COMPUTING THE COMMON ZEROS OF TWO BIVARIATE FUNCTIONS VIA BÉZOUT RESULTANTS

YUJI NAKATSUKASA*, VANNI NOFERINI†, AND ALEX TOWNSEND‡

Abstract. The common zeros of two bivariate functions can be computed by finding the common zeros of their polynomial interpolants expressed in a tensor Chebyshev basis. From here we develop a bivariate rootfinding algorithm based on the hidden variable resultant method and Bézout matrices with polynomial entries. Using techniques including domain subdivision, Bézoutian regularization and local refinement we are able to reliably and accurately compute the simple common zeros of two smooth functions with polynomial interpolants of very high degree (≥ 1000). We analyze the resultant method and its conditioning by noting that the Bézout matrices are matrix polynomials. Our robust algorithm is implemented in the `roots` command in Chebfun2, a software package written in object-oriented MATLAB for computing with bivariate functions.

Key words. bivariate function, polynomial, rootfinding, resultant, Bézoutian, conditioning, regularization, Chebfun2

AMS subject classifications. 65D15, 65F15, 65F22

1. Introduction. There are two operations on bivariate functions that are commonly referred to as *rootfinding*: (1) Finding the zero level curves of $f(x, y)$, and (2) Finding the common zeros of $f(x, y)$ and $g(x, y)$. Despite sharing the same name these operations are mathematically distinct; the first has solutions along curves while the second, typically, has isolated solutions. In this paper we concentrate on the second, that is, computing all the pairs $(x, y) \in [-1, 1] \times [-1, 1]$ such that

$$\begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix} = 0 \tag{1.1}$$

under the assumption that the solution set is zero-dimensional, i.e., the solutions are isolated. In this typical situation we describe a robust, accurate, and fast numerical algorithm that can solve problems of much higher degree than in previous studies.

Our algorithm has been implemented in the `roots` command in Chebfun2, an open source software system written in object-oriented MATLAB that extends Chebfun [46] to functions of two variables defined on rectangles [43]. Chebfun2 approximates a bivariate function by a polynomial interpolant represented as a "chebfun2" object, which is accurate to relative machine precision [43]. Our first step is to use Chebfun2 to replace $f(x, y)$ and $g(x, y)$ in (1.1) by polynomial interpolants $p(x, y)$ and $q(x, y)$, respectively, before finding the common zeros of the corresponding polynomial system. Throughout this paper we use f and g to denote smooth bivariate functions, and p and q for their polynomial interpolants, which approximate f and g to relative machine precision, respectively, of the form:

$$p(x,y) = \sum_{i=0}^{m_p} \sum_{j=0}^{n_p} P_{ij} T_i(y) T_j(x), \qquad q(x,y) = \sum_{i=0}^{m_q} \sum_{j=0}^{n_q} Q_{ij} T_i(y) T_j(x), \tag{1.2}$$

*School of Mathematics, University of Manchester, Manchester, England, M13 9PL (yuji.nakatsukasa@manchester.ac.uk). Supported by EPSRC grant EP/I005293/1.

†School of Mathematics, University of Manchester, Manchester, England, M13 9PL (vanni.noferini@manchester.ac.uk). Supported by ERC Advanced Grant MATFUN (267526).

‡Mathematical Institute, University of Oxford, Oxford, England, OX1 3LB (townsend@maths.ox.ac.uk). Supported by EPSRC grant EP/P505666/1 and the ERC grant FP7/2007-2013 to L. N. Trefethen

where $T_j(x) = \cos(j \cos^{-1}(x))$ is the Chebyshev polynomial of degree j, and $P \in \mathbb{R}^{m_p \times n_p}, Q \in \mathbb{R}^{m_q \times n_q}$ are the matrices of coefficients.

The resulting polynomial system, $p(x, y) = q(x, y) = 0$, is solved by an algorithm based on the *hidden variable resultant method* and Bézout resultant matrices that have entries containing univariate polynomials. One component of the solutions is computed by solving a polynomial eigenvalue problem via a standard approach in the matrix polynomial literature known as *linearization* [19] and a robust eigenvalue solver, such as `eig` in MATLAB. The remaining component is found by univariate rootfinding based on the *colleague matrix* [12]. Usually, resultant methods have several computational drawbacks and our algorithm is motivated by attempting to overcome them:

1. *Computational complexity:* The complexity of resultant methods based on Sylvester and Bézout matrices grows like the degree of p and q to the power of 6. Therefore, computations quickly become unfeasible when the degrees are larger than, say, 30, and this is one reason why the literature concentrates on examples with degree 20, or smaller [1, 9, 17, 25, 32]. Typically, we reduce the complexity to quartic scaling, and sometimes even further, by using domain subdivision, which is beneficial for both accuracy and efficiency (see section 4). Subdivision can allow us to compute the common zeros to (1.1) with polynomial interpolants of degree as large as 1000, or more.
2. *Conditioning:* We analyze the conditioning of Bézout matrix polynomials and derive condition numbers. The analysis reveals that recasting the problem in terms of resultants can worsen the conditioning of the problem, and we resolve this via a *local refinement* process. The Bézout local refinement process improves accuracy significantly, and also identifies and removes spurious zeros; a well known difficulty for resultant based methods.
3. *Numerical instability:* The Bézout matrix polynomial is numerically close to singular, leading to numerical difficulties, and to overcome this we apply a regularization step. The regularization we employ crucially depends on the symmetry of Bézout matrices. We then proceed to solve the polynomial eigenvalue problem by linearization. This means the highly oscillatory determinants of resultant matrices are not interpolated, overcoming further numerical difficulties.
4. *Robustness:* It is not essential for a bivariate rootfinding method to construct an eigenvalue problem, but if it does it is convenient and can improve robustness. For this reason we solve an eigenvalue problem to find the first component of the solutions to (1.1), and another one to find the second. Therefore, our algorithm inherits some of its robustness from the QZ algorithm [20], which is implemented in the `eig` command in MATLAB.
5. *Resultant construction:* Resultant methods are more commonly based on Sylvester resultant matrices that are preferred due to their ease of construction, and this is especially true when dealing with polynomials expressed in an orthogonal polynomial basis rather than the monomial basis. However, the Bézout resultant matrices, when using the tensor Chebyshev basis, are easily constructed by using MATLAB code in [42]. We choose Bézout over Sylvester because its conditioning can be analyzed (see section 5) so its numerical behavior is better understood, and its symmetry can be exploited.

Figure 1.1 summarizes the main steps in our bivariate rootfinding algorithm based on the hidden variable resultant method, domain subdivision, Bézout regularization, and the tensor Chebyshev basis. In particular, it is crucial that we express the poly-

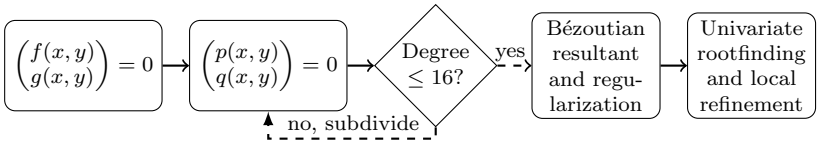

Fig. 1.1: Flowchart of our bivariate rootfinding algorithm based on the hidden variable resultant method and Bézout matrices. First, we approximate the bivariate functions f and g by polynomial interpolants, using domain subdivision until the polynomial degree on each subdomain is less than 16 (section 4). Then we construct the resultant (section 3), regularize (section 6), and then solve for one of the components before using a univariate rootfinding algorithm to compute the remaining component of the common zeros. Finally, we refine the solutions and remove any spurious zeros via a local Bézout resultant (section 7).

nomial interpolants of f and g in a tensor Chebyshev basis so that we can employ fast transforms for interpolation based on the fast Fourier transform, overcome numerical instability associated with interpolation at equally-spaced points, and prevent the common zeros from being sensitive to perturbations of the polynomial coefficients [47].

Our algorithm is capable of finding the common zeros of f and g when the degrees of their polynomial interpolants are much higher than in previous studies. For example, the built-in commands in Maple and Mathematica, which implement resultant-based algorithms in the monomial basis, require about 60 seconds for degree 16 examples (with dubious accuracy) on a Core i7 desktop machine, and cannot handle degree 30 or higher. Our MATLAB code completes degree 16 in a fraction of a second, needs about 5 seconds for degree 40, and approximately 100 seconds for degree 100. Often, the runtime scales *quartically*, and sometimes much less, with the degree, and our code can solve problems with degree 1000 (see section 8). In this paper we refer to the maximal degree instead of the total degree, that is, $\sum_{i=0}^{N}\sum_{j=0}^{N} c_{ij}y^i x^j$ is a degree N bivariate polynomial.

For definiteness, unless stated otherwise, we refer to the problem of finding the common zeros in the domain $[-1,1]\times[-1,1]$, but the algorithm and code deal with any rectangular domain.

In the next section we review previous literature related to solving (1.1), and section 3 contains the details of the hidden variable resultant method based on Bézout matrices. In section 4 we explain the process of domain subdivision in our algorithm, and section 5 analyzes the conditioning of the Bézout resultant method. In section 6 we describe a regularization step to improve the stability of the algorithm. In section 7 we discuss further details of the algorithm, and in section 8 we present several numerical examples that highlight its robustness and accuracy.

2. Existing bivariate rootfinding algorithms. A number of studies have been carried out on bivariate rootfinding [9, 25, 32], and we mainly concentrate on approaches that require numeric, as opposed to symbolic, computations. These include variations on the resultant method, contouring algorithms, homotopy continuation, and a two-parameter eigenvalue approach. It is also worth mentioning the existence of important approaches based on Gröbner bases, even though some symbolic manipulations are usually employed [15]. Generally, Gröbner bases are feasible only for small degree polynomial systems because of their doubly exponential worst-case complexity,

and numerical instability.

2.1. Resultant methods. The hidden variable resultant method is based on selecting one variable, say y, and writing $p(x, y)$ and $q(x, y)$ as polynomials in x with coefficients in $\mathbb{C}[y]$, that is,

$$p(x, y) = p_y(x) = \sum_{j=0}^{n_p} \alpha_j(y) x^j, \qquad q(x, y) = q_y(x) = \sum_{j=0}^{n_q} \beta_j(y) x^j. \tag{2.1}$$

Note that, although we have expressed (2.1) in the monomial basis, any polynomial basis can be used, see for example [8]. Monomials are the standard basis in the literature [16, 18], due to their simplicity and flexibility for algebraic manipulations. On the other hand, the Chebyshev polynomial basis is a better choice for numerical stability on the real interval $[-1, 1]$ [45]. For this reason, we always express polynomials in the Chebyshev basis.

The two polynomials $p_y(x)$ and $q_y(x)$ in (2.1), thought of as univariate in x, have a common zero if and only if a resultant matrix is singular [5]. Therefore, the y-values of the solutions to (1.1) can be computed by finding the y-values such that a resultant matrix is singular.

There are many different resultant matrices such as Sylvester [16], Bézout [7], and other matrices [5, 18, 26], and this choice can affect subsequent efficiency (see Table 3.1) and conditioning (see section 5). Usually, resultant matrices are constructed from polynomials expressed in the monomial basis [9, 32], but they can be derived when using any other bases, see for example [8].

Finding the y-values such that the resultant matrix is singular is equivalent to a polynomial eigenvalue problem, and many techniques exist such as methods based on contour integrals [2, 6], Newton-type methods and inverse iteration methods (see the review [34]), the Ehrlich-Aberth method [10], and the standard approach of solving via linearization [19]. In our implementation we use linearization, which replaces a polynomial eigenvalue problem with a generalized eigenvalue problem with the same eigenvalues and Jordan structure [19]. We then apply the QZ algorithm to the linearization; for a discussion of possible alternatives see [34]. We leave for future research the study of the possible incorporation in our algorithm of alternative techniques to solve the polynomial eigenproblem.

Another related approach is the u-resultant method, which works similarly and starts by introducing a dummy variable to make the polynomial interpolants homogeneous. The hidden variable resultant method is then applied to the new polynomial system selecting the dummy variable first. This is quite natural because it ensures that the x- and y-variables are treated in the same way, but unfortunately, making a polynomial homogeneous inherently requires the monomial basis and hence, entails a corresponding numerical instability.

Some resultant methods first apply an orthogonal transformation, such as the `BivariateRootfinding` command in Maple, which is used to ensure that two common zeros do not share the same y-value. We provide a careful case-by-case study to show that our approach does not require such a transform. Furthermore, other changes of variables can be applied, such as $x = (z + \omega)/2$ and $y = (z - \omega)/2$, and the selecting the variable ω, which satisfies $\omega = \bar{z}$ when x and y are real. We have found that such changes of variables give little improvement for most practical examples.

2.2. Contouring algorithms. Contouring algorithms such as marching squares and marching triangles [23] are employed in computer graphics to generate zero level

curves of bivariate functions. These contouring algorithms can be very efficient at solving (1.1), and until this paper the `roots(f,g)` command in Chebfun2 exclusively employed such a contouring approach [43]. In the older version of Chebfun2 the zero level curves of f and g were computed separately using the MATLAB command `contourc`, and then the intersections of these zero level curves were used as initial guesses for Newton's iteration.

Bivariate rootfinding based on a contouring algorithm suffers from several drawbacks:

1. The level curves of f and g may not be smooth even for very low degree polynomials, for example, $f(x, y) = y^2 - x^3$.
2. The number of disconnected components of the level curves of f and g can be potentially quite large [21].
3. Contours of f or g are not always grouped correctly, leading to the potential for missed solutions.
4. The zero level curves must be discretized and therefore, the algorithm requires a fine tuning of parameters to balance efficiency and reliability.

For many practical applications solving (1.1) via a contouring algorithm may be an adequate approach, but not always, and this paper started with seeking a more robust alternative algorithm.

Remark The `roots(f,g)` command in Chebfun2 implements both the algorithm based on a contouring algorithm and the approach described in this paper. An optional parameter can be supplied to select one or the other of these choices. We have decided to keep the contouring approach as an option because it can sometimes run faster, but we treat its computed solutions with suspicion, and we employ the approach described in this paper as a robust alternative.

2.3. Other numerical methods. Homotopy continuation methods [39] have a simple underlying approach based on solving an initial easy polynomial system that can be continuously deformed into (1.1). Along the way several polynomial systems are solved with the current solution set being used as an initial guess for the next. These methods have received significant research attention and are a purely numerical approach that can solve multivariate versions of (1.1) [39].

The two-parameter eigenvalue approach constructs a determinantal expression for the polynomial interpolants to f and g and then rewrites (1.1) as a two-parameter eigenvalue problem [3],

$$A_1 v = xB_1 v + yC_1 v, \qquad A_2 w = xB_2 w + yC_2 w.$$

This approach has advantages because the two-parameter eigenvalue problem can be solved with the QZ algorithm [24], or other techniques [24]. However, the construction of a determinantal expression and hence, the matrices A_i, B_i, C_i for $i = 1, 2$ currently requires the solution of a multivariate polynomial system [36].

3. The resultant method with Bézout matrices. We now describe in more detail the hidden variable resultant method using Bézout matrices that forms the core of our algorithm. The initial step of the resultant method selects a variable to solve for first, and our choice is based on the efficiency of subsequent steps. For simplicity, throughout the paper, we select the y-variable and write the polynomials $p(x, y)$ and $q(x, y)$ as functions of x with coefficients in $\mathbb{C}[y]$, using the Chebyshev basis:

$$p_y(x) = \sum_{j=0}^{n_p} \alpha_j(y) T_j(x), \qquad q_y(x) = \sum_{j=0}^{n_q} \beta_j(y) T_j(x), \qquad x \in [-1, 1], \tag{3.1}$$

where $\alpha_j, \beta_j \in \mathbb{C}[y]$, i.e., polynomials in y with complex coefficients, have the polynomial expansions

$$\alpha_j(y) = \sum_{i=0}^{m_p} P_{ij} T_i(y), \qquad \beta_j(y) = \sum_{i=0}^{m_q} Q_{ij} T_i(y), \qquad y \in [-1, 1],$$

where the matrices P and Q are as in (1.2).

The (Chebyshev) Bézout matrix of p_y and q_y in (3.1), denoted by $B(p_y, q_y)$, is defined[1] by $B(p_y, q_y) = (b_{ij})_{0 \le i,j \le N-1}$ [42], where $N = \max(n_p, n_q)$ and the entries satisfy

$$\frac{p_y(s)q_y(t) - p_y(t)q_y(s)}{s - t} = \sum_{i,j=0}^{N-1} b_{ij} T_i(s) T_j(t). \tag{3.2}$$

We observe that since $b_{ij} \in \mathbb{C}[y]$, $B(p_y, q_y)$ can be expressed as a *matrix polynomial* [19], i.e., a polynomial with matrix coefficients, that is

$$B(y) = B(p_y, q_y)(y) = \sum_{i=0}^{M} A_i T_i(y), \qquad A_i \in \mathbb{C}^{N \times N}, \tag{3.3}$$

where $M = m_p + m_q$ is the sum of the degrees of $p(x, y)$ and $q(x, y)$ in the y-variable. The *resultant* of p_y and q_y is defined as the determinant of $B(y)$, which is a scalar univariate polynomial in y. The usual resultant definition is via the Sylvester matrix, but we use the symmetric Bézout matrix, and later explain that these two definitions are almost the same, but subtly different (see section 3.1).

It is well-known [16, Prop. 3.5.8 & Cor. 3.5.4] that two bivariate polynomials share a common factor in $\mathbb{C}[x, y]$ only if their resultant is identically zero. Throughout this paper we are assuming the solution set to (1.1) is zero-dimensional and hence, that the polynomial interpolants p and q do not share a common factor. This has two important implications:

- (Bézout's Theorem) The number of solutions to (1.1) is finite [27, Ch. 3];
- (Non-degeneracy) The resultant, $\det(B(y))$, is not identically zero and therefore, $B(y)$ is a *regular* matrix polynomial [19].

Since the resultant is not identically zero, the solutions to

$$\det\left(B(y)\right) = \det\left(B(p_y, q_y)\right) = 0 \tag{3.4}$$

are the *finite eigenvalues* of $B(y)$, and we observe that if y_* is a finite eigenvalue of $B(y)$, then the resultant of $p_{y_*}(x) = p(x, y_*)$ and $q_{y_*}(x) = q(x, y_*)$ is zero. Usually, a zero resultant means that $p(x, y_*)$ and $q(x, y_*)$ share a finite common root [5], but it can also happen that they share a "common root at infinity" (see section 3.1).

The eigenvalues of $B(y)$ can be found by via linearization of matrix polynomials expressed in polynomial bases [31, 42], which constructs a generalized eigenvalue problem $Cv = \lambda Ev$ with the same eigenvalues as $B(y)$. In our implementation we choose a linearization computed by the MATLAB code of [42], and then employ the `eig` command in MATLAB to solve $Cv = \lambda Ev$. The process of linearization converts the problem of finding the eigenvalues of $B(y)$, a matrix polynomial of degree N and

[1] Historically, this functional viewpoint of a Bézout matrix is in fact due to Cayley, who modified the original method of Bézout, both in the monomial basis [38, Lesson IX].

Resultant	Size of A_i in (3.3)	Degree	Size of $Cv = \lambda Ev$
Bézout (y first)	$\max(n_p, n_q)$	$m_p + m_q$	$\max(n_p, n_q)(m_p + m_q)$
Bézout (x first)	$\max(m_p, m_q)$	$n_p + n_q$	$\max(m_p, m_q)(n_p + n_q)$
Sylvester (y first)	$n_p + n_q$	$\max(m_p, m_q)$	$\max(m_p, m_q)(n_p + n_q)$
Sylvester (x first)	$m_p + m_q$	$\max(n_p, n_q)$	$\max(n_p, n_q)(m_p + m_q)$

Table 3.1: Sizes and degrees of matrix polynomials constructed from the Bézout and Sylvester [5] resultant matrices. The product of the size of the A_i in (3.3) and degree is the size of the resulting generalized eigenvalue problem $Cv = \lambda Ev$, which depends on whether the x- or y-variable is solved for first. We use the Bézout resultant matrix and solve for the y-values first if $\max(n_p, n_q)(m_p + m_q) \leq \max(m_p, m_q)(n_p + n_q)$; the x-values first, otherwise.

size M, to an $MN \times MN$ generalized eigenvalue problem. Typically, the majority of the computational cost of our algorithm is in solving these generalized eigenvalue problems.

After filtering out the y-values that do not correspond to solutions in $[-1, 1] \times [-1, 1]$, the x-values are computed via two independent univariate rootfinding problems: $p_y(x) = 0$ and $q_y(x) = 0$. These univariate problems are solved in standard Chebfun fashion by computing the eigenvalues of a colleague matrix [45] (see section 7).

The resultant method with Bézout matrices also works if we select the x-value to solve for first, and this choice can change the cost of the computation. Let n_p and m_p be the degrees of $p(x, y)$ in the x- and y-variable, respectively, and similarly let n_q and m_q be the degrees for $q(x, y)$. If the y-values are solved for first, then the generalized eigenvalue problem $Cv = \lambda Ev$ formed after linearization is of size $\max(n_p, n_q)(m_p + m_q)$ and the cube of this number is the expected computational complexity. If the x-values are solved for first, then the generalized eigenvalue problem is of size $\max(m_p, m_q)(n_p + n_q)$, which is usually roughly comparable, but in some cases can be considerably larger (or smaller). Table 3.1 gives the various sizes and degrees resulting from the choice of resultant matrix and variable computed first. We always use Bézout resultant matrices because of their symmetry, but select the variable to solve for first by minimizing the size of the resulting generalized eigenvalue problem.

3.1. Algebraic subtleties. The resultant method is mathematically quite subtle [16, Ch. 3 & 9], and we summarize some of these subtleties here. These delicate issues arise because there are many different ways in which a common zero of p and q can manifest itself as an eigenvalue of $B(y)$. As always we assume that the solution set to (1.1) is zero-dimensional, and here further assume the common zeros to (1.1) are simple, i.e., if (x_*, y_*) is a common zero of p and q, then the Jacobian of p and q is invertible there. There are two mutually exclusive cases:

1. *Finite common zero:* There exists $(x_*, y_*) \in \mathbb{C}^2$ such that $p(x, y_*)$ and $q(x, y_*)$ are nonzero and share a common finite root at x_*. In this case, y_* is an eigenvalue of $B(y)$ with an eigenvector in Vandermonde form, i.e., $[T_0(x_*), T_1(x_*), \ldots, T_{N-1}(x_*)]^T$.
2. *Common zero at infinity:* There exists $y_* \in \mathbb{C}$ such that $p(x, y_*)$ and $q(x, y_*)$

are nonzero and both have a zero coefficient in $T_N(x)$. We say that they share a "common zero at infinity" and in this case, y_* is a finite eigenvalue of $B(y)$ with eigenvector $[0, 0, \ldots, 0, 1]^T$.

It is worth noting that, if the degrees in x of $p_y(x)$ and $q_y(x)$ differ, then the definition of the resultant via $B(y)$ is slightly different to the determinant of the Sylvester matrix, as can be seen by adapting a result in [28]. For instance, if $n_p > n_q$, then the two determinants differ by a factor $C(\alpha_{n_p}(y))^{n_p - n_q}$, where C is a constant and $\alpha_{n_p}(y)$ is the leading coefficient of $p_y(x)$, but this does not alter the eigenvalues corresponding to finite common zeros.

The mathematical subtlety continues when there are many common zeros (possibly including "zeros at infinity") of p and q sharing the same y-value. In this case $B(y)$ has an eigenvalue with multiplicity greater than 1 even though p and q only have simple common zeros. Furthermore, there can exist $y_* \in \mathbb{C}$ such that either $p(x, y_*)$ or $q(x, y_*)$ is identically zero, in which case y_* is an eigenvalue of $B(y)$ with $B(y_*) = 0$, a zero matrix.

Eigenvalues of $B(y)$ of multiplicity > 1 can be ill-conditioned and hence, difficult to compute accurately, particularly in the presence of nontrivial Jordan chains. However, in the generic case where all the common zeros of p and q are simple[2], the x-values for any two common zeros with the same y-value are distinct. Moreover, the eigenvalues of $B(y)$ have the same geometric and algebraic multiplicity with eigenvectors spanned by Vandermonde vectors $\text{span}\{[T_0(x_i), \ldots, T_{N-1}(x_i)]^T\}$, where x_i are the x-values of the common zeros (x_i, y_*). This means the eigenvalues of $B(y)$ in $[-1, 1]$ are semisimple and hence, can be obtained accurately.

The intuitive explanation is that such solutions result only in *nondefective* (or semisimple) eigenvalues of $B(y)$, and nondefective multiple eigenvalues are no more sensitive to perturbation than simple eigenvalues. The simplest examples are symmetric matrices, whose eigenvalues are always well-conditioned as can be seen by Weyl's theorem and its sharpness. This fact is often overlooked and multiple eigenvalues are sometimes unnecessarily assumed to be ill-conditioned. Conditioning of semisimple multiple eigenvalues has been studied for matrices [40] and matrix pencils $\lambda X + Y$ [48], which show they are well-conditioned provided that $W^T X Z$ is nonsingular, where the columns of W and Z span the corresponding left and right eigenspaces, respectively. The bottom line is that, if the original problem (1.1) only has simple common zeros, then the resultant method can compute accurate solutions, and the intermediate step of solving the resultant (3.4) is valid. This means there is no requirement for a resultant method to do a change of variables to reduce the chance that two well-separated common zeros share the same y-value.

In the presence of multiple common zeros the situation becomes more delicate, and a complete analysis is beyond the scope of this paper. A reasonable requirement for a numerical algorithm is that it detects simple common zeros that are not too ill-conditioned, while multiple zeros are inherently ill-conditioned and difficult to compute accurately. Our code typically computes double eigenvalues with $\mathcal{O}(u^{1/2})$ accuracy, where u is the unit roundoff, i.e., in a backward stable manner.

Numerically, another well-known difficulty with resultant-based methods is the presence of spurious zeros, which are computed eigenvalues of $B(y)$ that do not correspond to any y-value of the common zeros (see section 7).

[2]This includes those at infinity. The requirement can be formalized, but the algebraic details are beyond the scope of this paper.

4. Subdivision of the domain. The Bézout resultant method of section 3 requires three main features to become practical: subdivision, local Bézout refinement, and regularization. The first of these is a 2D version of Boyd's subdivision technique [13, 14] for univariate rootfinding as utilized for many years in Chebfun [12, 46].

We recursively subdivide $[-1,1]\times[-1,1]$ into rectangles, in the x- and y-variable independently, until the polynomial interpolants to f and g on each subdomain are all of degree less than or equal to 16, a parameter determined by experimentation. Figure 4.1 shows how the square $[-1,1]\times[-1,1]$ is recursively subdivided for $\sin((x-1/10)y)\cos(1/(x+(y-9/10)+5)) = (y-1/10)\cos((x+(y+9/10)^2/4)) = 0$ with a polynomial system of degree less than or equal to 16 being solved in each subregion.

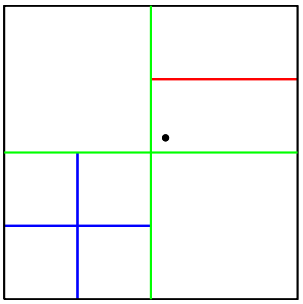

Fig. 4.1: Subdivision of $[-1,1]\times[-1,1]$ used for the solution of $f = \sin((x-1/10)y)\cos(1/(x+(y-9/10)+5)) = 0$ and $g = (y-1/10)\cos((x+(y+9/10)^2/4)) = 0$. The blue and red lines represent subdivisions used to ensure the piecewise interpolants to f and g have degree less than 16 on each subdomain. The green lines show subdivisions used by both interpolants. In each subdomain we solve a polynomial system of small degree, and the black dot is the only common zero.

We do not exactly bisect in the x- or y-direction, but instead subdivide asymmetrically to avoid splitting exactly at a solution to $f(x,y) = g(x,y) = 0$. That is, in the x-direction we subdivide $[-1,1]\times[-1,1]$ into the two subregions $[-1,r_x]\times[-1,1]$ and $[r_x,1]\times[-1,1]$, and in the y-direction $[-1,1]\times[-1,r_y]$ and $[-1,1]\times[r_y,1]$, where r_x and r_y are small arbitrary constants[3]. This is to avoid accidentally subdividing at a solution since functions arising in practice often have zeros at special points like 0 or $\pi/6$.

Usually, a piecewise polynomial interpolant can be of lower degree than a global polynomial approximation, but not always. For subdivision to be computationally worthwhile we require that, on average, each subdivision reduces the polynomial degree (in the x or y direction) by at least 79%. The estimate 79% is derived from the fact that $2(.79)^3 < 1$, and each subdivision in the x-direction splits a problem into two with the complexity of the algorithm depending cubically on the degree in x. Let $n = \max(m_p, n_p, m_q, n_q)$ be the maximal degree of p and q, and let $d = 16$ be the terminating degree for subdivsion. We define K as the smallest integer such that $n(.79)^K \le d$, and we stop subdividing if either the degree becomes lower than d, or

[3]We use $r_x \approx -0.004$ and $r_y \approx -0.0005$. There is no special significance of these constants apart from that they are small and arbitrary.

subdivision has been done K times.

For simplicity, suppose all the degrees of p and q are equal to n in both x and y, and that each subdivision leads to a reduction in the degree by a factor $0 < \tau \leq 1$ in both x and y. Subdividing is done k times in both x and y until $n\tau^k \leq d = 16$, so $k \approx (\log d - \log n)/\log \tau$. We then have at most 4^k subdomains, each requiring $\mathcal{O}(d^6)$ cost to solve a small generalized eigenvalue problem. Therefore, asymptotically in n, the overall cost is

$$\mathcal{O}\left(4^k\right) = \mathcal{O}\left(4^{\frac{\log d - \log n}{\log \tau}}\right) = \mathcal{O}\left(4^{-\frac{\log n}{\log \tau}}\right) = \mathcal{O}\left(n^{-\frac{\log 4}{\log \tau}}\right).$$

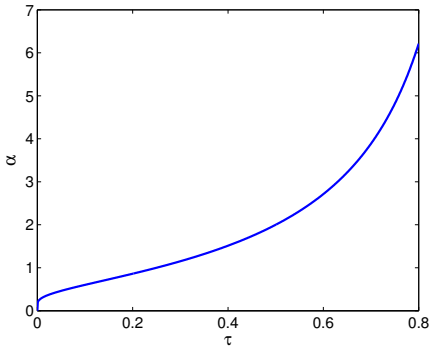

Fig. 4.2: Subdivision complexity is $\mathcal{O}\left(n^\alpha\right)$, where $\alpha = -\frac{\log 4}{\log \tau}$ and $0 < \tau \leq 1$ measures the reduction in polynomial degree after subdivision. When $\tau < 1/2$ polynomial evaluation is the significant cost rather than the solution of generalized eigenvalue problems.

Figure 4.2 shows the exponent $-\frac{\log 4}{\log \tau}$ as a function of τ. When $\tau \lesssim 0.5$, function evaluation dominates the overall complexity, which can be as low as $\mathcal{O}(n^2)$. When $\tau \approx 0.79$ it is as high as $\mathcal{O}(n^6)$, the same as the original problem without subdivision. We note that even if $\tau \approx 0.79$, it can still be useful to subdivide as long as $\tau < 1$, as it reduces the storage cost from $\mathcal{O}(n^4)$, for storing the linearization matrix pencil, to $\mathcal{O}(n^2)$, for storing the function evaluations.

The average degree reduction τ can take any value between 0 and 1 as the following examples show:

1. Suppose that $f(x, y) = f_1(x)f_2(y)$, where $f_1(x)$ and $f_2(y)$ are entire oscillatory functions. Then by Nyquist's sampling theorem [33] the functions h and k must be sampled at, at least, 2 points per wavelength to be resolved in the x and y direction. Since subdividing halves the number of oscillations in the x and y direction, the number of points required for resolution is halved in the x and y direction and therefore, $\tau^2 \approx 1/4$ and $\tau \approx 0.5$.
2. Suppose that $f(x, y) = h(x-y)$, where h is an entire oscillatory function, then by Nyquist's sampling theorem f must be sampled at 2 points per wavelength along the diagonal, i.e., $y = x$. A subdivision in both x and y reduces the length of the diagonal by 2 and hence, $\tau^2 \approx 1/2$ and $\tau \approx 0.707$.
3. Take the very contrived function $f(x, y) = |x - r_x||y - r_y|$, which is of very high numerical degree on $[-1, 1] \times [-1, 1]$. However, on subdivision the degree of f on each subdomain is just 1 and therefore, $\tau \approx 0$.

4. Suppose that $f(x,y) = |\sin(c(x-y))|$ with $c \gg 1$, then f is of very high numerical degree with a discontinuous first derivative along many diagonals. In this example, subdivision barely reduces the numerical degree and hence, for all practical purposes $\tau \approx 1$.

Two dimensional subdivision is also beneficial for accuracy of the computed solutions to (1.1) because it reduces the *dynamical range* (see section 5). In addition, subdivision reduces the size of the Bézout matrix, and this decreases the distance from singularity of the matrix polynomial $B(y)$ as defined in (3.3).

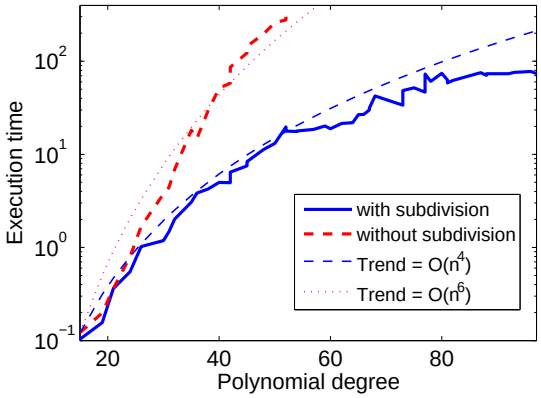

Fig. 4.3: Execution timings for computing the solutions to $\sin(\omega(x+y)) = \cos(\omega(x-y)) = 0$, where $\omega = 1, \ldots, 50$ (up to 20 without subdivision). In this example, subdivision has reduced the complexity of our algorithm from $\mathcal{O}(n^6)$ to $\mathcal{O}(n^4)$.

Typically, we observe that $\tau \approx \sqrt{2}/2 \approx 0.707$ and therefore, subdivision leads to the complexity of $\mathcal{O}(n^4)$. Figure 4.3 shows the computational time for solving $\sin(\omega(x+y)) = \cos(\omega xy) = 0$, for increasing ω, with and without subdivision.

5. Conditioning of the Bézout polynomial eigenvalue problem. Suppose that (x_*, y_*) is a simple common zero of p and q, and let $(\hat{x}, \hat{y}) = (x_* + \delta x, y_* + \delta y)$ be a common zero of the perturbed polynomials $\hat{p} = p + \delta p$ and $\hat{q} = q + \delta p$. Then we have, to first order,

$$0 = \begin{bmatrix} \hat{p}(\hat{x},\hat{y}) \\ \hat{q}(\hat{x},\hat{y}) \end{bmatrix} = \begin{bmatrix} \partial_x p(x_*,y_*) & \partial_y p(x_*,y_*) \\ \partial_x q(x_*,y_*) & \partial_y q(x_*,y_*) \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} + \begin{bmatrix} \delta p(\hat{x},\hat{y}) \\ \delta q(\hat{x},\hat{y}) \end{bmatrix}. \tag{5.1}$$

Our implementation initially scales p and q so that $\|p\|_\infty = \|q\|_\infty = 1$, and we assume this in the analysis below.

A stable numerical method computes a solution with an error of size $\mathcal{O}(\kappa_* u)$, where κ_* is the absolute condition number [22, Ch. 1] of (x_*, y_*), defined as

$$\kappa_* = \lim_{\epsilon \to 0^+} \sup \left\{ \frac{1}{\epsilon} \min \left\| \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right\|_2 : \hat{p}(\hat{x},\hat{y}) = \hat{q}(\hat{x},\hat{y}) = 0 \right\}, \tag{5.2}$$

where the supremum is taken over the set $\{(\hat{p},\hat{q}) : \left\| \left[\begin{smallmatrix} \|\delta p\|_\infty \\ \|\delta q\|_\infty \end{smallmatrix} \right] \right\|_2 \leq \epsilon\}$. Thus, by (5.1) the condition number (5.2) of the rootfinding problem is $\kappa_* = \|J^{-1}\|_2$, where J denotes the Jacobian matrix of p and q at (x_*, y_*).

To analyze the error in a solution obtained by our algorithm, we next examine the conditioning of the polynomial eigenvalue problem $B(y)$, which we use to find the y-values of the solution. To first order in u, the error in the computed eigenvalues of $B(y)$ is bounded by $u\cdot$(conditioning)$\cdot$(backward error), where the condition number $\kappa(y_*, B)$ is the conditioning of y_* as an eigenvalue of $B(y)$, defined by

$$\kappa(y_*, B) = \lim_{\epsilon \to 0^+} \sup \left\{ \frac{1}{\epsilon} \min |\hat{y} - y_*| : \det \widehat{B}(\hat{y}) = 0 \right\}, \tag{5.3}$$

where the supremum is taken over the set of matrix polynomials $\widehat{B}(y)$ such that $\max_{y\in[-1,1]} \|\widehat{B}(y) - B(y)\|_2 \leq \epsilon$. The backward error is the smallest $\|\Delta B(y)\| = \max_{y\in[-1,1]} \|\Delta B(y)\|_2$ such that the computed solution is an exact solution of $B(y) + \Delta B(y)$.

The special structure of Bézout matrix polynomials lets us characterize the eigenvalue condition number $\kappa(y_*, B)$ with respect to p and q, as we show in the next theorem.

THEOREM 5.1. *Let $B(y)$ be the Bézout matrix polynomial in* (3.3) *with $\|p\|_\infty = \|q\|_\infty = 1$, and suppose $y_* \in [-1, 1]$ is a simple eigenvalue of $B(y)$. Then the absolute condition number $\kappa(y_*, B)$ as defined in* (5.3) *is*

$$\kappa(y_*, B) = \frac{\|v\|_2^2}{|\det J|}, \tag{5.4}$$

where $J = \left[\begin{smallmatrix} \partial_x p & \partial_y p \\ \partial_x q & \partial_x q \end{smallmatrix}\right]$ is the Jacobian and v is in Vandermonde form $v = [T_0(x_*), \ldots, T_{N-1}(x_*)]^T$. The condition number $\kappa(y_*, B)$ satisfies

$$\frac{1}{2} \frac{\|J^{-1}\|_2^2}{\kappa_2(J)} \leq \kappa(y_*, B) \leq 2N \frac{\|J^{-1}\|_2^2}{\kappa_2(J)}, \tag{5.5}$$

where $\kappa_2(J) = \|J\|_2 \|J^{-1}\|_2$, and

$$\frac{\kappa_*}{\|\mathrm{adj}(J)\|_2} \leq \kappa(y_*, B) \leq \frac{\kappa_* N}{\|\mathrm{adj}(J)\|_2}, \tag{5.6}$$

where κ_* is as in (5.3) *and* $\mathrm{adj}(J) = \left[\begin{smallmatrix} \partial_y q & -\partial_y p \\ -\partial_x q & \partial_x p \end{smallmatrix}\right]$ *is the algebraic adjugate matrix of J.*

Proof. Since $B(y)$ is a Bézout matrix polynomial, the eigenvector corresponding to y_* is in Vandermonde form $v = [T_0(x_*), T_1(x_*), \ldots, T_{N-1}(x_*)]^T$ (see section 3.1). The first-order perturbation expansion of y_* when $B(y)$ is perturbed to $B(y) + \Delta B(y)$ is [41, Thm. 5]

$$\hat{y} = y_* - \frac{v^T \Delta B(y_*) v}{v^T B'(y_*) v}, \tag{5.7}$$

where the derivative in $B'(y_*)$ is taken with respect to y.

From (5.3) and (5.7) we can see that $\kappa(y_*, B) \leq \|v\|_2^2 / |v^T B'(y_*) v|$, and to show that this upper bound is attainable we take $\Delta B(y) = \epsilon v v^T t(y)$, where $t(y)$ is a scalar polynomial such that $\max_{y\in[-1,1]} |t(y)| = |t(y_*)| = 1$. To bound the numerator of $\|v\|_2^2 / |v^T B'(y_*) v|$, we use $1 \leq \|v\|_2 \leq \sqrt{N}$, which follows from the Vandermonde structure. To bound the denominator, we note that from (3.2) and (3.3) the term $v^T B(y) v$ can be interpreted as evaluation at $s = t = x_*$ of the Bézout function

$$\mathcal{B}(p, q) = \frac{p(s, y)q(t, y) - q(s, y)p(t, y)}{s - t}. \tag{5.8}$$

A convenient way to work out $v^T B'(y)v$ is by differentiating the Bézoutian (5.8) with respect to y. We then take the limit

$$v^T B'(y)v = \lim_{(s,t)\to(x_*,x_*)} \mathcal{B}(\partial_y p, q) + \mathcal{B}(p, \partial_y q),$$

which can be easily evaluated by using L'Hôspital's rule. Using also $p(x_*, y_*) = q(x_*, y_*) = 0$ we conclude that

$$|v^T B'(y_*)v| = |\partial_x p \partial_y q - \partial_y p \partial_x q| = |\det J|$$

and hence, $\kappa(y_*, B) = \frac{\|v\|_2^2}{|\det J|}$, yielding (5.4).

The Jacobian is a 2×2 matrix so $\frac{1}{|\det J|} = \frac{\|J^{-1}\|_\infty}{\|J\|_1}$, and $\frac{1}{2}\frac{\|J^{-1}\|_2}{\|J\|_2} \le \frac{\|J^{-1}\|_\infty}{\|J\|_1} \le 2\frac{\|J^{-1}\|_2}{\|J\|_2}$, and since $\frac{\|J^{-1}\|_2}{\|J\|_2} = \frac{\|J^{-1}\|_2^2}{\|J^{-1}\|_2\|J\|_2} = \frac{\kappa_*^2}{\kappa_2(J)}$ we obtain (5.5). The inequalities (5.6) follow from $J^{-1} = \frac{1}{\det J}\begin{bmatrix} \partial_y q & -\partial_y p \\ -\partial_x q & \partial_x p \end{bmatrix} = \frac{1}{\det J}\operatorname{adj}(J)$.

□ □ The condition analysis and estimates in Theorem 5.1 appear to be the first in the literature for the Bézout polynomial eigenproblem $B(y)$. Importantly, they reveal situations where the Bézout approach can worsen the conditioning: the eigenvalues of $B(y)$ provide stable solutions if $|\det J|^{-1}$ is of size $\mathcal{O}(\|J^{-1}\|_2)$, but not if it is much larger.

Specifically, (5.5) shows that unless J is ill-conditioned, $\kappa(y_*, B)$ can be as large as the *square* of the original conditioning κ_*, and the Bézout approach may result in solutions with errors as large as $\mathcal{O}(\kappa_*^2 u)$ and may miss solutions with $\kappa_* > \mathcal{O}(u^{-1/2})$. The inequalities (5.6) show that $\kappa(y_*, B) \gg \kappa_*$ if $\|\operatorname{adj}(J)\|_2 \ll 1$, that is, if the derivatives of p and q are small at (x_*, y_*).

It is worth noting that the quantity $|\det J| = |\partial_x p \partial_y q - \partial_y p \partial_x q|$ in (5.4) does not change if we swap the roles of x and y and consider $B(x)$ instead of $B(y)$. Thus the conditioning of the Bézoutian matrix polynomial cannot be improved by swapping x and y, and the decision to solve for x or y first can be based solely on the size of the generalized eigenvalue problems (see Table 3.1).

5.1. Improving accuracy to $\mathcal{O}(\|J^{-1}\|_2 u)$. The preceding discussion suggests that the Bézout resultant approach can worsen the conditioning of the problem and hence, may give inaccurate or miss solutions. We overcome this by a local refinement and detecting ill-conditioned regions.

Local Bézoutian refinement. After the initial computation, we employ a local Bézoutian refinement, which reruns our algorithm in a small region containing (x_*, y_*), where the polynomials are both small.

Suppose that we work in a rectangular domain $\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. For simplicity assume that $|\Omega| = x_{\max} - x_{\min} \approx y_{\max} - y_{\min}$ with $|\Omega| \ll 1$. Since p and q are polynomials, Ω can be taken to be small enough so that $\|p\|_\Omega \|q\|_\Omega = \mathcal{O}(|\Omega|^2 \|\nabla p\|_2 \|\nabla q\|_2)$. In our code, and for simplicity of analysis, we map Ω to $[-1, 1] \times [-1, 1]$ via the linear transformations

$$x \leftarrow \frac{x - \frac{1}{2}(x_{\max} + x_{\min})}{\frac{1}{2}(x_{\max} - x_{\min})}, \quad y \leftarrow \frac{y - \frac{1}{2}(y_{\max} + y_{\min})}{\frac{1}{2}(y_{\max} - y_{\min})}. \tag{5.9}$$

A normwise backward stable algorithm for the polynomial eigenvalue problem $B(y)$ gives solutions with backward error $\mathcal{O}(u \max_i \|A_i\|_2)$, where A_i are the coefficient matrices of $B(y)$ in (3.3). Then $\|A_i\|_2$ are of size $\mathcal{O}(\|p\|_\Omega \|q\|_\Omega)$ and hence,

$\mathcal{O}(|\Omega|^2\|\nabla p\|_2\|\nabla q\|_2)$ because A_i are the coefficients of the bivariate polynomial (3.2). Therefore the backward error resulting from solving $B(y)$ is of size $\mathcal{O}(u|\Omega|^2\|\nabla p\|_2\|\nabla q\|_2)$.

The mapping (5.9) results in modified gradients such that $\|\nabla p\|_\Omega = \mathcal{O}(|\Omega|\|\nabla p\|_2)$ and $\|\nabla q\|_\Omega = \mathcal{O}(|\Omega|\|\nabla q\|_2)$. Thus, by (5.4), assuming that J is not too ill-conditioned, the conditioning is $\kappa_\Omega(y_*, B) = \mathcal{O}(1/|\Omega|^2\|\nabla p\|_2\|\nabla q\|_2)$.

The error in the computed solution is bounded by $\mathcal{O}(u)$(conditioning)·(backward error), so we conclude that the error resulting from solving the polynomial eigenvalue problem $B(y)$ is

$$\mathcal{O}(\kappa_\Omega(y_*, B)u|\Omega|^2\|\nabla p\|_2\|\nabla q\|_2) = \mathcal{O}(u).$$

This corresponds to error of size $\mathcal{O}(u|\Omega|)$ when we map x and y back to $[-1, 1]\times[-1, 1]$ by the inverse transformation of (5.9).

In addition, approximating p and q in Ω results in backward errors $\mathcal{O}(u\|p\|_\infty)$ and $\mathcal{O}(u\|q\|_\infty)$, respectively, in the polynomials themselves, so by (5.1) the overall error is $\mathcal{O}(u\|J^{-1}\|_2)$, reflecting the conditioning of the original problem. Hence, the computed solutions are computed stably.

The condition on Ω can relaxed to the weaker requirement $\|p\|_\Omega\|q\|_\Omega = \mathcal{O}(|\Omega|\|J^{-1}\|_2\|\nabla p\|_2\|\nabla q\|_2)$ to arrive at the same conclusion that the error is $\mathcal{O}(u\|J^{-1}\|_2)$, as can be verified by the same argument as above.

The crux of the discussion is that although the conditioning of the Bézout approach (5.4) can be worse than the original conditioning κ_*, by working in a local domain we reduce $\|A_i\|_2$, thereby reducing the backward error to obtain a stable solution with respect to the original problem.

Detecting ill-conditioned region and rerunning. Solutions can be missed if $\kappa(y_*, B) > \mathcal{O}(u^{-1})$, and this is unacceptable if the original conditioning is $\kappa_* \ll \mathcal{O}(u^{-1})$. Since $\mathcal{O}(\|\operatorname{adj}(J)\|_2)$ is $\mathcal{O}(\|J\|_2)$, the estimate (5.6) shows that $\kappa(y_*, B) \gg \kappa_*$ if and only if $\|J\|_\infty$ is small, and since $p(x_*, y_*) = q(x_*, y_*) = 0$, this implies that p and q are small near (x_*, y_*). To detect ill-conditioned regions in which solutions might have been missed by the initial Bézoutian method, we sample p, q, and J on an equispaced grid and find locations in which $|p|, |q| \le \mathcal{O}(u^{1/2})$ and $|\det J| \le \mathcal{O}(u)$. If such points exist, we identify the rectangular subdomain(s) that contain the points and rerun the Bézoutian method there. In each subdomain the polynomials are small with $|p|, |q| \le \mathcal{O}(u^{1/2})$ and hence, by the argument above, solutions with $\kappa_* \ll \mathcal{O}(u^{-1})$ are not missed and are computed with accuracy $\mathcal{O}(\|J^{-1}\|_2 u)$.

We note that it is easy to construct low degree polynomials p and q so that $\|J^{-1}\|_2 \gg u^{-1}$ at a solution (x_*, y_*), and such an ill-conditioned example cannot be solved accurately in double precision.

Sylvester resultant matrix. The Sylvester resultant matrix could be used to replace the Bézout resultant, and our experiments suggest that the Sylvester resultant can be better conditioned when $\kappa(y_*, B) \gg \kappa_*$ and J is well-conditioned. However, we observe the Sylvester resultant can have numerical difficulties when many solutions align along one coordinate direction, whereas the Bézout resultant does not. Since the Sylvester matrix is not symmetric its left eigenvector is nontrivial and so is more difficult to analyze. For these reasons, we decided to select the Bézout resultant. Moreover, the Bézout resultant is symmetric and this can be exploited in a regularization step.

6. Bézoutian regularization. The resultant, $\det(B(y))$, is zero in exact arithmetic at precisely the y-values of the common zeros of p and q, which includes those not in $[-1, 1]$. However, problematically, $B(y)$ can be numerically singular, i.e.,

$\|B(y)\|_2\|B(y)^{-1}\|_2 \geq 10^{15}$, for many values of y. Hence, a backward stable eigensolver, such as the QZ algorithm applied to a linearization of $B(y)$, can give spurious eigenvalues of $B(y)$ anywhere in the complex plane [4, Sec 8.7.4], and these are known to also cause catastrophic ill-conditioning of the other eigenvalues [35, Ch. 13]. As a consequence the computed solutions can be inaccurate or spurious, and as a remedy we apply a regularization step to $B(y)$.

6.1. Numerical singularity of Bézout matrices. The functions $f(x, y)$ and $g(x, y)$ are assumed to be smooth, and therefore, their polynomial interpolants p and q, typically, have tensor Chebyshev expansions as in (1.2) with coefficient matrices P and Q that have rapidly decaying entries. Hence, the polynomials $p_y(x)$ and $q_y(x)$ have rapidly decaying Chebyshev coefficients, and the Bézout matrix $B(y_0)$, for any $y_0 \in [-1, 1]$, inherits a similar decay as P and Q through its definition (3.2). This can also be seen from the fact that the Bézout function (3.2) is a bivariate polynomial and the entries of $B(y_0)$ are the coefficients in its Chebyshev expansion. In this section $B(y)$ always denotes the matrix polynomial, whereas $B(y_0)$ is its evaluation (a symmetric matrix) at y_0, which is any fixed value in $[-1, 1]$.

The decaying entries of $B(y_0)$ mean its entries in the last column are $\mathcal{O}(u)$ or smaller in magnitude, where $u \approx 1.1\times10^{-16}$ is the unit roundoff, and the last canonical vector, $e_N = [0, \ldots, 0, 1]^T$, is numerically an eigenvector with zero eigenvalue. Similar reasoning indicates that all the canonical vectors $e_N, e_{N-1}, \ldots, e_{N-k+1}$ for some $1 \leq k < N$ are also numerically nearly an eigenvector with zero eigenvalue. Hence, an approximate null space of $B(y)$ is approximately S, where

$$S = [e_{N-k+1}|e_{N-k+2}|\ldots|e_N], \qquad 1 \leq k < N.$$

This argument makes no reference to y_0, and so we have $\|B(y_0)S\|_2 \ll \|B(y_0)\|_2$ for any $y_0 \in [-1, 1]$. Thus the matrix polynomial $B(y)$ is close to singular. Note that such eigenpairs of $B(y_0)$ have nothing to do with the eigenpairs of the matrix polynomial $B(y)$. By contrast, at the eigenvalues $y_* \in [-1, 1]$ of $B(y)$, $B(y_*)$ has a nontrivial null space with a null vector in Vandermonde form $[T_0(x_*), \ldots, T_{N-1}(x_*)]^T$, which coincides with the eigenvector of the matrix polynomial $B(y)$ at the eigenvalue y_*. Consequently, $B(y_*)$ has a null vector numerically far from span(S), which the numerical null space of $B(y_0)$ does not contain for other values of y_0.

6.2. Regularization details. First we partition $B(y)$ into four parts,

$$B(y) = \begin{bmatrix} B_1(y) & E(y)^T \\ E(y) & B_0(y) \end{bmatrix}, \tag{6.1}$$

where $B_0(y)$ and $E(y)$ are $k\times k$ and $k\times(N-k)$, respectively, and choose k so that the $(N-k)\times(N-k)$ matrix polynomial $B_1(y)$ is numerically nonsingular. We choose k to be the largest integer so that the following conditions are satisfied for any $y_0 \in [-1, 1]$:

- $\|B_0(y_0)\|_2 = \mathcal{O}(u)$;
- $\|E(y_0)\|_2 = \mathcal{O}(u^{1/2})$.

Note that although the bottom-right part of $B(y)$ is of size $\mathcal{O}(u)$, $E(y)$ is typically not of that size; but we can still justify working with $B_1(y)$ by showing the eigenvalues of $B_1(y)$ in $y \in [-1, 1]$ are numerically close to the eigenvalues of $B(y)$. For our analysis the symmetry of $B(y)$ is crucial, and this is another reason we use Bézoutians instead of the Sylvester matrix.

Recall that the eigenvalues of a regular matrix polynomial $B(y)$ are the values y_* for which the matrix $B(y_*)$ has a zero eigenvalue. We show that for any eigenvalue $y_* \in [-1, 1]$ of $B(y)$, the matrix $B_1(y_*)$ with k chosen as above is nearly singular, that is, $B_1(y_*)$ has an eigenvalue $\mathcal{O}(u)$. Numerically, this means that y_* can be computed stably via the regularized polynomial eigenvalue problem $B_1(y)$.

We first argue that for any given $y_0 \in [-1, 1]$, the matrix $B(y_0)$ has at least k eigenvalues of size $\mathcal{O}(u)$. One way to see this is to introduce

$$\widetilde{B}(y) = \begin{bmatrix} B_1(y) & E^T(y) \\ E(y) & 0 \end{bmatrix},$$

and note that by Weyl's theorem, the matrix $\widetilde{B}(y_0)$ has eigenvalues within $\|B_0(y)\|_2 = \mathcal{O}(u)$ of those of $B(y_0)$. Moreover, by [30, Thm. 3.1], $\widetilde{B}(y)$ has at least k eigenvalues that match those of $-E(y_0)^T B_1(y_0)^{-1} E(y_0)$ up to $\mathcal{O}(\|E(y_0)\|_2^4) = \mathcal{O}(u^2)$. Hence, it suffices to show that $\|E(y_0)^T B_1(y_0)^{-1} E(y_0)\|_2 = \mathcal{O}(u)$. Unfortunately, the bound $\|E^T(y_0) B_1(y_0)^{-1} E(y_0)\|_2 \leq \|E(y_0)\|_2^2 \|B_1(y_0)^{-1}\|_2$ is insufficient because $\|B_1(y_0)^{-1}\|_2$ can be large.

To see why we can nonetheless expect $\|E(y_0)^T B_1(y_0)^{-1} E(y_0)\|_2 = \mathcal{O}(u)$, consider the LDL^T factorization $LDL^T = B_1(y_0)$, where L is unit lower triangular. For notational simplicity, we write L and D instead of $L(y_0)$ and $D(y_0)$. We claim that the L factor has decaying off-diagonal elements, and D has decaying diagonals[4]. To see this, consider the first step in performing the LDL^T factorization. The first column of L is parallel to that of $B_1(y_0)$, which has decaying coefficients. The Schur complement is the $(n-1) \times (n-1)$ matrix $B_{1,1}(y_0) - \ell d_1 \ell^T$, where $B_{1,1}(y_0)$ is the lower-right part of $B_1(y_0)$, and $\ell \in \mathbb{R}^{n-1}$ is the bottom part of L's first column. Now since ℓ has decaying coefficients, so does the matrix $\ell d_1 \ell^T$ and hence, the decay property is inherited by $B_{1,1}(y_0) - \ell d_1 \ell^T$. The rest of the LDL^T factorization performs the same operation on $B_{1,1}(y_0) - \ell d_1 \ell^T$, so the claim follows by repeating the argument, observing that the diagonals of D become smaller as the factorization proceeds.

We have $B_1^{-1} = L^{-T} D^{-1} L^{-1}$ so $E^T B_1^{-1} E = E^T L^{-T} D^{-1} L^{-1} E$, and since the elements of E decay towards the bottom-right corner, so do those of $L^{-1}E$. On the other hand, the diagonals of D^{-1} grow towards the bottom-right, and the large elements of D^{-1} are multiplied by the small elements of E, so that $\|E^T B_1^{-1} E\|_2 \ll \|E\|^2 \|B_1^{-1}\|_2$. Indeed, in practice, we generally observe that $\|E^T B_1^{-1} E\|_2 = \mathcal{O}(\|E\|_2^2) = \mathcal{O}(u)$.

The remaining task is to show that $B_1(y_*)$ has an eigenvalue of size $\mathcal{O}(u)$ if the numerical null space of $B(y_*)$ has dimension $k + 1$. One observation is that the eigenvalues of Hermitian matrices partitioned into 2×2 blocks as in (6.1) match those of the diagonal blocks to within the norm of the off-diagonal *squared*, divided by the *gap* between the spectra [29]. However, this is not sufficient to show that the small eigenvalues of B_1 do not get perturbed by E since the gap is small for such eigenvalues.

To show that $B_1(y_*)$ has an eigenvalue of size $\mathcal{O}(u)$, we invoke the Cauchy interlacing theorem [35, Ch. 10]: arranging the eigenvalues of $B_1(y_0)$ and $B(y_0)$ in nondecreasing order,

$$\lambda_i(B_1(y_0)) \leq \lambda_i(B(y_0)).$$

[4]Strictly speaking, D needs to be allowed to have 2×2 blocks, since an LDL^T factorization with D diagonal may not exist, as the example $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ illustrates. It is possible to extend the argument to such cases, but most symmetric matrices do permit D to be diagonal, and our purpose is to explain the behavior observed in practice.

The reverse identity also holds: the ith largest eigenvalue of $B(y_0)$ is at least as large as that of $B_1(y_0)$. Consequently, an eigenvalue λ_i of $B_1(y_0)$ with $|\lambda_i| > \|B_0(y_0)\|_2$ can only increase in absolute value by E. Since $\|B_0(y_0)\|_2 = \mathcal{O}(u)$, this means that if $B(y_*)$ has an additional null space relative to $B(y_0)$ at other values of $y_0 \in [-1, 1]$, then $B_1(y_*)$ must have an eigenvalue $\mathcal{O}(u)$. This shows that the y-values for which B has an extra null space can be reliably detected via solving a polynomial eigenvalue with respect to the smaller matrix polynomial $B_1(y)$.

The above argument holds only if the coefficients of $B(y)$ have the decaying property inherited from that of p and q. When the coefficients of p and q do not decay the two conditions $\|B_0(y)\|_2 = \mathcal{O}(u)$ and $\|E(y)\|_2 = \mathcal{O}(u^{1/2})$ are not satisfied, and we proceed to solve the polynomial eigenvalue problem for $B(y)$ without regularization. This does not cause difficulties because then $B(y)$ is not numerically singular.

Note that our regularization is *not* equivalent to cutting off the high-degree terms in $p(x, y)$ and $q(x, y)$; instead, it corresponds to cutting off high-degree terms in the Bézoutian (3.2). We also note that although the initial motivation for regularizing $B(y)$ was for improved stability, it also results in improved efficiency because the polynomial eigenvalue problem becomes smaller in size and lower in degree.

7. Further implementation details. Some implementation details remain and we discuss them in this section.

Construction of Bézout matrices. Our construction of (Chebyshev) Bézout matrices is based on the MATLAB code given in [42], which exploits a connection between Bézout matrices and the block symmetric linearization of scalar and matrix polynomials. We have improved the efficiency of the code by restricting it to only construct Chebyshev Bézout matrices. More specifically, in [42] it was shown that if $u(x)$ is of degree d and $v(x)$ is of degree at most $d - 1$, then the $d \times d$ symmetric pencil introduced in [31] of the form $\lambda X + Y$ is such that X is the Bézout matrix of u and v. Therefore, the Bézout matrix $B(y_0)$ can be obtained from $p_{y_0}(x)$ and $q_{y_0}(x)$ in (3.1), attaching zero leading coefficients if necessary, by forming the matrix X in the symmetric pencil $\lambda X + Y$. This provides the ability to compute $B(y_0)$ for any $y_0 \in [-1, 1]$.

The coefficient matrices A_i in $B(y) = \sum_{i=0}^{m_{p_s}+m_{q_s}} A_i T_i(y)$ are obtained by first sampling $B(y)$ at $m_{p_s} + m_{q_s} + 1$ Chebyshev points in the subdivided y-interval, then converting to coefficient space via the fast Fourier transform, applying the transform entrywise. Here m_{p_s} and m_{q_s} are the degrees of p and q in y in the subdivided domain.

Finding the eigenvalues of $B(y)$ via linearization. Once the Chebyshev coefficients are obtained, we solve the polynomial eigenvalue problem via the standard approach of linearization. As in Chebfun, we use the colleague matrix pencil [45, Ch. 18] for matrix polynomials, the standard companion-like linearization for polynomials expressed in the Chebyshev basis.

After forming the coefficient matrices of $B(y)$, for efficiency we remove the leading coefficients with Frobenius norms smaller than $u \max_i \|A_i\|_F$, where A_i are as in (3.3). The size of the pencil is (dimension of $B_1(y)$)·(degree of $B_1(y)$) as summarized in Table 3.1, but regularization and truncation can reduce the size.

The colleague matrix pencil for a matrix polynomial $P(\lambda) = \sum_{i=0}^{n} A_i T_i(\lambda)$ is

$$\lambda X + Y = \lambda \begin{bmatrix} A_n & & & \\ & I_n & & \\ & & \ddots & \\ & & & I_n \end{bmatrix} - \frac{1}{2} \begin{bmatrix} -A_{n-1} & I_n - A_{n-2} & -A_{n-2} & \cdots & -A_0 \\ I_n & 0 & I_n & & \\ & \ddots & \ddots & \ddots & \\ & & I_n & 0 & I_n \\ & & & 2I_n & 0 \end{bmatrix}.$$

The eigenvalues of the matrix polynomial $P(\lambda)$ match those of the matrix pencil $\lambda X + Y$, which can be computed by the QZ algorithm [20, Ch. 7].

Univariate rootfinding. Once we have found the y-values of the solutions we then find the x-values by a univariate rootfinding algorithm based on computing the eigenvalues of the colleague matrix [45], and using the `eig` command in MATLAB.

We compute the roots of both $p(x, y_*)$ and $q(x, y_*)$ on the subdivided x-interval separately, and test if both functions are small enough at the roots and discard those that fail this test. The function value tolerance depends on the localization level of the algorithm, as we explain shortly. We then merge the x-values that are less than $\mathcal{O}(u)$ apart, as common roots can be double-counted by this process. Due to subdivision, the polynomials are of degree less than 16 on each subregion, and computing their roots is negligible in cost.

Local Bézoutian refinement. This component is crucial to the success of the algorithm. After computing the approximate solutions via the initial Bézoutians we further employ highly localized Bézoutians near the solutions for improved stability and accuracy. Specifically, we

1. Group the computed zeros into clusters, each of whose members are within $\mathcal{O}(u^{1/4})$ in Hausdorff distance; and
2. For each cluster, execute another Bézout-based rootfinder in a small domain of width $\mathcal{O}(u^{1/4})$ that contains the cluster.

The distance $u^{1/4}$ was chosen so as to be small enough not to contain too many solutions, and large enough to accommodate the errors in the initially computed solutions. The local Bézoutian refinement is beneficial to prevent worsening of the conditioning (see section 5), and here we argue that the local refinement is also needed because the initial solution may contain spurious, multiply-counted, or inaccurate zeros. The task of the initial global Bézoutian is to obtain an estimate of the solutions that are allowed to have error larger than $\mathcal{O}(u)$, but must not be missed. Hence, at first we accept x-values with $|f(\hat{x}_*, \hat{y}_*)|, |g(\hat{x}_*, \hat{y}_*)| \leq \mathcal{O}(u^{1/2})$, which does not remove those corresponding to near-multiple or ill-conditioned solutions, and then during the local refinement we do a more stringent test requiring $|f|$ and $|g|$ to be $\mathcal{O}(u)$.

Since at the local level the domain is much smaller than the original, the polynomial interpolants of f and g are of very low degree and hence, the overhead in cost is marginal.

Sometimes the local domain is so small that one of the polynomials, say p, is numerically constant in one (or both) variable, say x. In such cases we trivially find the roots y_* of $p(y)$ and compute the roots of $q(x, y_*)$ on the local interval.

Local refinement usually results in a significant improvement in accuracy as discussed in section 5. Moreover, the low-degree polynomials result in a Bézout matrix polynomial $B(y)$ that is far from singular, and so its numerical solution can be carried out stably.

The refinement is vital when many solutions exist with nearly the same value in the first coordinate y. Specifically, suppose that $(x_i, y_0+\delta_i)$ for $i = 1, \ldots, k$ are simple zeros of f and g, where $|\delta_i|$ are small. Then the computed eigenvalues of the Bézout matrix polynomial take many (at least k) values close to y_0. The algorithm then finds the corresponding x-values via a univariate rootfinder, but this process faces difficulty, as for each computed $y_0 + \delta_i$, the two functions $f(x, y_0 + \delta_i)$ and $g(x, y_0 + \delta_i)$ have k nearly multiple zeros, and this can result in counting the same zero multiple times. The local Bézout refinement resolves this by regarding such multiply counted zeros as a cluster and working in a domain that contains very few common zeros.

Spurious zeros and their removal. One difficulty with resultant-based bivariate rootfinders is *spurious zeros*, which are y-values for which the resultant is numerically singular, but p and q do not have a common zero. There are several ways spurious zeros can arise: (i) A shared zero at infinity: if $\deg p > \deg q$ in x then there is a spurious zero at values of y for which the leading coefficient of p is zero; (ii) An exact nonreal common zero, or an exact real common zero lying outside the domain; and (iii) Numerical artifacts that arise due to the ill-conditioning of $B(y)$. Fortunately, these can all be detected in the local refinement stage, where p and q are of low degree.

8. Numerical examples. In this section we present six examples to illustrate the accuracy and robustness of our algorithm and its ability to solve problems with very high polynomial degree. Throughout, the zero contours of f and g in (1.1) are drawn as blue and red curves, respectively, computed by the command `roots(f)` in Chebfun2 [43]. The black dots are the solutions computed by our algorithm described in this paper as realized by the command `roots(f,g)` in Chebfun2.

Example 1 (Coordinate alignment). This example is of small degree, with the functions f and g being approximated by polynomial interpolants of degrees $(m_p, n_p, m_q, n_q) = (20, 20, 24, 30)$:

$$\begin{pmatrix} T_7(x)T_7(y)\cos(xy) \\ T_{10}(x)T_{10}(y)\cos(x^2y) \end{pmatrix} = 0, \qquad (x, y) \in [-1, 1] \times [-1, 1]. \tag{8.1}$$

The common simple zeros align along both coordinate directions, but this does not reduce the accuracy of the resultant method, despite $B(y)$ having multiple eigenvalues (see section 3). The solutions to (8.1) are known exactly and the maximum absolute error in the computed solutions is 8.88×10^{-16}. In Figure 8.1 we show the zero contours and common zeros for (8.1). The zero contour lines quadratically cluster near the edge of the domain following the distribution of the roots of Chebyshev polynomials.

Example 2 (Face and apple). Here we select functions f and g that are exactly polynomials, i.e., $f = p$ and $g = q$, with zero contours suggesting a face and an apple, respectively. These bivariate polynomials were taken from [37], and the degrees are $(m_p, n_p, m_q, n_q) = (10, 18, 8, 8)$. Note that in this example we have taken the domain $[-2, 2] \times [-1.5, 4]$.

This example is of mathematical interest because both polynomials are relatively small $\leq 10^{-5}$ near the origin where the Bézout resultant can severely worsen the condition number of the solutions. Such solutions can be initially missed, and to recover them the code detects ill-conditioned subdomains and reruns the Bézoutian there (see section 5.1).

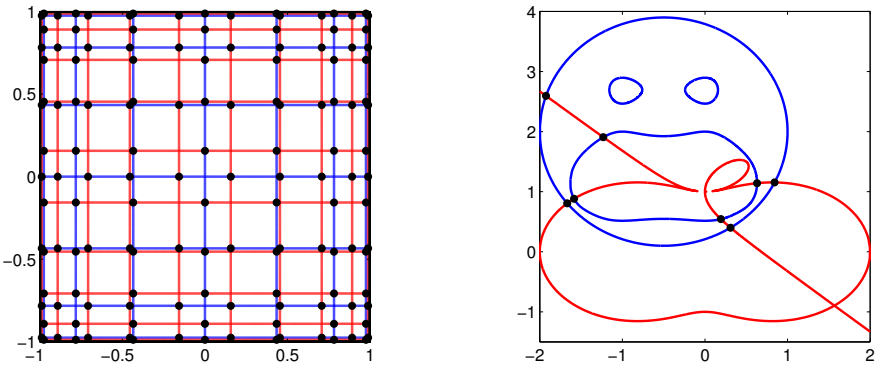

Fig. 8.1: The zero contours of f (red) and g (blue), and the common zeros (black dots) computed with our algorithm. Left: Coordinate alignment (8.1). Simple common zeros are aligned with the coordinate directions, but this causes no numerical difficulties (see section 3). Right: Face and apple. Even though the polynomials are small near the origin we can detect this and use a local Bézout refinement to obtain accurate solutions (see section 7).

Example 3 (Hadamard). Chebfun2 allows us to construct polynomials from interpolation data at a tensor Chebyshev grid [43], and for this example we take the interpolation data to be the Hadamard matrices H_{32} and H_{64} of size 32×32 and 64×64, i.e., we solve (1.1), where $f(x_i, x_j) = H_{32}(i, j)$, $g(x_i, x_j) = H_{64}(i, j)$, and x_i are Chebyshev points. The Hadamard matrices contain ± 1 entries and therefore, the constructed polynomials $p = f$ and $q = g$ (of degrees $(m_p, n_p, m_q, n_q) = (31, 31, 63, 63)$) have many zero contours and common zeros. Our algorithm requires 89 seconds and the maximum of $|f(x_*, y_*)|$ and $|g(x_*, y_*)|$ over all computed solutions (x_*, y_*) is 3.98×10^{-13}.

In this example subdivision does not lead to an efficient reduction in the degree, and τ, as defined in section 4, is estimated to be 0.82 for f and 0.75 for g. We obtained the estimates for τ by subdividing the domain into $2^5 \times 2^5$ subdomains and taking the average degree reduction.

Example 4 (Travelling waves). Here we choose functions in (1.1) so that f and g oscillate in x and y, respectively:

$$\begin{pmatrix} \sin(\omega x - y/\omega) + y \\ \sin(x/\omega - \omega y) - x \end{pmatrix} = 0, \quad (x, y) \in [-1, 1] \times [-1, 1], \qquad \omega \in \mathbb{R}, \tag{8.2}$$

where increasing ω makes the problem of higher degree, and for Figure 8.2 we have selected $\omega = 30$. The degrees of the polynomial interpolants p and q are $(m_p, n_p, m_q, n_q) = (7, 63, 62, 6)$. Note that f requires many subdivisions in x, and g requires many in y to reduce the degree. However, since f and g must be subdivided on the same grids we actually subdivide f and g many times in both directions. Figure 8.2 shows the zero contours and solutions for (8.2) for $\omega = 30$. Our algorithm required 11.0 seconds and the maximum of $|f(x_*, y_*)|$ and $|g(x_*, y_*)|$ over all computed solutions (x_*, y_*) is 1.38×10^{-13}.

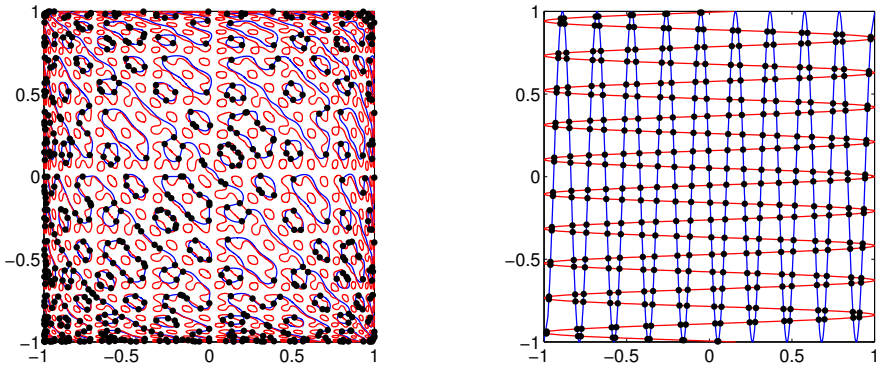

Fig. 8.2: Left: Hadamard example. For this example significant subdivision is required as subdividing does not reduce the degree very effectively. Right: Travelling waves (8.2). For large ω this example is difficult for contouring algorithms as the zero curves become oscillatory.

In this example the τ estimates are both about 0.72 in f and g and hence, the estimated exponent of the complexity is $-\frac{\log 4}{\log \tau} = 4.2$.

Example 5 (Airy and Bessel functions). In this example we choose the following problem:

$$\begin{pmatrix} \mathrm{Ai}(-13(x^2y + y^2))) \\ J_0(500x)y + xJ_1(500y) \end{pmatrix} = 0, \tag{8.3}$$

where Ai is the Airy function and J_0 and J_1 are Bessel functions of the first kind. We have selected these functions because they require very high degree polynomial interpolants, $(m_p, n_p, m_q, n_q) = (171, 120, 569, 568)$, to be approximated to machine precision. Our code computed the 5932 common zeros in 501 seconds, and we independently verified that our algorithm found all the solutions to (8.3) by using a method based on a contouring algorithm with significant domain subdivision.

In this example the τ estimates are both 0.59 in f and g and hence, the estimated exponent is $-\frac{\log 4}{\log \tau} = 2.6$.

Example 6 (SIAM 100-Digit Challenge problem). An article in SIAM News in 2002 set a challenge to solve ten problem to ten digits (the solution to each problem was a single real number) [44]. One of these problems was to find the global minimum of the following complicated and highly oscillatory function:

$$\begin{aligned} f(x,y) = &\left(\frac{x^2}{4} + e^{\sin(50x)} + \sin(70\sin(x))\right) + \left(\frac{y^2}{4} + \sin(60e^y) + \sin(\sin(80y))\right) \\ &- \cos(10x)\sin(10y) - \sin(10x)\cos(10y). \end{aligned} \tag{8.4}$$

Since all the local extrema of (8.4) satisfy $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = 0$ we can solve this problem by finding all the local extrema before selecting the global minimum from them. A simple argument shows that the global minimum occurs in the domain $[-1, 1] \times [-1, 1]$

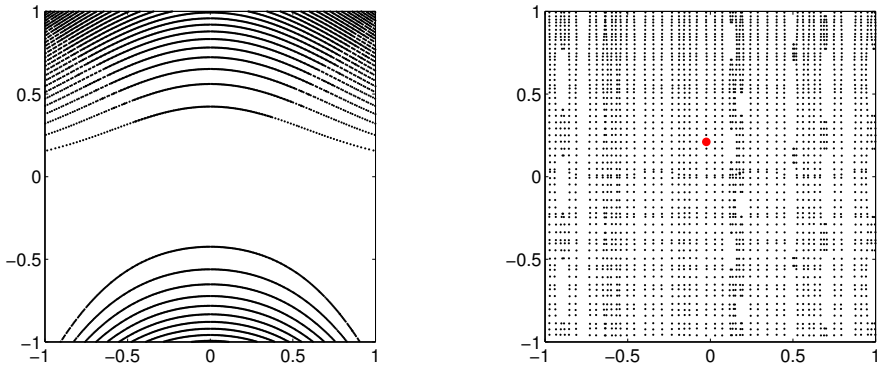

Fig. 8.3: Left: Solutions to (8.3). The number of isolated solutions is 5932; there are so many that they appear as lines rather than dots. Right: SIAM digit problem (8.4). The zeros of f_x and f_y are shown as black dots and the location of the minimum is a red dot. Currently, we consider these examples as of very high degree, and present them as an indication of the robustness of our algorithm.

[11]. We approximate f on $[-1,1]\times[-1,1]$ and note that the partial derivatives of f are of numerical degrees 625 and 901 in x and y, respectively, i.e., $(m_p, n_p, m_q, n_q) = (901, 625, 901, 625)$, and we currently consider this of very high degree. Nevertheless, our algorithm computes all 2720 local extrema of (8.4) in 257 seconds and obtains the global minimum to an accuracy of 1.12×10^{-15}. The function (8.4) does have some structure because it is a low rank function [43], but this structure is not directly exploited in the rootfinding algorithm. In Figure 8.3 we show the location of the 2720 local extrema and plot the one corresponding to the location of the global minimum as a red dot. In this example the τ estimates are both about 0.53 in f and g and hence, the estimated exponent of the complexity is $-\frac{\log 4}{\log \tau} = 2.2$.

Finally, to test the code with examples of even higher degree, we doubled all the coefficients inside the trigonometric functions in (8.4), e.g., $e^{\sin(50x)} \leftarrow e^{\sin(100x)}$, $\sin(70\sin(x)) \leftarrow \sin(140\sin(x))$, and ran the same experiment. The partial derivatives of this f are now of numerical degrees $(m_p, n_p, m_q, n_q) = (1781, 1204, 1781, 1204)$. Our code computed 9318 local extrema in 1300 seconds, and the estimated complexity exponent is 2.1.

Acknowledgements. The third author had a personal discussion with John Boyd, in which he suggested a contouring algorithm. We are also grateful for a fruitful discussion with Rob Corless. We would like to thank Nick Higham, Françoise Tisseur, and Nick Trefethen for their financial and intellectual support throughout the collaboration for this paper.

REFERENCES

[1] D. A. Aruliah, R. M. Corless, L. Gonzalez-Vega, and A. Shakoori. Geometric applications of the Bezout matrix in the Lagrange basis. In *Proceedings of the 2007 international workshop on Symbolic-numeric computation*, pages 55–64. ACM, 2007.

[2] J. Asakura, T. Sakurai, H. Tadano, T. Ikegami, and K. Kimura. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 1:52–55, 2009.

[3] F. V. Atkinson. *Multiparameter Eigenvalue Problems*. Academic Press, New York, NY, USA, 1972.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, PA, USA, 2000.

[5] S. Barnett. Greatest common divisors from generalized Sylvester resultant matrices. *Linear Multilinear Algebra*, 8:271–279, 1980.

[6] W.-J. Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra Appl.*, 436(10):3839–3863, 2012.

[7] É. Bézout. *Théorie Générale des Équations Algébriques*. PhD thesis, Pierres, Paris, 1779.

[8] D. A. Bini and L. Gemignani. Bernstein–Bezoutian matrices. *Theor. Comput. Sci.*, 315(2):319–333, 2004.

[9] D. A. Bini and A. Marco. Computing curve intersection by means of simultaneous iterations. *Numer. Algorithms*, 43:151–175, 2006.

[10] D. A. Bini and V. Noferini. Solving polynomial eigenvalue problems by means of the Ehrlich-Aberth method. to appear in Linear Algebra Appl.

[11] F. Bornemann, D. Laurie, S. Wagon, and H. Waldvogel. *The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing*. SIAM, 2004.

[12] J. P. Boyd. Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding. *SIAM J. Numer. Anal.*, 40:1666–1682, 2002.

[13] J. P. Boyd. Computing real roots of a polynomial in Chebyshev series form through subdivision. *Appl. Numer. Math.*, 56:1077–1091, 2006.

[14] J. P. Boyd. Computing real roots of a polynomial in Chebyshev series form through subdivision with linear testing and cubic solves. *Appl. Math. Comput.*, 174:1642–1658, 2006.

[15] B. Buchberger. Introduction to Gröbner bases. In *Gröbner Basis and Applications*, volume 251, pages 3–31. Cambridge University Press, 1998.

[16] D. A. Cox, J. B. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, third edition, 2007.

[17] G. M. Diaz-Toca, M. Fioravanti, L. Gonzalez-Vega, and A. Shakoori. Using implicit equations of parametric curves and surfaces without computing them: Polynomial algebra by values. *Comput. Aided Geom. D.*, 30:116–139, 2013.

[18] I. Z. Emiris and B. Mourrai. Matrices in elimination theory. *J. Symbolic Comput.*, 28:3–44, 1999.

[19] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. SIAM, Philadelphia, USA, (unabridged republication of book first published by academic press in 1982) edition, 2009.

[20] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.

[21] C. G. A. Harnack. Über Vieltheiligkeit der ebenen algebraischen Curven. *Math. Ann.*, 10:189–199, 1876.

[22] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, USA, second edition, 2002.

[23] A. Hilton, A. J. Stoddart, J. Illingwort, and T. Windeatt. Marching triangles: range image fusion for complex object modelling. In *International Conference on Image Processing*, volume 1, 1996.

[24] M. E. Hochstenbach, T. Košir, and B. Plestenjak. A Jacobi-Davidson type method for the two-parameter eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 26(2):477–497, 2004.

[25] G. Jónsson and S. Vavasis. Accurate solution of polynomial equations using Macaulay resultant matrices. *Math. Comp.*, 74:221–262, 2005.

[26] D. Kapur and T. Saxena. Comparison of various multivariate resultant formulations. In *A. Levelt, ed., Proc. Int. Symp. on Symbolic and Algebraic Computation*, pages 187–194, Montreal, 1995.

[27] F. C. Kirwan. *Complex Algebraic Curves*. Cambridge University Press, 1992.

[28] N. Kravitsky. On the discriminant function of two commuting nonselfadjoint operators. *Integr. Equat. Oper. Th.*, 3:97–125, 1980.

[29] C.-K. Li and R.-C. Li. A note on eigenvalues of perturbed Hermitian matrices. *Linear Algebra Appl.*, 395:183–190, 2005.

[30] R.-C. Li, Y. Nakatsukasa, N. Truhar, and W. Wang. Perturbation of multiple eigenvalues of Hermitian matrices. *Linear Algebra Appl.*, 437:202–213, 2012.

[31] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann. Vector spaces of linearizations for matrix polynomials. *SIAM J. Matrix Anal. Appl.*, 28:971–1004, 2006.

[32] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves I: simple intersections. *ACM Trans. Graph.*, 13:73–100, 1994.

[33] R. J. Marks II. *Introduction to Shannon Sampling and Interpolation Theory*. Springer-Verlag New York, Inc., 1991.
[34] V. Mehrmann and H. Voss. Nonlinear eigenvalue problems: A challenge for modern eigenvalue methods. *Mitt. der Ges. fr Angewandte Mathematik and Mechanik*, 27:121–151, 2005.
[35] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, 1998.
[36] D. Plaumann, B. Sturmfels, and C. Vinzant. Computing linear matrix representations of Helton–Vinnikov curves. *Mathematical Methods in Systems, Optimization, and Control Operator Theory*, 222:259–277, 2012.
[37] M. Sagraloff et al. Gallery of algebraic curves and their arrangements. `http://exacus.mpi-inf.mpg.de/gallery.html`.
[38] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G. E. Stechert & Co., New York, 1885.
[39] A. J. Sommese and C. W. Wampler. *The Numerical Solution of Systems of Polynomials*. World Scientific, 2005.
[40] J.-G. Sun. On condition numbers of a nondefective multiple eigenvalue. *Numer. Math.*, 61:265–275, 1992.
[41] F. Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra Appl.*, 309:339–361, 2000.
[42] A. Townsend, V. Noferini, and Y. Nakatsukasa. Vector spaces of linearizations for matrix polynomials: a bivariate polynomial approach. Preprint, 2013.
[43] A. Townsend and L. N. Trefethen. An extension of Chebfun to two dimensions. Submitted, 2013.
[44] L. N. Trefethen. A hundred-dollar, hundred-digit challenge. *SIAM News*, 35, Jan/Feb 2002.
[45] L. N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, Philadelphia, 2013.
[46] L. N. Trefethen et al. Chebfun version 4.2. Software, 2011. The Chebfun Development Team.
[47] J. H. Wilkinson. The perfidious polynomial. In *G. H. Golub, ed., Studies in Numerical Analysis*. Math. Assoc. Amer., 1984.
[48] H. Xie and H. Dai. On the sensitivity of multiple eigenvalues of nonsymmetric matrix pencils. *Linear Algebra Appl.*, 374:143–158, 2003.