Numer. Math. (2015) 129:181–209
DOI 10.1007/s00211-014-0635-z

Numerische Mathematik

Computing the common zeros of two bivariate functions via Bézout resultants

Yuji Nakatsukasa · Vanni Noferini · Alex Townsend

Received: 19 May 2013 / Revised: 5 March 2014 / Published online: 9 May 2014

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 ($\geq 1{,}000$). We analyze the resultant method and its conditioning by noting that the Bézout matrices are matrix polynomials. Two implementations are available: one on the MATLAB Central File Exchange and another in the `roots` command in Chebfun2 that is adapted to suit Chebfun's methodology.

Mathematics Subject Classification (2000) 65D15 · 65F15 · 65F22

Yuji Nakatsukasa was partially supported by EPSRC grant EP/I005293/1.
Vanni Noferini was supported by ERC Advanced Grant MATFUN (267526).
Alex Townsend was supported by EPSRC grant EP/P505666/1 and the ERC grant FP7/2007-2013 to L. N. Trefethen.

Y. Nakatsukasa
Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Tokyo 113-8656, Japan
e-mail: nakatsukasa@mist.i.u-tokyo.ac.jp

V. Noferini (✉)
School of Mathematics, University of Manchester, Manchester M13 9PL, UK
e-mail: vanni.noferini@manchester.ac.uk

A. Townsend
Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK
e-mail: townsend@maths.ox.ac.uk

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 real pairs $(x, y) \in [-1, 1] \times [-1, 1]$ such that

$$\begin{pmatrix} f(x, y) \\ g(x, y) \end{pmatrix} = 0 \tag{1}$$

under the assumption that f and g are real-valued in $[-1, 1] \times [-1, 1]$ and 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 first step is to replace f and g in (1) by polynomial interpolants p and q, respectively, before finding the common zeros of the corresponding polynomial system. Throughout this paper we assume that f and g are smooth bivariate functions, and p and q are their polynomial interpolants, which approximate f and g to relative machine precision. The polynomials p and q are of the form:

$$p(x, y) = \sum_{i=0}^{m_p} \sum_{j=0}^{n_p} P_{ij} T_i(y) T_j(x), \qquad q(x, y) = \sum_{i=0}^{m_q} \sum_{j=0}^{n_q} Q_{ij} T_i(y) T_j(x), \tag{2}$$

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

$$\|f - p\|_\infty = \mathcal{O}(u)\|f\|_\infty, \quad \|g - q\|_\infty = \mathcal{O}(u)\|g\|_\infty, \tag{3}$$

where $\|f\|_\infty = \max_{x,y\in[-1,1]} |f(x, y)|$ and u is the unit roundoff.

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* [22] and a robust eigenvalue solver, such as the `eig` command in MATLAB. The remaining component is found by univariate rootfinding based on the *colleague matrix* [13]. 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 Bézout (and Sylvester) 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 [1,10,19,28,34]. 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 Sect. 4). Subdivision can allow us to compute the common zeros to (1) with polynomial interpolants of degree as large as 1,000, 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 the 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 often 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 proceed to solve the polynomial eigenvalue problem by linearization, which avoids the interpolation of highly oscillatory determinants of resultants, 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 one component of the solutions to $p(x, y) = q(x, y) = 0$, and a second eigenvalue problem to find the other. Therefore, our algorithm inherits some of its robustness from the QZ algorithm [23], 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 also easily constructed by using the MATLAB code in [39]. We choose Bézout over Sylvester because its conditioning can be analyzed (see Sect. 5) so its numerical behavior is better understood, and its symmetry can be exploited.

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

Figure 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 polynomial 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 [52].

Subdivision is also a key part of our algorithm and is crucial for the method to be practical for polynomial systems of high degree. There are two alternative subdivision strategies. The first consists of resampling the original functions f and g to construct new interpolants on each subdomain and the second involves resampling

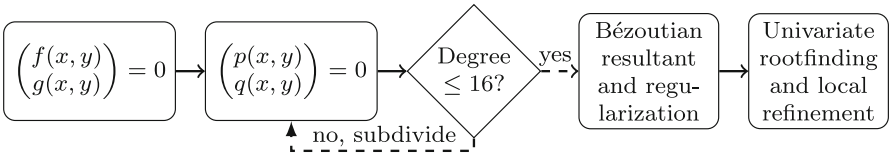

Fig. 1 Flowchart of our bivariate rootfinder based on the hidden variable resultant method and Bézout matrices. First, we approximate f and g by polynomials, using domain subdivision until the polynomial degree on each subdomain is ≤ 16 (Sect. 4). Then, we construct the resultant (Sect. 3), regularize (Sect. 6), and solve for one of the components before using a univariate rootfinder for the remaining component of the common zeros. Finally, we improve the accuracy of the solutions by local refinement (Sect. 7)

the global interpolants. The first implementation is available on MATLAB Central File Exchange [38], and it is particularly advantageous for functions that, for example, have highly varying magnitudes or cannot be evaluated with constant relative accuracy throughout the domain. The second approach is compatible with the Chebfun environment [51], where the interpolants may have been constructed from sampling data. This version of the code is available in the `roots` command of Chebfun2 [48], a software package written in MATLAB for computing with bivariate functions.

Both implementations are 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 s for degree 16 examples (with dubious accuracy) and cannot handle degree 30 or higher; however, it should be noted that these algorithms also compute complex solutions, while we focus on real solutions. Our MATLAB codes solve degree 16 in a fraction of a second, need about 5 s for degree 40, and approximately 100 s for degree 100. Often, the runtime scales *quartically*, and sometimes much less, with the degree, and our codes can solve problems with degree 1,000 (see Sect. 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.

In the next section we review previous literature related to solving $p(x, y) = q(x, y) = 0$. The core of our codes is a hidden variable resultant algorithm, exploiting Bézout matrices, that allows us to efficiently and reliably solve bivariate polynomial systems of low degree ($\leq$16). Section 3 contains the details of this method based on Bézout technology. In Sect. 4 we explain the process of domain subdivision in our algorithm. Section 5 analyzes the conditioning of the Bézout resultant method. In Sect. 6 we describe a regularization step to improve the stability of the algorithm. In Sect. 7 we discuss further details of the algorithm, and in Sect. 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 [10,20,28,34], and we mainly concentrate on approaches that require numeric, as opposed to symbolic, computations. These include variations on the resultant method, contouring algo-

rithms, 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 [17]. Generally, Gröbner bases are feasible only for small degree polynomial systems because of their complexity and numerical instability.

2.1 Resultant methods

The hidden variable resultant method is based on selecting one variable, say y, and writing p and q as polynomials in x with coefficients in $\mathbb{R}[y]$, that is,

$$p(x, y) = p_y(x) = \sum_{j=0}^{n_p} \alpha_j(y) x^j, \qquad q(x, y) = q_y(x) = \sum_{j=0}^{n_q} \beta_j(y) x^j. \tag{4}$$

Note that, although we have expressed (4) in the monomial basis, any polynomial basis can be used, see, for example, [9]. Monomials are the standard basis in the literature [18,21], 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 $[-1, 1]$ [50]. For this reason, we always express polynomials in the Chebyshev basis.

The two polynomials $p_y(x)$ and $q_y(x)$ in (4), 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) can be computed by finding the y-values such that a resultant matrix is singular.

There are many different resultant matrices such as Sylvester [18], Bézout [8], and other matrices [5,21,29], and this choice can affect subsequent efficiency (see Table 1) and conditioning (see Sect. 5). Usually, resultant matrices are constructed from polynomials expressed in the monomial basis [10,34], but they can be derived when using any other bases [9].

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 [2,7], Newton-type methods, inverse iteration methods (see the review [36]), the Ehrlich–Aberth method [11], and the standard approach of solving via linearization [22]. In our implementation we use linearization, which replaces a polynomial eigenvalue problem with a generalized eigenvalue problem with the same eigenvalues and Jordan structure [22]. Then we apply the QZ algorithm to the linearization; for a discussion of possible alternatives, see [36]. We leave for future research the study of the possible incorporation in our algorithm of alternative matrix polynomial eigensolvers.

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 a coordinate rotation, 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 $y = (z-\omega)/2$ [45], then 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 [26] are employed in computer graphics to generate zero level curves of bivariate functions. These contouring algorithms can be very efficient at solving (1), and until this paper the `roots(f,g)` command in Chebfun2 exclusively employed such a contouring approach [48]. 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.

Contouring algorithms suffer 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 [24].
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) via a contouring algorithm may be an adequate approach, but not always.

Remark 1 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 algorithm in this paper as a robust alternative.

2.3 Other numerical methods

Homotopy continuation methods [44] have a simple underlying approach based on solving an initial easy polynomial system that can be continuously deformed into (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 rootfinding problems [6,44].

The two-parameter eigenvalue approach constructs a determinantal expression for the polynomial interpolants to f and g and then rewrites $p(x, y) = q(x, y) = 0$ as a two-parameter eigenvalue problem [3],

$$A_1 v = x B_1 v + y C_1 v, \qquad A_2 w = x B_2 w + y C_2 w.$$

This approach has advantages because the two-parameter eigenvalue problem can be solved with the QZ algorithm [27], or other techniques [27]. 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 [41]. Alternatively, matrices of much larger size can be constructed using a generalized companion form, but these are too large to be efficient [37].

3 The resultant method with Bézout matrices

We now describe 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 $\mathbb{R}[y]$, using the Chebyshev basis:

$$p_y(x) = \sum_{j=0}^{n_p} \alpha_j(y) T_j(x), \qquad q_y(x) = \sum_{j=0}^{n_q} \beta_j(y) T_j(x), \qquad x \in [-1, 1], \tag{5}$$

where $\alpha_j, \beta_j \in \mathbb{R}[y]$, i.e., polynomials in y with real 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 (2).

The (Chebyshev) Bézout matrix of p_y and q_y in (5), denoted by $B(p_y, q_y)$, is defined[1] by $B(p_y, q_y) = (b_{ij})_{0 \le i,j \le N-1}$ [39], where $N = \max(n_p, n_q)$ and the entries satisfy

$$\frac{p_y(s)q_y(t) - p_y(t)q_y(s)}{s - t} = \sum_{i,j=0}^{N-1} b_{ij} T_i(s) T_j(t). \tag{6}$$

[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 [43, Lesson IX].

We observe that since $b_{ij} \in \mathbb{R}[y]$, $B(p_y, q_y)$ can be expressed as a *matrix polynomial* [22], i.e., a polynomial with matrix coefficients, that is

$$B(y) = B(p_y, q_y)(y) = \sum_{i=0}^{M} A_i T_i(y), \qquad A_i \in \mathbb{R}^{N\times N}, \tag{7}$$

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 subtly different (see Sect. 3.1).

It is well-known [18, Proposition 3.5.8 and Corollary 3.5.4] that two bivariate polynomials share a common factor in $\mathbb{R}[x, y]$ only if their resultant is identically zero. Throughout this paper we are assuming the solution set to (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 $p(x, y) = q(x, y) = 0$ is finite [30, Ch. 3];
- (Non-degeneracy) The resultant, $\det(B(y))$, is not identically zero and therefore, $B(y)$ is a *regular* matrix polynomial [22].

Since the resultant is not identically zero, the solutions to

$$\det\left(B(y)\right) = \det\left(B(p_y, q_y)\right) = 0 \tag{8}$$

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 they can also have a "common root at infinity" (see Sect. 3.1).

The eigenvalues of $B(y)$ can be found via linearization of matrix polynomials expressed in polynomial bases [33,39], which constructs a generalized eigenvalue problem $Cv = \lambda Ev$ with the same eigenvalues as $B(y)$. In our implementation we choose the colleague linearization, which is a companion-like matrix pencil for (matrix) polynomials expressed in the Chebyshev basis and 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 M and size N, 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 by a rootfinder based on the colleague matrix [13,51] (see Sect. 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

Table 1 Sizes and degrees of matrix polynomials constructed from the Bézout and Sylvester [5] resultant matrices

Resultant	Size of A_i in (7)	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)	$m_p + m_q$	$\max(n_p, n_q)$	$\max(n_p, n_q)(m_p + m_q)$

The product of the size of the A_i in (7) 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

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 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 [18, 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 $p(x, y) = q(x, y) = 0$ is zero-dimensional, and here further assume the common zeros 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., $\left[T_0(x_*), T_1(x_*), \ldots, T_{N-1}(x_*)\right]^T$.
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 $[0, 0, \ldots, 0, 1]^T$.

Note 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 from the determinant of the Sylvester matrix, as can be seen from a result in [31]. For instance, if $n_p > n_q$, then the two determinants differ by a factor $C(\alpha_{n_p}(y))^{n_p - n_q}$, when 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 algebraic 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 $\text{span}\{\big[T_0(x_i), \ldots, T_{N-1}(x_i)\big]^T\}$, 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 for both matrices [46] and matrix pencils [53], showing they are well-conditioned. This is often overlooked and multiple eigenvalues are sometimes unnecessarily assumed to be ill-conditioned. Therefore, if the original problem (1) only has simple common zeros, then the resultant method can compute accurate solutions and there is no need to perform a change of variables so that well-separated common zeros have different y-values.

In the presence of multiple or near-multiple common zeros the situation becomes more delicate, and in such cases the corresponding eigenvalues of $B(y)$ are ill-conditioned (see Theorem 1). Our focus is on the generic case where the solutions are simple, and 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 common zeros of multiplicity two with $\mathcal{O}(u^{1/2})$ accuracy, i.e., the error expected from a backward stable algorithm. However, since near-multiple solutions are also ill-conditioned any numerical scheme, including our own, suffer from inevitable numerical difficulties.

4 Subdivision of the domain

The Bézout resultant method of Sect. 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 for univariate rootfinding [14,15], as utilized for many years in Chebfun [13,51].

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

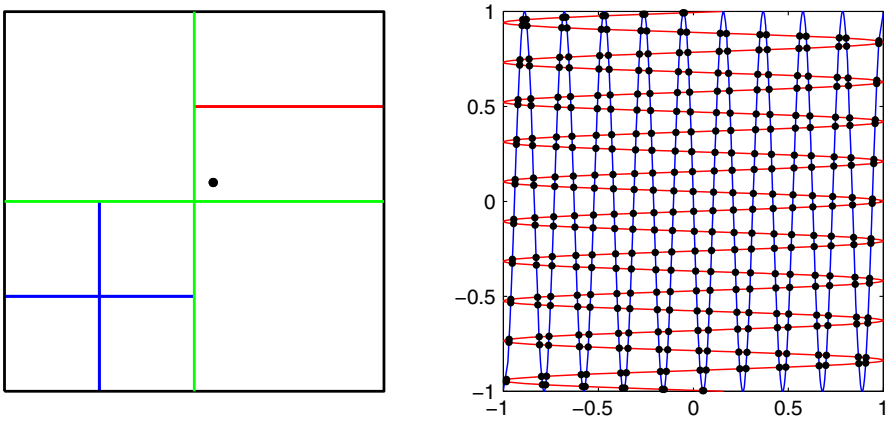

Fig. 2 *Left* Subdivision of $[-1, 1] \times [-1, 1]$ used for $f = \sin((x - 1/10)y)\cos(1/(x + (y - 9/10) + 5))$ and $g = (y - 1/10)\cos((x + (y + 9/10)^2/4))$. The *blue* and *red lines* represent subdivisions used to ensure the piecewise interpolants to f and g have degree ≤ 16, and the *green lines* are for both. The *black dot* is the only common zero. *Right* Zero contours (*blue, red*) and solutions (*black*) for oscillatory functions f and g as in (9)

We recursively subdivide $[-1, 1] \times [-1, 1]$, in the x- and y-variable independently, until the polynomial interpolants of f and g on each subdomain are of degree ≤ 16, a parameter determined by experimentation. Specifically, we subdivide in x if $\max(n_p, n_q) > 16$, and subdivide in y if $\max(m_p, m_q) > 16$. Figure 2 (left) shows how $[-1, 1] \times [-1, 1]$ is subdivided for $\sin((x-1/10)y)\cos(1/(x+(y-9/10)+5)) = (y - 1/10)\cos((x + (y + 9/10)^2/4)) = 0$. A polynomial system of degree ≤ 16 is solved on each subdomain.

Occasionally, f and g require a different amount of subdivision in one or both of the coordinate directions. Regardless, we need to subdivide f and g on the same grids. For example, consider the following problem:

$$\begin{pmatrix} f(x, y) \\ g(x, y) \end{pmatrix} = \begin{pmatrix} \sin(30x - y/30) + y \\ \sin(x/30 - 30y) - x \end{pmatrix} = 0, \quad (x, y) \in [-1, 1] \times [-1, 1]. \tag{9}$$

Here, the global interpolants p and q have polynomial degrees $(m_p, n_p, m_q, n_q) = (7, 63, 62, 6)$, so f requires many subdivisions in x, and g requires many in y. However, since f and g must be subdivided on the same grids we actually subdivide f and g many times in both coordinate directions. Figure 2 shows the zero contours and common zeros for (9).

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 subdomains $[-1, r_x] \times [-1, 1]$ 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 subdivid-

[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.

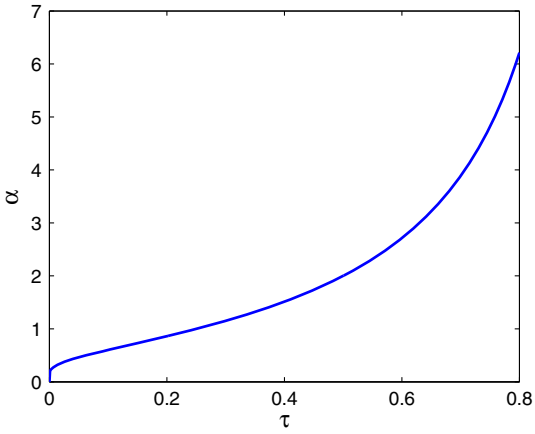

Fig. 3 Subdivision complexity is $\mathcal{O}\left(n^{\alpha}\right)$, where $\alpha = -\log 4/\log \tau$ and $0 < \tau \leq 1$ measures the reduction in polynomial degree after subdivision. When $\tau < 1/2$ polynomial evaluation is the dominating computational cost

ing at a solution since problems arising in practice often have zeros at special points like $(0, 0)$. 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 $2(.79)^3 \approx 1$, and each subdivision in the x-direction splits a problem into two, and the complexity of the algorithm depends 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 to be the smallest integer such that $n(.79)^K \leq d$, and we stop subdividing when either the degree is $< d$, or subdivision has been performed K times. After each subdivision we eliminate subdomains with $2|P_{00}| > \sum_{i,j} |P_{ij}|$ or $2|Q_{00}| > \sum_{i,j} |Q_{ij}|$, which certainly do not contain a solution.

Suppose for the moment that 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 $0 < \tau \leq 1$. Subdividing is done k times in both x and y until $n\tau^k \leq d = 16$, so $k \approx (\log d - \log n)/\log \tau$ and we have at most 4^k subdomains, each requiring $\mathcal{O}(d^6)$ operations to solve the local polynomial system. Therefore, the overall complexity is

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

Figure 3 shows the exponent $-\log 4/\log \tau$ as a function of τ. When $\tau \lesssim 0.5$, the overall complexity is as low as $\mathcal{O}(n^2)$ and dominated by the cost of subdivision. When $\tau \approx 0.79$ the complexity is as high as $\mathcal{O}(n^6)$, the same as without subdivision, but as long as $\tau < 1$ subdivision is useful 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 matrix of function samples.

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

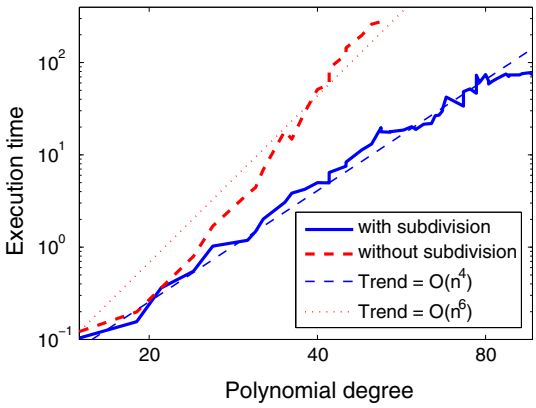

Fig. 4 A log–log plot of polynomial degree vs. execution time for computing the solutions to $\sin(\omega(x+y)) = \cos(\omega(x-y)) = 0$, where $1 \le \omega \le 50$. Here, subdivision has reduced the complexity from $\mathcal{O}(n^6)$ to $\mathcal{O}(n^4)$

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 [35] the functions f_1 and f_2 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., $y = x$. 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) because it reduces the norms of the coefficient matrices A_i, which has desirable consequences (see Sect. 5). In addition, subdivision reduces the size of the Bézout matrix, and this increases the distance from singularity of the matrix polynomial $B(y)$ as defined in (7).

Typically, we observe that $\tau \approx \sqrt{2}/2 \approx 0.707$, and hence subdivision leads to a complexity of $\mathcal{O}(n^4)$. Figure 4 shows the computational time for solving $\sin(\omega(x + y)) = \cos(\omega(x - y)) = 0$, $1 \le \omega \le 50$, 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,

$$0 = \begin{bmatrix} \hat{p}(\hat{x}, \hat{y}) \\ \hat{q}(\hat{x}, \hat{y}) \end{bmatrix} = \begin{bmatrix} \partial_x p(x_*, y_*) & \partial_y p(x_*, y_*) \\ \partial_x q(x_*, y_*) & \partial_y q(x_*, y_*) \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} + \begin{bmatrix} \delta p(\hat{x}, \hat{y}) \\ \delta q(\hat{x}, \hat{y}) \end{bmatrix}. \tag{10}$$

Our implementation initially scales p and q so that $\|p\|_\infty = \|q\|_\infty = 1$, where $\|p\|_\infty = \max_{x,y\in[-1,1]} |p(x, y)|$. Below we assume $\|p\|_\infty = \|q\|_\infty = 1$.

A stable numerical method computes a solution with an error of size $\mathcal{O}(\kappa_* u)$, where u is the unit roundoff and κ_* is the absolute condition number [25, Ch. 1] of (x_*, y_*) defined as

$$\kappa_* = \lim_{\epsilon\to 0^+} \sup \left\{ \frac{1}{\epsilon} \min \left\| \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right\|_2 : \hat{p}(\hat{x}, \hat{y}) = \hat{q}(\hat{x}, \hat{y}) = 0 \right\}, \tag{11}$$

where the supremum is taken over the set $\{(\hat{p}, \hat{q}) : \left\| \left[\begin{smallmatrix} \|\delta p\|_\infty \\ \|\delta q\|_\infty \end{smallmatrix} \right] \right\|_2 \le \epsilon\}$. Here and below $\|\cdot\|_2$ denotes either the Euclidean 2-norm of a column vector or the operator 2-norm of a square matrix, i.e., the largest singular value. Thus, by (10) the condition number (11) 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 computed solution, we examine the conditioning of the polynomial eigenvalue problem $B(y)$, which we use to find the y-values of the solution. To first order, the error in the computed eigenvalues of $B(y)$ is bounded by (conditioning)·(backward error), where $\kappa(y_*, B)$ denotes the conditioning of y_* as an eigenvalue of $B(y)$. It is defined by

$$\kappa(y_*, B) = \lim_{\epsilon\to 0^+} \sup \left\{ \frac{1}{\epsilon} \min |\hat{y} - y_*| : \det \widehat{B}(\hat{y}) = 0 \right\}, \tag{12}$$

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

The special structure of Bézout matrix polynomials allows us analyze $\kappa(y_*, B)$.

Theorem 1 *Let $B(y)$ be the Bézout matrix polynomial in (7) with $\|p\|_\infty = \|q\|_\infty = 1$, and suppose $(x_*, y_*) \in [-1, 1]\times[-1, 1]$ is a simple common zero of p and q yielding a simple eigenvalue y_* of $B(y)$. Then the absolute condition number of the eigenvalues of $B(y)$, as defined in (12), is*

$$\kappa(y_*, B) = \frac{\|v\|_2^2}{|\det J|}, \tag{13}$$

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

$$\frac{1}{2} \frac{\kappa_*^2}{\kappa_2(J)} \le \kappa(y_*, B) \le 2N \frac{\kappa_*^2}{\kappa_2(J)}, \tag{14}$$

$$\frac{1}{2} \frac{\kappa_*}{\|J\|_2} \le \kappa(y_*, B) \le 2N \frac{\kappa_*}{\|J\|_2}, \tag{15}$$

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

Proof Since $B(y)$ is a Bézout matrix polynomial, the eigenvector corresponding to y_* is of the form $v = \left[T_0(x_*), \ldots, T_{N-1}(x_*)\right]^T$ (see Sect. 3.1). The first-order perturbation expansion of y_* when $B(y)$ is perturbed to $B(y) + \Delta B(y)$ is [47, Thm. 5]

$$\hat{y} = y_* - \frac{v^T \Delta B(y_*) v}{v^T B'(y_*) v}, \tag{16}$$

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

From (12) and (16) we have $\kappa(y_*, B) \leq \|v\|_2^2/|v^T B'(y_*)v|$, and to show that this upper bound is always attained we take $\Delta B(y) = \epsilon v v^T t(y)$, where $t(y)$ is a scalar polynomial such that $\max_{y\in[-1,1]} |t(y)| = |t(y_*)| = \frac{1}{\|v\|_2}$.

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 $|T_k(y_*)| \leq 1$. To bound the denominator, we note that from (6) and (7) the term $v^T B(y)v$ can be interpreted as evaluation at $s = t = x_*$ of the Bézout function

$$\mathcal{B}(p, q) = \frac{p(s, y)q(t, y) - q(s, y)p(t, y)}{s - t}. \tag{17}$$

A convenient way to work out $v^T B'(y)v$ is by differentiating the Bézoutian (17) with respect to y and 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 evaluated by L'Hôspital's rule. Finally, since $p(x_*, y_*) = q(x_*, y_*) = 0$ we conclude that

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

hence, $\kappa(y_*, B) = \frac{\|v\|_2^2}{|\det J|}$, yielding (13).

The Jacobian is a 2×2 matrix so $\frac{1}{|\det J|} = \frac{\|J^{-T}\|_1}{\|J\|_1}$, in which $\|\cdot\|_1$ denotes the operator 1-norm of a matrix. Moreover, using $\frac{1}{2}\frac{\|J^{-1}\|_2}{\|J\|_2} \leq \frac{\|J^{-T}\|_1}{\|J\|_1} \leq 2\frac{\|J^{-1}\|_2}{\|J\|_2}$ and $\frac{\|J^{-1}\|_2}{\|J\|_2} = \frac{\|J^{-1}\|_2^2}{\|J^{-1}\|_2\|J\|_2} = \frac{\kappa_*^2}{\kappa_2(J)}$ we obtain (14), and (15) follows from $\kappa_2(J) = \kappa_*\|J\|_2$. □

The condition analysis and estimates in Theorem 1 appear to be the first in the literature for the Bézout polynomial eigenproblem $B(y)$. Importantly, it reveals situations when the Bézout resultant method worsens the conditioning. If $\|J\|_2 \geq 1$, then the eigenvalues of $B(y)$ can be computed accurately; Theorem 1 warns that this may not be the case when $\|J\|_2 \ll 1$, i.e., if the derivatives of p and q are small at (x_*, y_*). Specifically, the inequalities (15) show that in this case $\kappa(y_*, B) \gg \kappa_*$ if $\|J\|_2 \ll 1$. In particular, inequalities (14) show that $\kappa(y_*, B)$ 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 $\kappa_* > \mathcal{O}(u^{-1/2})$.

It is worth noting that the quantity $|\det J| = |\partial_x p \partial_y q - \partial_y p \partial_x q|$ in (13) 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 Bézout 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 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 $x_{\max} - x_{\min} \approx y_{\max} - y_{\min}$ with $|\Omega| = (x_{\max} - x_{\min})(y_{\max} - y_{\min}) \ll 1$. Since p and q are polynomials, Ω can be taken to be sufficiently small so that $\|p\|_\Omega \|q\|_\Omega = \mathcal{O}(|\Omega| \|\nabla p\|_2 \|\nabla q\|_2)$, where $\|p\|_\Omega = \max_{(x,y)\in\Omega} |p(x, y)|$ and $\|\nabla p\|_2 = \|\left[\begin{smallmatrix} \partial_x p(x_*, y_*) \\ \partial_y p(x_*, y_*) \end{smallmatrix}\right]\|_2$. In our codes, and for simplicity of analysis, we map Ω to $[-1, 1] \times [-1, 1]$ via the linear transformations

$$x \leftarrow \frac{x - \frac{1}{2}(x_{\max} + x_{\min})}{\frac{1}{2}(x_{\max} - x_{\min})}, \quad y \leftarrow \frac{y - \frac{1}{2}(y_{\max} + y_{\min})}{\frac{1}{2}(y_{\max} - y_{\min})}. \tag{18}$$

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 (7). Then $\|A_i\|_2$ are of size $\mathcal{O}(\|p\|_\Omega \|q\|_\Omega)$, and hence $\mathcal{O}(|\Omega| \|\nabla p\|_2 \|\nabla q\|_2)$. Therefore, the backward error resulting from solving $B(y)$ is of size $\mathcal{O}(u|\Omega| \|\nabla p\|_2 \|\nabla q\|_2)$.

The mapping (18) results in modified gradients such that $\|\nabla p\|_\Omega = \mathcal{O}(|\Omega|^{\frac{1}{2}} \|\nabla p\|_2)$ and $\|\nabla q\|_\Omega = \mathcal{O}(|\Omega|^{\frac{1}{2}} \|\nabla q\|_2)$. Thus, by (13), assuming that J is not too ill-conditioned, the conditioning is $\kappa_\Omega(y_*, B) = \mathcal{O}((|\Omega| \|\nabla p\|_2 \|\nabla q\|_2)^{-1})$. The crux of the discussion is that globally the conditioning (13) of the eigenvalues of the Bézout matrix polynomial can be worse than the original conditioning, but after local refinement the condition number significantly improves.

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 an error of size $\mathcal{O}(u|\Omega|)$ when we map x and y back to $[-1, 1] \times [-1, 1]$ by (18).

In addition, approximating p and q in Ω results in forward errors $\mathcal{O}(u\|p\|_\infty)$ and $\mathcal{O}(u\|q\|_\infty)$, respectively. So by (10) the overall error is $\mathcal{O}(u\|J^{-1}\|_2)$, reflecting the conditioning of the original problem, and the solutions are computed stably. The condition on Ω can be 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.

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 $\kappa_* \ll \mathcal{O}(u^{-1})$. The estimate (15) shows that $\kappa(y_*, B) \gg \kappa_*$ if and only if $\|J\|_2$ 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 in $[-1, 1] \times [-1, 1]$ of size $(2\max(n_p, n_q))^2$ 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 these 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 $\mathcal{O}(u\|J^{-1}\|_2)$.

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.

If f and g vary widely in magnitude in the domain of interest, then their global interpolants have poor relative accuracy in regions where $|f| \ll \|f\|_\infty$ and $|g| \ll \|g\|_\infty$, affecting the accuracy of the computed solutions. This issue is related to the *dynamic range* as discussed in [14,16] for univariate rootfinding, and also arises in bivariate rootfinding. A common solution is to resample the original functions f, g instead of the polynomial interpolants p, q when working in a subdomain. Provided that f, g can always be computed with relative accuracy, the accuracy in a subdomain can significantly improve, leading to accurate solutions. The Chebfun2 `roots` command does not implement this, because its primary goal is to achieve performance by working exclusively with polynomials, and its functionality is much broader than bivariate rootfinding. However, the implementation on the MATLAB Central File Exchange [38] does resample f and g after subdivision working with functions independently of Chebfun2.

A typical case where the dynamic range becomes an issue is when f, g are moderate degree polynomials represented in the monomial basis. In such cases our scheme can give inaccurate results without resampling the original functions, which is the same as any other scheme based on Chebyshev polynomials. This is reflected in the comparisons in [45] and resampling the original functions f and g is required, as it happens in our code [38].

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, through experiments we observe that 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 polynomial is not symmetric the form of its left eigenvectors are nontrivial and so its conditioning is more difficult to analyze.

We also note that once subdivision is employed to yield systems of low-degree polynomials, we can employ other methods to compute their common zeros, including those discussed in Sect. 2. We have chosen the Bézout resultant because it is the only method whose conditioning is fully understood. The conditioning of other resultants have been analyzed, but less deterministically: for example, Jónsson and Vavasis analyze the conditioning of the Macaulay resultant using probabilistic arguments [28]. A detailed comparison with other approaches is beyond the scope of this paper.

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.

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. 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 $\mathbb{C}$ [4, Sec 8.7.4], and they can cause catastrophic ill-conditioning of the other eigenvalues [40, Ch. 13].

Consequently 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 and g are assumed to be smooth, and therefore, their polynomial interpolants p and q, typically, have tensor Chebyshev expansions as in (2) with coefficient matrices P and Q with 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 (6). In this Sect. $B(y)$ 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. Hence $e_N = [0, \ldots, 0, 1]^T$ 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 $S = \left[e_{N-k+1}|e_{N-k+2}|\ldots|e_N\right]$, $1 \leq k < N$. 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 approximate eigenpairs of $B(y_0)$ are an artifact of finite precision arithmetic, and have nothing to do with the true 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

$\left[T_0(x_*), \ldots, T_{N-1}(x_*)\right]^T$, which coincides with the eigenvector of the matrix polynomial $B(y)$ at the eigenvalue y_*. Consequently, $B(y_*)$ has a null vector 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,

$$B(y) = \begin{bmatrix} B_1(y) & E(y)^T \\ E(y) & B_0(y) \end{bmatrix}, \tag{19}$$

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

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

Note that although the bottom-right part of $B(y)$ is of norm $\mathcal{O}(u)$, $E(y)$ 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 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 $B(y_*)$ is singular. 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 of size $\mathcal{O}(u)$. Numerically, this means that y_* can be computed stably via the regularized polynomial eigenvalue problem $B_1(y)$.

First, we 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

$$\widetilde{B}(y) = \begin{bmatrix} B_1(y) & E^T(y) \\ E(y) & 0 \end{bmatrix}, \tag{20}$$

and note that by Weyl's theorem, the matrix $\widetilde{B}(y_0)$ has eigenvalues within $\|B_0(y_0)\|_2 = \mathcal{O}(u)$ of those of $B(y_0)$. Moreover, by [32, 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 $\|E(y_0)^T B_1(y_0)^{-1} E(y_0)\|_2 = \mathcal{O}(u)$.

To see why we might expect $\|E(y_0)^T B_1(y_0)^{-1} E(y_0)\|_2 = \mathcal{O}(u)$, consider the LDL^T factorization $B_1(y_0) = LDL^T$, where L is unit lower triangular. For notational simplicity, we write L and D instead of $L(y_0)$ and $D(y_0)$. We expect that the L factor has decaying off-diagonal elements, and D has decaying diagonals.[4] To see why,

[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,

consider the first step in 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 $L^{-1}E$. Meanwhile, 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 $k+1$.

To show that $B_1(y_*)$ has an eigenvalue of size $\mathcal{O}(u)$, we invoke the Cauchy interlacing theorem [40, Ch. 10]: arranging the eigenvalues of $B_1(y_0)$ and $B(y_0)$ in nondecreasing order,

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

Hence $\lambda_i(B(y_0))$ is at least as small as $\lambda_i(B_1(y_0))$. An analogous identity also holds for the largest eigenvalues: 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 since E can be seen as a perturbation in $\widetilde{B}(y_0)$ in (20). 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 of size $\mathcal{O}(u)$. This shows that the y-values for which $B(y)$ has an extra null space can be reliably detected via solving the smaller problem $B_1(y)$.

The argument above holds when the coefficients of $B(y)$ decay, a property inherited from p and q. Otherwise, $\|B_0(y)\|_2$ and $\|E(y)\|_2$ are not negilible and we solve the polynomial eigenvalue problem for $B(y)$ without regularization. Typically, in this scenario $B(y)$ 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 cuts off high-degree terms in the Bézoutian (6). We also note that although the initial motivation for regularizing $B(y)$ was for stability, it also results in improved efficiency because the polynomial eigenvalue problem becomes smaller in size and lower in degree.

Footnote 4 continued
but most symmetric matrices do permit D to be diagonal, and our purpose is to explain the behavior observed in practice.

7 Further implementation details

Some further details remain and we discuss them in this section.

7.1 Construction of Bézout matrices

Our construction of (Chebyshev) Bézout matrices is based on the MATLAB code given in [39], which exploits a connection between Bézout matrices and the block symmetric linearization of scalar and matrix polynomials. We have improved the efficiency of that code by restricting it to only construct Chebyshev Bézout matrices. More specifically, in [39] it was shown that if $u(x)$ is of degree d and $v(x)$ is of degree at most $d-1$, then in the family of linearizations introduced in [33] there is a $d \times d$ symmetric pencil of the form $\lambda X + Y$ 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 (5), providing the ability to compute $B(y_0)$ for any $y_0 \in [-1, 1]$.

Let m_{p_s} and m_{q_s} be the degrees of p and q in y in the subdivided domain. Then, the coefficient matrices A_i in $B(y) = \sum_{i=0}^{m_{p_s}+m_{q_s}} A_i T_i(y)$ can be obtained by sampling $B(y)$ at $m_{p_s} + m_{q_s} + 1$ Chebyshev points in the subdivided y-interval and converting them to Chebyshev coefficients using the fast Fourier transform.

7.2 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 [50, Ch. 18] for matrix polynomials, the standard companion-like linearization for the Chebyshev basis.

After forming the coefficient matrices of $B(y)$, we remove the leading coefficients with Frobenius norms smaller than $u \max_i \|A_i\|_F$, where A_i are as in (7). An analogous technique is used in the 1D Chebfun, and since this can be regarded as a small normwise perturbation, it can be done without causing instability issues. The size of the pencil is (dimension of $B_1(y)$)·(degree of $B_1(y)$) as summarized in Table 1, but regularization and truncation can reduce the size.

The colleague matrix pencil for a matrix polynomial $\sum_{i=0}^{M} A_i T_i(\lambda)$, $A_i \in \mathbb{R}^{N\times N}$ is [50]

$$\lambda X + Y = \lambda \begin{bmatrix} A_M & & & \\ & I_N & & \\ & & \ddots & \\ & & & I_N \end{bmatrix} - \frac{1}{2} \begin{bmatrix} -A_{M-1} & I_N - A_{M-2} & -A_{M-3} & \cdots & -A_0 \\ I_N & 0 & I_N & & \\ & \ddots & \ddots & \ddots & \\ & & I_N & 0 & I_N \\ & & & 2I_N & 0 \end{bmatrix}.$$

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 [23, Ch. 7].

7.3 Univariate rootfinding

Once we have found the y-values of the solutions, then we find the x-values by a univariate rootfinding algorithm based on computing the eigenvalues of the colleague matrix [50], and using the `eig` command in MATLAB. Due to subdivision, the polynomials are of degree ≤ 16 on each subregion, and computing their roots is negligible in cost.

We compute the roots of both $p(x, y_*)$ and $q(x, y_*)$ on the subdivided x-interval separately and numerically verify that the absolute values of p and q are less than a certain tolerance at these solutions, discarding those that are not. The exact tolerance we take depends on whether we are seeking an initial estimate or performing a local refinement (see Subsect. 7.4). Afterwards, we merge any x-values that are closer than a distance of $\mathcal{O}(u)$ apart by averaging to prevent double-counting a solution due to rounding errors.

7.4 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 do the following:

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 the conditioning worsening (see Sect. 5), spurious, multiply-counted, and inaccurate solutions. 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 Sect. 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. For example, 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 subdomain that contains very few common zeros.

7.5 Solutions off the domain, but numerically close

Some care is required to avoid missing solutions that lie on or near the boundary of the domain $[-1, 1] \times [-1, 1]$, as a small backward perturbation can move them outside. This is the same reason we do not exactly bisect in the subdivision (see Sect. 4). We must also be careful not to miss real solutions that are numerically perturbed to complex ones with negligible imaginary parts.

To ensure that we do not miss the solutions near the boundary, we initially look for solutions in the slightly enlarged domain $[-1 - \delta, 1 + \delta] \times [-1 - \delta, 1 + \delta]$, where $\delta = 10^{-10}$. Then we disregard solutions outside $[-1, 1] \times [-1, 1]$. Solutions off the domain within a distance of 10^{-15} are regarded as solutions on the boundary by perturbing them.

To capture solutions that have been numerically perturbed to have negligible imaginary parts we check for the eigenvalues of $B(y)$ with real parts in $[-1, 1] \times [-1, 1]$ and the imaginary parts of size $\mathcal{O}(u^{1/2})$, or smaller. The tolerance of size $\mathcal{O}(u^{1/2})$ is reasonable since two numerically close common zeros can be perturbed into the complex plane by this amount.

7.6 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 in the domain of interest.

There are several ways spurious zeros can arise: (i) A shared zero at infinity, for example, 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, a computed spurious zero arising before local refinement will reveal itself when we refine. There are two reasons why refinement removes spurious zeros. First, refinement reduces the degree of the polynomial approximants, meaning it is less likely to encounter shared zeros at infinity. Second, the matrix polynomial $B(y)$ is smaller in both size and degree and typically has a greatly improved condition number. In the unlikely event that a spurious zero is present in the final stage, univariate

rootfinding performs an additional control to check that we have the common zeros of the original bivariate functions.

The above qualitative arguments suggest, and experiments corroborate, that local refinement is an effective approach for the removal of spurious zeros, a well-known challenge for resultant-based methods. Although in theory spurious zeros can still arise, we are not aware of an example where local refinement fails to remove them.

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) are drawn as blue and red curves, respectively, computed by the command `roots(f)` in Chebfun2 [48]. 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 $(m_p, n_p, m_q, n_q) = (20, 20, 24, 30)$:

$$\begin{pmatrix} T_7(x)T_7(y)\cos(xy) \\ T_{10}(x)T_{10}(y)\cos(x^2y) \end{pmatrix} = 0, \qquad (x, y) \in [-1, 1] \times [-1, 1]. \tag{21}$$

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 Sect. 3). The solutions to (21) are known exactly and the maximum absolute error in the computed solutions is 8.88×10^{-16}. In Fig. 5 we show the zero contours and

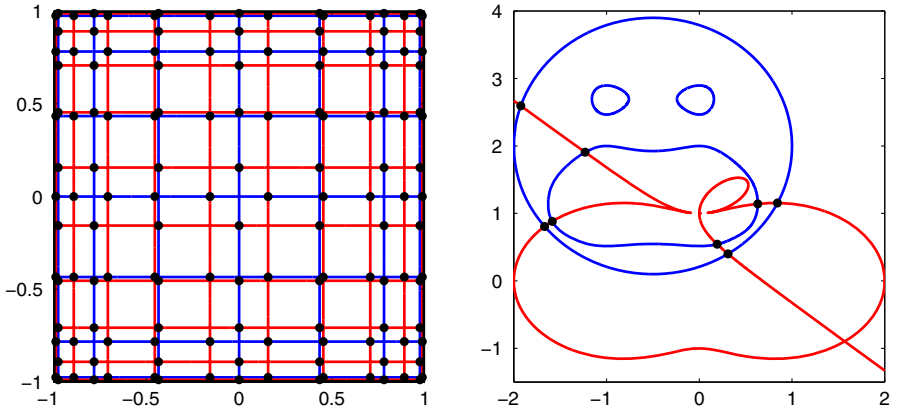

Fig. 5 The zero contours of f (*red*) and g (*blue*), and the common zeros (*black dots*) computed with our algorithm. *Left* Coordinate alignment (21). Simple common zeros are aligned with the coordinate directions, but this causes no numerical difficulties (see Sect. 3). *Right* Face and apple. Even though the polynomials are small near the origin we can detect this and use a local Bézout refinement to obtain accurate solutions (see Sect. 7)

common zeros for (21). 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 [42], 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 Sect. 5.1).

Example 3 (Devil's example) A particularly ill-conditioned problem is the following:

$$\begin{pmatrix} \prod_{i=0}^{10} \left(y^2(4y^2 - \frac{i}{10}) - x^2(4x^2 - 1)\right) \\ 256(x^2 + y^2)^2 + 288(x^2 + y^2) - 512(x^3 - 3xy^2) - 27 \end{pmatrix} = 0 \qquad (22)$$

on $(x, y) \in [-1, 1] \times [-1, 1]$. Again $f = p$ and $g = q$, with degrees $(m_p, n_p, m_q, n_q) = (44, 44, 4, 4)$. For this example it is extremely difficult without resampling the original functions to obtain accurate solutions because the functions vary widely in magnitude. However, this is easily overcome by resampling the original functions f and g during subdivision as in [38].

Example 4 (Hadamard) Chebfun2 allows us to construct polynomials from interpolation data at a tensor Chebyshev grid [48], even when a function handle is not specified. 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), where $p(x_i, x_j) = H_{32}(i, j)$, $q(x_i, x_j) = H_{64}(i, j)$, and x_i are Chebyshev points. The Hadamard matrices contain ± 1 entries and therefore, p and q (of degrees $(m_p, n_p, m_q, n_q) = (31, 31, 63, 63)$) have many zero contours and simple common zeros. Our algorithm requires 89 s and the maximum of $|p(x_*, y_*)|$ and $|q(x_*, y_*)|$ over all computed solutions (x_*, y_*) is 3.98×10^{-13} (Fig. 6).

In this example function handles f and g are not available so during subdivision the global polynomial must be resampled. Furthermore, subdivision does not lead to an efficient reduction in the degree, and τ, as defined in Sect. 4, is estimated to be 0.82 for p and 0.75 for q. We obtained the estimates for τ by subdividing the domain into $2^5 \times 2^5$ subdomains and taking the average degree reduction.

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

$$\begin{pmatrix} \mathrm{Ai}(-13(x^2 y + y^2))) \\ J_0(500x)y + x J_1(500y) \end{pmatrix} = 0, \qquad (23)$$

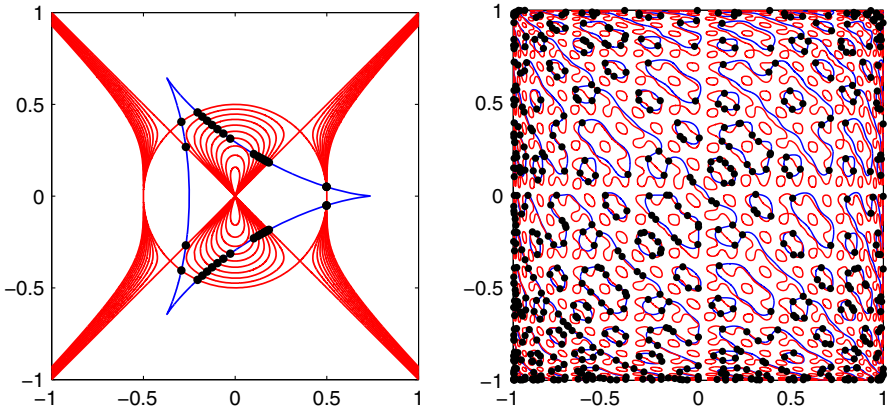

Fig. 6 *Left* Devil's example (22). By resampling f, g during subdivision accurate solutions are obtained despite the dynamical range issue. *Right* Hadamard example, polynomials obtained by data values. For this example significant subdivision is required as subdividing does not reduce the degree very effectively

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 5,932 common zeros in 501 s, and we independently verified that our algorithm found all the solutions to (23) 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 $-\log 4/\log \tau = 2.6$.

Example 6 (SIAM 100-Digit Challenge problem) An article in SIAM News in 2002 set a challenge to solve ten problems, each to ten digits (the solution to each problem was a single real number) [49]. One of these problems was to find the global minimum of the following complicated and highly oscillatory function:

$$f(x, y) = \left(\frac{x^2}{4} + e^{\sin(50x)} + \sin(70\sin(x))\right) + \left(\frac{y^2}{4} + \sin(60e^y) + \sin(\sin(80y))\right) - \cos(10x)\sin(10y) - \sin(10x)\cos(10y). \tag{24}$$

Since all the local extrema of (24) 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 $[-1, 1] \times [-1, 1]$ [12]. 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 2,720 local extrema of (24) in 257 s and obtains the global minimum to an accuracy of 1.12×10^{-15}. The function (24) does have some structure because it is a low rank function [48], but this structure is not directly exploited in the rootfinding algorithm. In Fig. 7 we show the location of the 2,720 local extrema

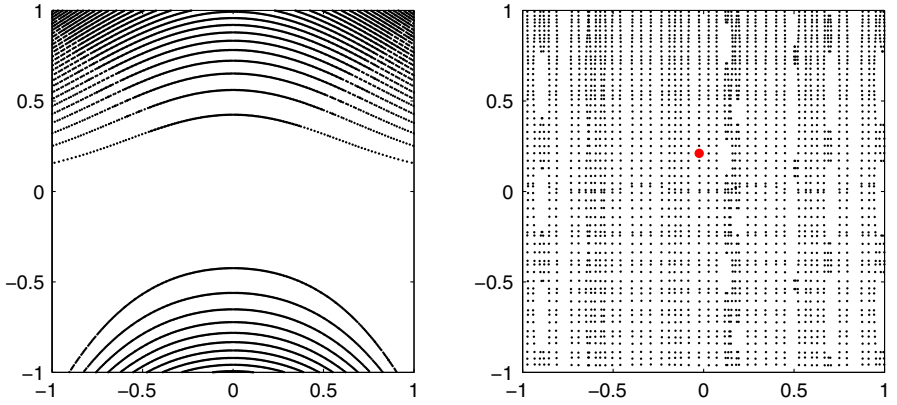

Fig. 7 *Left* Solutions to (23). The number of isolated solutions is 5,932; there are so many that they appear as lines rather than *dots*. *Right* SIAM digit problem (24). The zeros of f_x and f_y are shown as *black dots* and the location of the minimum is a *red dot*. We consider these examples as of very high degree, and present them as an indication of the robustness of our algorithm

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 $-\log 4/\log \tau = 2.2$.

Finally, to test the code with examples of even higher degree, we doubled all the coefficients inside the trigonometric functions in (24), e.g., $e^{\sin(50x)} \leftarrow e^{\sin(100x)}$, $\sin(70\sin(x)) \leftarrow \sin(140\sin(x))$, etc., and ran the same experiment. The partial derivatives of this f are now of numerical degrees $(m_p, n_p, m_q, n_q) = (1, 781, 1, 204, 1, 781, 1, 204)$. Our code computed 9,318 local extrema in 1,300 s, and the estimated complexity exponent is 2.1. For this problem, the dominant part of the runtime is taken by the subdivision step.

Acknowledgments We are grateful to John Boyd for making us aware of contouring algorithms and to Rob Corless for a fruitful discussion. We would like to thank Nick Higham, Françoise Tisseur, and Nick Trefethen for their support throughout our collaboration. We thank the anonymous referees for their useful comments that have led to improvements in the paper.

References

1. Aruliah, D.A., Corless, R.M., Gonzalez-Vega, L., Shakoori, A.: Geometric applications of the Bezout matrix in the Lagrange basis. In: Proceedings of the 2007 International Workshop on Symbolic-Numeric Computation, pp. 55–64. ACM Press, New York (2007)
2. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for nonlinear eigenvalue problems using contour integrals. JSIAM Lett. **1**, 52–55 (2009)
3. Atkinson, F.V.: Multiparameter Eigenvalue Problems. Academic Press, New York (1972)
4. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia (2000)
5. Barnett, S.: Greatest common divisors from generalized sylvester resultant matrices. Linear Multilinear Algebra **8**, 271–279 (1980)
6. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Numerically Solving Polynomial Systems with Bertini. SIAM, Philadelphia (2013)

7. Beyn, W.-J.: An integral method for solving nonlinear eigenvalue problems. Linear Algebra Appl. **436**(10), 3839–3863 (2012)
8. Bézout, É.: Théorie Générale des Équations Algébriques. PhD thesis, Pierres, Paris (1779)
9. Bini, D.A., Gemignani, L.: Bernstein-bezoutian matrices. Theor. Comput. Sci. **315**(2), 319–333 (2004)
10. Bini, D.A., Marco, A.: Computing curve intersection by means of simultaneous iterations. Numer. Algorithms **43**, 151–175 (2006)
11. Bini, D.A., Noferini, V.: Solving polynomial eigenvalue problems by means of the Ehrlich–Aberth method. Linear Algebra Appl. **439**, 1130–1149 (2013)
12. Bornemann, F., Laurie, D., Wagon, S., Waldvogel, H.: The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing. SIAM, Philadelphia (2004)
13. Boyd, J.P.: Computing zeros on a real interval through chebyshev expansion and polynomial rootfinding. SIAM J. Numer. Anal. **40**, 1666–1682 (2002)
14. Boyd, J.P.: Computing real roots of a polynomial in chebyshev series form through subdivision. Appl. Numer. Math. **56**, 1077–1091 (2006)
15. Boyd, J.P.: 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)
16. Boyd, J.P., Gally, D.H.: Numerical experiments on the accuracy of the chebyshev-frobenius companion matrix method for finding the zeros of a truncated series of chebyshev polynomials. J. Comput. Appl. Math. **205**(1), 281–295 (2007)
17. Buchberger, B.: Introduction to Gröbner bases. In: Gröbner Basis and Applications, vol. 251, pp. 3–31. Cambridge University Press, Cambridge (1998)
18. Cox, D.A., Little, J.B., O'Shea, D.: Ideals, Varieties, and Algorithms: Introduction to Computational Algebraic Geometry and Commutative Algebra, 3rd edn. Springer, Berlin (2007)
19. Diaz-Toca, G.M., Fioravanti, M., Gonzalez-Vega, L., Shakoori, A.: Using implicit equations of parametric curves and surfaces without computing them: polynomial algebra by values. Comput. Aided Geom. D. **30**, 116–139 (2013)
20. Dreesen, P., Batselier, K., De Moor, B.: Back to the roots: Polynomial system solving, linear algebra, systems theory. In: Proceedings of 16th IFAC Symposium on System Identification, pp. 1203–1208 (2012)
21. Emiris, I.Z., Mourrai, B.: Matrices in elimination theory. J. Symb. Comput. **28**, 3–44 (1999)
22. Gohberg, I., Lancaster, P., Rodman, L.: Matrix Polynomials. SIAM, Philadelphia (unabridged republication of book first published by academic press in 1982) edition (2009)
23. Golub, G.H., Van Loan, C.F.: Matrix Computations. The Johns Hopkins University Press, Baltimore (1996)
24. Harnack, C.G.A.: Über vieltheiligkeit der ebenen algebraischen curven. Math. Ann. **10**, 189–199 (1876)
25. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
26. Hilton, A., Stoddart, A.J., Illingwort, J., Windeatt, T.: Marching triangles: range image fusion for complex object modelling. In: International Conference on Image Processing, vol. 1 (1996)
27. Hochstenbach, M.E., Košir, T., Plestenjak, B.: A jacobi-davidson type method for the two-parameter eigenvalue problem. SIAM J. Matrix Anal. Appl. **26**(2), 477–497 (2004)
28. Jónsson, G., Vavasis, S.: Accurate solution of polynomial equations using macaulay resultant matrices. Math. Comp. **74**, 221–262 (2005)
29. Kapur, D., Saxena, T.: Comparison of various multivariate resultant formulations. In: Levelt, A. (ed) Proceedings of International Symposium on Symbolic and Algebraic Computation, pp. 187–194. Montreal (1995)
30. Kirwan, F.C.: Complex Algebraic Curves. Cambridge University Press, Cambridge (1992)
31. Kravitsky, N.: On the discriminant function of two commuting nonselfadjoint operators. Integr. Equ. Oper. Theory **3**, 97–125 (1980)
32. Li, R.-C., Nakatsukasa, Y., Truhar, N., Wang, W.: Perturbation of multiple eigenvalues of hermitian matrices. Linear Algebra Appl. **437**, 202–213 (2012)
33. Mackey, D.S., Mackey, N., Mehl, C., Mehrmann, V.: Vector spaces of linearizations for matrix polynomials. SIAM J. Matrix Anal. Appl. **28**, 971–1004 (2006)
34. Manocha, D., Demmel, J.: Algorithms for intersecting parametric and algebraic curves I: simple intersections. ACM Trans. Graph. **13**, 73–100 (1994)
35. Marks II, R.J.: Introduction to Shannon Sampling and Interpolation Theory. Springer, New York (1991)
36. Mehrmann, V., Voss, H.: Nonlinear eigenvalue problems: A challenge for modern eigenvalue methods. Mitt. der Ges. fr Angewandte Mathematik and Mechanik **27**, 121–151 (2005)

37. Muhič, A., Plestenjak, B.: On the quaratic two-parameter eigenvalue problem and its linearization. Linear Algebra Appl. **432**, 2529–2542 (2010)
38. Nakatsukasa, Y., Noferini, V., Townsend, A.: Computing common zeros of two bivariate functions. MATLAB Central File Exchange (2013). http://www.mathworks.com/matlabcentral/fileexchange/44084
39. Nakatsukasa, Y., Noferini, V., Townsend, A.: Vector spaces of linearizations for matrix polynomials: a bivariate polynomial approach. Preprint (2014)
40. Parlett, B.N.: The Symmetric Eigenvalue Problem. SIAM, Philadelphia (1998)
41. Plaumann, D., Sturmfels, B., Vinzant, C.: Computing linear matrix representations of Helton–Vinnikov curves. Math. Methods Syst. Optim. Control Oper. Theory **222**, 259–277 (2012)
42. Sagraloff, M. et al.: Gallery of algebraic curves and their arrangements. http://exacus.mpi-inf.mpg.de/gallery.html
43. Salmon, G.: Lessons Introductory to the Modern Higher Algebra. G. E. Stechert & Co., New York (1885)
44. Sommese, A.J., Wampler, C.W.: The Numerical Solution of Systems of Polynomials. World Scientific, Singapore (2005)
45. Sorber, L., Van Barel, M., De Lathauwer, L.: Numerical solution of bivariate and polyanalytic polynomial systems. Preprint (2013)
46. Sun, J.-G.: On condition numbers of a nondefective multiple eigenvalue. Numer. Math. **61**, 265–275 (1992)
47. Tisseur, F.: Backward error and condition of polynomial eigenvalue problems. Linear Algebra Appl. **309**, 339–361 (2000)
48. Townsend, A., Trefethen, L.N.: An extension of chebfun to two dimensions. SIAM J. Sci. Comput. **35**(6), C495–C518 (2013)
49. Trefethen, L.N.: A hundred-dollar, hundred-digit challenge. SIAM News, 35 (2002)
50. Trefethen, L.N.: Approximation Theory and Approximation Practice. SIAM, Philadelphia (2013)
51. Trefethen, L.N. et al.: Chebfun version 4.2.2949. Software. The Chebfun Development Team (2013)
52. Wilkinson, J.H.: The perfidious polynomial. In: Golub, G.H. (ed) Studies in Numerical Analysis. Mathematical Association of America (1984)
53. Xie, H., Dai, H.: On the sensitivity of multiple eigenvalues of nonsymmetric matrix pencils. Linear Algebra Appl. **374**, 143–158 (2003)