomial of degree 15. The factors g and h were constructed using Maple's `randpoly` function:

$$g_0(x,y) = -84 + 41\,x + 23\,y + 99\,x^2y^5 - 61\,x^2y^4 \\ -50\,x^2y^3 - 12\,x^2y^2 - 18\,x^2y - 26\,xy^7 - 62\,xy^6 \\ +xy^5 - 47\,xy^4 - 91\,xy^3 - 47\,xy^2 + 66\,x^3y - 55\,x^7y \\ -35\,x^6y^2 + 97\,x^6y + 79\,x^5y^3 + 56\,x^5y^2 + 49\,x^5y \\ +57\,x^4y^4 - 59\,x^4y^3 + 45\,x^4y^2 - 8\,x^4y + 92\,x^3y^5 \\ +77\,x^3y^2 + 54\,x^3 + 53\,y^6 + 31\,x^2 - 90\,y^7 - 58\,y^8 \\ -85\,x^8 - 37\,x^7 - 86\,y^2 + 50\,x^6 + 83\,y^3 + 63\,x^5 + 94\,y^4 \\ -93\,x^4 - y^5 - 5\,x^2y^6 - 61\,xy + 43\,x^3y^4 - 62\,x^3y^3$$

$$h_0(x,y) = -76 - 53\,x + 88\,y + 66\,x^2y^5 - 29\,x^2y^4 \\ -91\,x^2y^3 - 53\,x^2y^2 - 19\,x^2y + 68\,xy^6 - 72\,xy^5 \\ -87\,xy^4 + 79\,xy^3 + 43\,xy^2 + 80\,x^3y - 50\,x^6y \\ -53\,x^5y^2 + 85\,x^5y + 78\,x^4y^3 + 17\,x^4y^2 + 72\,x^4y \\ +30\,x^3y^2 + 72\,x^3 - 23\,y^6 - 47\,x^2 - 61\,y^7 + 19\,x^7 \\ -42\,y^2 + 88\,x^6 - 34\,y^3 + 49\,x^5 + 31\,y^4 - 99\,x^4 - 37\,y^5 \\ -66\,xy - 85\,x^3y^4 - 86\,x^3y^3$$

We construct the polynomial

$$f(x,y) = g_0(x,y)\,h_0(x,y) + .000038190.$$

Random pertubations of the constant terms are as difficult for a general algorithm as perturbations of all terms. We take our random initial value, x_0, and solution, y_0, to be

$$x_0 \;=\; .709739738485 + .231104248045\,i$$
$$y_0 \;=\; -1.78481798491727 + .362826862436653\,i$$

We randomly select 96 points from the NODES solution and use 4 Newton iterations to refine them to 30 digits of accuracy, and take the SVD, again using 30 digits.

This yields one factor $g(x,y) \approx$

$$
\begin{aligned}
&-.0556934\,y - .0993286\,x - .0750914\,x^2 - .152626\,x^5 \\
&+.225287\,x^4 - .200987\,y^3 + .208376\,y^2 - .130787\,x^3 \\
&-.121128\,x^6 + .00246325\,y^5 - .227633\,y^4 + .0896292\,x^7 \\
&-.128380\,y^6 + .140503\,y^8 + .218023\,y^7 + .205910\,x^8 \\
&+.220507\,x\,y^3 + .0193743\,x^4\,y - .186511\,x^3\,y^2 + .121151\,x^2\,y^3 \\
&+.113867\,x\,y^4 - .109010\,x^4\,y^2 + .150200\,x^3\,y^3 + .147777\,x^2\,y^4 \\
&-.00244011\,x\,y^5 - .234977\,x^6\,y - .135662\,x^5\,y^2 + .142930\,x^4\,y^3 \\
&-.104172\,x^3\,y^4 - .239828\,x^2\,y^5 + .150178\,x\,y^6 + .133236\,x^7\,y \\
&+.0847897\,x^6\,y^2 + .147811\,x\,y + .0436168\,x^2\,y + .113946\,x\,y^2 \\
&-.159850\,x^3\,y + .0290970\,x^2\,y^2 + .0629809\,x\,y^7 - .191375\,x^5\,y^3 \\
&-.138079\,x^4\,y^4 - .222870\,x^3\,y^5 + .0121108\,x^2\,y^6 - .118718\,x^5\,y \\
&+.203497\,,
\end{aligned}
$$

and approximate division yields $h(x,y) \approx$

$$
\begin{aligned}
&21874.01827\,x - 36328.17297\,y - 32616.09320\,x\,y^3 \\
&-33025.43804\,x^3\,y - 17759.24918\,x\,y^2 + 35913.92936\,x\,y^4 \\
&+20641.92003\,x^6\,y - 35088.72943\,x^5\,y + 21877.20676\,x^5\,y^2 \\
&+40865.89508\,x^4 - 27248.14126\,x^2\,y^5 + 11967.23492\,x^2\,y^4 \\
&+37562.97909\,x^2\,y^3 + 21876.38942\,x^2\,y^2 + 29724.58872\,x\,y^5 \\
&+31369.19022 + 7841.558137\,x^2\,y - 28070.23313\,x\,y^6 \\
&+19397.99685\,x^2 - 29726.85747\,x^3 - 36325.08269\,x^6 \\
&-20228.23278\,x^5 + 9495.734559\,y^6 + 25183.17639\,y^7 \\
&-7844.188751\,x^7 + 17334.48685\,y^2 + 14032.89722\,y^3 \\
&-12799.49260\,y^4 + 15273.80181\,y^5 - 32199.67475\,x^4\,y^3 \\
&-7016.151176\,x^4\,y^2 - 29718.78564\,x^4\,y - 12384.61689\,x^3\,y^2 \\
&+27240.48566\,x\,y + 35089.23956\,x^3\,y^4 + 35501.29499\,x^3\,y^3\,.
\end{aligned}
$$

Then we have $\|f - gh\|/\|f\| = .00012$.

6. CONCLUDING REMARKS

We presented a new algorithm for factoring bivariate approximate polynomials, and analyzed its behaviour. The algorithm works for bivariate approximate polynomials that are close enough to exactly factorable polynomials. The complexity of the algorithm is polynomial in the degree of the input polynomials, modulo the conjecture in section 4.

7. REFERENCES

[1] ASCHER, U., AND PETZOLD, L. R. Projected collocation for higher-order higher-index differential-algebraic equations. *Journal of Computational and Applied Mathematics 43* (1992), 243–259.

[2] BECKERMANN, B., AND LABAHN, G. When are two polynomials relatively prime? *Journal of Symbolic Computation 26* (1998), 677–689.

[3] BERZ, M., AND MAKINO, K. Verified integration of ODEs and Flows using differential algebraic methods on high-order Taylor models. *Reliable Computing 4* (1998), 361–369.

[4] BJORCK, A. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, USA, 1996.

[5] CORLESS, R. M. An elementary solution of a minimax problem arising in algorithms for automatic mesh selection. Tech. Rep. TR-00-10, The Ontario Research Centre for Computer Algebra, Dec 2000.

[6] CORLESS, R. M., GIANNI, P. M., AND TRAGER, B. M. A reordered Schur factorization method for zero-dimensional polynomial systems with multiple roots. In *International Symposium on Symbolic and Algebraic Computation* (Maui, USA, 1997), W. Küchlin, Ed., ACM, pp. 133–140.

[7] CORLESS, R. M., GIANNI, P. M., TRAGER, B. M., AND WATT, S. M. The Singular Value Decomposition for polynomial systems. In *International Symposium on Symbolic and Algebraic Computation* (Montréal, Canada, 1995), A. Levelt, Ed., ACM, pp. 195–207.

[8] CORLESS, R. M., GIESBRECHT, M. W., JEFFREY, D. J., AND WATT, S. M. Approximate polynomial decomposition. In *International Symposium on Symbolic and Algebraic Computation* (Vancouver, Canada, 1999), S. S. Dooley, Ed., ACM, pp. 213–220.

[9] CORLESS, R. M., GIESBRECHT, M. W., KOTSIREAS, I. S., AND WATT, S. M. Numerical implicitization of parametric hypersurfaces with linear algebra. In *Proceedings of AISC* (2000), vol. 1930 of *LNAI*, Springer, p. to appear.

[10] GALLIGO, A., AND WATT, S. M. A numerical absolute primality test for bivariate polynomials. In *International Symposium on Symbolic and Algebraic Computation* (Maui, USA, 1997), W. Küchlin, Ed., ACM, pp. 217–224.

[11] GIANNI, P., SEPPÄLÄ, M., SILHOL, R., AND TRAGER, B. Riemann surfaces, plane algebraic curves and their period matrices. *Journal of Symbolic Computation 26*, 6 (1998), 789–803. Special issue of the JSC on Symbolic Numeric Algebra for Polynomials S. M. Watt and H. J. Stetter, editors.

[12] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix Computations*, 2nd ed. Johns Hopkins University Press, Baltimore and London, 1989.

[13] HITZ, M. A., KALTOFEN, E., AND LAKSHMAN Y. N. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In *International Symposium on Symbolic and Algebraic Computation* (Vancouver, Canada, 1999), S. S. Dooley, Ed., ACM, pp. 205–212.

[14] HUANG, Y., WU, W., STETTER, H. J., AND ZHI, L. Pseudofactors of multivariate polynomials. In *International Symposium on Symbolic and Algebraic Computation* (St. Andrew's, Scotland, 2000), C. Traverso, Ed., ACM, pp. 161–168.

[15] KALTOFEN, E. Fast parallel absolute irreducibility testing. *Journal of Symbolic Computation 1*, 1 (1985), 57–67. Misprint corrections: *J. Symbolic Comput.* vol. 9, p. 320 (1989).

[16] KALTOFEN, E. Polynomial factorization 1987-1991. In *Proc. LATIN '92* (Heidelberg/New York, 1992), I. Simon, Ed., vol. 583 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 294–313.

[17] KALTOFEN, E. Challenges of symbolic computation: My favorite open problems. *Journal of Symbolic Computation 29*, 6 (2000), 891–919. With an additional open problem by R. M. Corless and D. J. Jeffrey.

[18] KARMARKAR, N., AND LAKSHMAN Y. N. Approximate polynomial greatest common divisors and nearest singular polynomials. In *International Symposium on Symbolic and Algebraic Computation* (Zürich, Switzerland, 1996), ACM, pp. 35–42.

[19] PAN, V. Y. Solving a polynomial equation: Some history and recent progress. *SIAM Review 39*, 2 (1997), 187–220.

[20] SASAKI, T. Approximate multivariate polynomial factorization based on zero-sum relations. International Symposium on Symbolic and Algebraic Computation, p. to appear.

[21] SASAKI, T., SAITO, T., AND HILANO, T. Analysis of approximate factorization algorithm I. *Japan J. Industrial and Applied Math 89* (1992), 351–368.

[22] SASAKI, T., SAITO, T., AND HILANO, T. A unified method for multivariate polynomial factorization. *Japan J. Industrial and Applied Math 10*, 1 (February 1993), 21–39.

[23] SASAKI, T., SUZUKI, M., KOLÁŘ, M., AND SASAKI, M. Approximate factorization of multivariate polynomials and absolute irreducibility testing. *Japan J. Industrial and Applied Math 8* (1991), 357–375.

[24] SHAMPINE, L. F., AND CORLESS, R. M. Initial value problems for ODEs in problem solving environments. *Journal of Computational and Applied Mathematics* (2000). to appear.