Numer. Math. (2018) 139:633–682
https://doi.org/10.1007/s00211-018-0948-4

Stable polefinding and rational least-squares fitting via eigenvalues

Shinji Ito[1,2] · **Yuji Nakatsukasa**[3]

Received: 6 September 2016 / Revised: 14 January 2018 / Published online: 21 February 2018

Abstract A common way of finding the poles of a meromorphic function f in a domain, where an explicit expression of f is unknown but f can be evaluated at any given z, is to interpolate f by a rational function $\frac{p}{q}$ such that $r(\gamma_i) = f(\gamma_i)$ at prescribed sample points $\{\gamma_i\}_{i=1}^{L}$, and then find the roots of q. This is a two-step process and the type of the rational interpolant needs to be specified by the user. Many other algorithms for polefinding and rational interpolation (or least-squares fitting) have been proposed, but their numerical stability has remained largely unexplored. In this work we describe an algorithm with the following three features: (1) it automatically finds an appropriate type for a rational approximant, thereby allowing the user to input just the function f, (2) it finds the poles via a generalized eigenvalue problem of matrices constructed directly from the sampled values $f(\gamma_i)$ in a one-step fashion, and (3) it computes rational approximants $\hat{p}, \hat{q}$ in a numerically stable manner, in that $(\hat{p}+\Delta p)/(\hat{q}+\Delta q) = f$ with small $\Delta p, \Delta q$ at the sample points, making it the first rational interpolation (or approximation) algorithm with guaranteed numerical stability. Our algorithm executes an implicit change of polynomial basis by the QR factorization, and allows for oversampling combined with least-squares fitting. Through experiments we illustrate the resulting accuracy and stability, which can significantly outperform existing algorithms.

Yuji Nakatsukasa is supported by JSPS as an Overseas Research Fellow.

✉ Yuji Nakatsukasa
nakatsukasa@maths.ox.ac.uk

Shinji Ito
s-ito@me.jp.nec.com

[1] Graduate School of Information Science and Technology, University of Tokyo, Tokyo 113-8656, Japan

[2] Present Address: NEC Corporation, Kanagawa 211-8666, Japan

[3] Mathematical Institute, University of Oxford, Oxford OX2 6GG, UK

Mathematics Subject Classification 65D05 Numerical analysis, Interpolation · 65D15 Numerical analysis, Algorithms for functional approximation

1 Introduction

Let f be a meromorphic function in a domain Ω, whose explicit expression is unknown but can be evaluated at any set of sample points $\{\gamma_i\}_{i=1}^{L}$. This paper investigates numerical algorithms for finding the poles of f, along with the associated problem of finding a rational approximant $r = p/q \approx f$ in Ω. Finding the poles of a meromorphic or rational function f is required in many situations, including resolvent-based eigensolvers [4,36,40] and the analysis of transfer functions [30,38].

One natural way of finding the poles is to first approximate f in Ω by a rational function $r = p/q$, then find the poles of r, i.e., the roots of q. A common approach to obtain $r \in \mathcal{R}_{m,n}$ (a rational function of type m, n, i.e., $r = p/q$ for polynomials p, q of degree at most m, n respectively) is to interpolate f at $m + n + 1$ points in Ω (such as the unit disk), a code for which is available in the Chebfun command `ratinterp` [21,35]. However, this is a two-step process; when the poles are of primary interest, explicitly forming r is unnecessary and can be a cause for numerical errors. Moreover, the type of the rational function is usually required as input.

In this paper we develop a polefinding algorithm `ratfun` that essentially involves just solving one generalized eigenvalue problem, thereby bypassing the need to form r. `ratfun` starts by finding an appropriate type for the rational approximant from the function values: roughly, it finds the pair (m, n) with a smallest possible n (without taking m excessively large) such that $r = p/q \approx f$ holds with $r \in \mathcal{R}_{m,n}$; in Sect. 3 we make this more precise. This allows the user to input just the function f to obtain the poles. The rational approximant can also be obtained if necessary.

Since polefinding for $r = p/q$ boils down to rootfinding for q, it is inevitable that the algorithm involves an iterative process (as opposed to processes requiring finitely many operations in exact arithmetic such as a linear system), and hence it is perhaps unsurprising that we arrive at an eigenvalue problem. Our algorithm has runtime that scales cubically with the type of the rational approximant, which is comparable to some of the state-of-the-art algorithms.

A key property of `ratfun` is its numerical stability. To our knowledge, no previous polefinding algorithm has been proven to be numerically stable. Numerical stability here means backward stability in the sense that $\frac{\hat{p}+\Delta p}{\hat{q}+\Delta q} = f$ holds exactly at the sample points, where $\frac{\hat{p}}{\hat{q}}$ is the computed rational approximant and $\|\Delta p\|_L/\|\hat{p}\|_L$, $\|\Delta q\|_L/\|\hat{q}\|_L$ are $O(u)$ where u is the unit roundoff (throughout we write $x = O(y)$ to mean $x \le My$ for a moderate $M > 0$), and $\|\cdot\|_L$ is the vector norm of function values at the sample points $\{\gamma_i\}_{i=1}^{L}$, see *Notation* below. Classical algorithms such as Cauchy's [13], Jacobi's [26], Thiele's continued fractions [44, Sect. 2.2.2] and Neville-type algorithms [44, Sect. 2.2.3] are known to be of little practical use due to instability [47, Ch. 26]. The more recent Chebfun's `ratinterp` is based on the SVD, and combined with a degree reduction technique, `ratinterp` is reliable in many situations. However, as we shall see, the algorithm is still unstable when a sample point lies near a pole. Once the numerical degree is determined, the way our algorithm `ratfun` finds the poles is mathematically equivalent to `ratinterp` (and some other algorithms), but overcomes this instability by avoiding the use of the FFT and employing a diagonal scaling to attenuate the effect of an excessively large sample value $|f(\gamma_i)|$.

Another practical method is a naive SVD-based interpolation algorithm (described in Sect. 2.1), and despite its simplicity and straightforward derivation, it works surprisingly well; indeed we prove stability in the above sense for obtaining r when an appropriate diagonal scaling is employed. Nonetheless, it is still based on a two-step approach, and the detour of forming the coefficients of p, q before computing the poles incurs unnecessary inaccuracy. As is well known, in rootfinding problems the choice of the polynomial basis is critical for accurate computation [47, App. 6], as Wilkinson famously illustrated in [52]. `ratfun`, by contrast, bypasses the coefficients and implicitly performs an appropriate change of polynomial basis.

Also worth mentioning are polefinding algorithms based on a Hankel generalized eigenvalue constructed via evaluating discretized contour integrals of the form $s_j := \frac{1}{2\pi \mathrm{i}} \oint_\gamma z^j \frac{1}{f(z)} dz$ [29,40]. This algorithm still has a two-step flavor (computing integrals and solving eigenproblem), and it was recently shown [3] to be mathematically equivalent to rational interpolation followed by polefinding, as in Chebfun's `ratinterp`. We shall see that this algorithm is also unstable.

We shall see that `ratfun` is also equivalent mathematically to these two algorithms, in that our eigenproblem can be reduced to the Hankel eigenproblem by a left equivalence transformation. However, numerically they are very different, and we explain why `ratfun` is stable while others are not.

The contributions of this paper can be summarized as follows.

1. Polefinding (and rootfinding if necessary) by a one-step eigenvalue problem.
2. Automatic determination of a type (m, n) for the rational approximant. This allows the user to obtain p, q from the input f alone. In previous algorithms the type (m, n) has been a required input.
3. Stability analysis. We introduce a natural measure of numerical stability for rational interpolation, and establish that our algorithm `ratfun` is numerically stable.

Table 1 compares algorithms for polefinding and indicates the stability and complexity of each method, along with the dominant computational operation. Here, RKFIT refers to the recent algorithm by Berljafa and Güttel [8], Hankel is the algorithm based on contour integration, resulting in a generalized eigenvalue problem involving Hankel matrices (summarized in Sect. 4.8), and naive is the naive method presented in Sect. 2.1. By "avoid `roots(q)`" we mean the algorithm can compute the poles without forming the polynomial q and then finding its roots.

This paper is organized as follows. In Sect. 2.1 we review some previous algorithms, which also leads naturally to our proposed algorithm. In Sect. 3 we discuss the process of finding an appropriate type of the rational approximation. Section 4 is the main part where our eigenvalue-based algorithm is derived, and we prove its numerical stability in Sect. 5. We present numerical experiments in Sect. 6.

Notation $\mathcal{P}_n$ is the set of polynomials of degree at most n, and $\mathcal{R}_{m,n}$ is the set of rational functions of type at most (m, n). Unless mentioned otherwise, f is assumed to

Table 1 Comparison between polefinding algorithms

	`ratinterp`	RKFIT	Hankel	Naive	`ratfun`
p, q stability	$\times$	$\times$	–	$\surd$	$\surd$
Avoid `roots(q)`	$\times$	$\times$	$\surd$	$\times$	$\surd$
Complexity	$O(Ln^2)$	$O(Ln^2)$	$O(n^3)$	$O(L(m+n)^2)$	$O(L(m+n)^2)$
Main computation	SVD etc	Krylov	GEP	SVD	Rectangular GEP

GEP generalized eigenvalue problem

be meromorphic in a region Ω in the complex plane, and (m, n) denotes the type of the rational approximant $r = p/q \approx f$ that our algorithm finds: $r \in \mathcal{R}_{m,n}$, that is, $p \in \mathcal{P}_m$ and $q \in \mathcal{P}_n$. When necessary, when f is rational, we denote by (M, N) its exact type, that is, $f = \frac{p}{q}$ where p, q are coprime polynomials of degree M, N, respectively. We define $c_p = [c_{p,0}, c_{p,1}, c_{p,2}, \dots, c_{p,m}]^\top$, $c_q = [c_{q,0}, c_{q,1}, c_{q,2}, \dots, c_{q,n}]^\top$ to be the vectors of their coefficients such that $p(z) = \sum_{i=0}^{m} c_{p,i}\phi_i(z)$ and $q(z) = \sum_{i=0}^{n} c_{q,i}\phi_i(z)$, in which $\{\phi_i(z)\}_{i=0}^{\max(m,n)}$ is a polynomial basis, which we take to be the monomials $\phi_i(z) = z^i$ unless otherwise mentioned. When other bases are taken we state the choice explicitly. L is the number of sample points, denoted by $\{\gamma_i\}_{i=1}^{L}$, which we assume to take distinct points in $\Omega \in \mathbb{C}$. $F = \mathrm{diag}(f(\gamma_1), \dots, f(\gamma_L))$ is the diagonal matrix of function values at the sample points. We also let $\Gamma = \mathrm{diag}(\gamma_1, \dots, \gamma_L)$. $\|\cdot\|_L$ denotes a norm of a function, defined via the function values at the sample points $\|g\|_L = \sqrt{\sum_{i=1}^{L} |g(\gamma_i)|^2}$. Computed approximants wear a hat, so for example $\hat{\xi}_i$ is a computed pole. V is the Vandermonde matrix generated from the L sample points $(\gamma_i)_{i=1}^{L}$ with (i, j)-element $(V)_{ij} = \gamma_i^{j-1}$:

$$V = \begin{bmatrix} 1 & \gamma_1 & \cdots & \gamma_1^{L-1} \\ 1 & \gamma_2 & \cdots & \gamma_2^{L-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \gamma_L & \cdots & \gamma_L^{L-1} \end{bmatrix}. \tag{1.1}$$

The Vandermonde matrix and its inverse play the important role of mapping between coefficient space and value space. When a non-monomial basis $\{\phi_i(z)\}$ is used, $(V)_{ij} = \phi_{j-1}(\gamma_i)$. We denote by V_i the matrix of first i columns of V. u denotes the unit roundoff, $u \approx 10^{-16}$ in IEEE double precision arithmetic. We write $x = \Theta(y)$ to mean $x = O(y)$ and $y = O(x)$.

2 Existing methods for rational interpolation and least-squares fitting

Rational interpolation is a classical problem in numerical analysis and many algorithms have been proposed, such as those by Jacobi [26], Neville and one based on continued fractions [44, Sect. 2.2]. Here we review those that can be considered among the most practical and stable. For more information on algorithms that are not explained, we refer to [12, Ch. 5], [44, Sect. 2.2] and [37, p. 59].

We first clarify what is meant by rational interpolation and least-squares fitting.

Rational interpolation With sample points γ_i for $i = 1, \dots, L$ with $L = m + n + 1$, the goal of rational interpolation is to find polynomials $p \in \mathcal{P}_m, q \in \mathcal{P}_n$ satisfying the set of L equations

$$f(\gamma_i) = \frac{p(\gamma_i)}{q(\gamma_i)}, \quad i = 1, \dots, L. \tag{2.1}$$

However, as is well known [12, Ch. 5], [47, Ch. 26], (2.1) does not always have a solution $p \in \mathcal{P}_m, q \in \mathcal{P}_n$. To avoid difficulties associated with nonexistence, a numerical algorithm often starts with the linearized equation

$$f(\gamma_i)q(\gamma_i) = p(\gamma_i), \quad i = 1, \dots, L, \tag{2.2}$$

which always has solution(s), which all correspond to the same rational function $\frac{p}{q}$. Most methods discussed in this paper work with (2.2).

Rational least-squares fitting When we sample f at more than $m + n + 1$ sample points $\{\gamma_i\}_{i=1}^L$ with $L > m + n + 1$, (2.2) has more equations than unknowns, and a natural approach is to find p, q such that

$$f(\gamma_i)q(\gamma_i) \approx p(\gamma_i), \quad i = 1, \dots, L. \tag{2.3}$$

This leads to a least-squares problem. Least-squares fitting is used throughout scientific computing, and it often leads to more robust algorithms than interpolation. For example, when function values contain random errors, polynomial least-squares fitting has the benefit of reducing the variance in the outcome as compared with interpolation [14, Sect. 4.5.5].

One main message of this paper is that the precise formulation of the least-squares problem (2.3) is crucial for numerical stability. For example, the minimizers of $\left\| \begin{bmatrix} f(\gamma_1)q(\gamma_1) - p(\gamma_1) \\ \vdots \\ f(\gamma_L)q(\gamma_L) - p(\gamma_L) \end{bmatrix} \right\|$ and $\left\| \begin{bmatrix} f(\gamma_1) - p(\gamma_1)/q(\gamma_1) \\ \vdots \\ f(\gamma_L) - p(\gamma_L)/q(\gamma_L) \end{bmatrix} \right\|$ are clearly different. As we describe below, our method works with $\left\| D \begin{bmatrix} f(\gamma_1)q(\gamma_1) - p(\gamma_1) \\ \vdots \\ f(\gamma_L)q(\gamma_L) - p(\gamma_L) \end{bmatrix} \right\|$ for an $L \times L$ diagonal matrix $D = \mathrm{diag}(d_i)$ chosen so that

$$d_i = \frac{\mathrm{median}_L |f(\gamma)|}{\max(|f(\gamma_i)|, \mathrm{median}_L |f(\gamma)|)}. \tag{2.4}$$

Here $\mathrm{median}_L |f(\gamma)|$ is the median value of $\{|f(\gamma_i)|\}_{i=1}^L$. This choice is crucial for establishing numerical stability.

2.1 Naive method

Perhaps the most straightforward, “naive” method for rational interpolation is to find the coefficients of $p(z) = \sum_{i=0}^{m} c_{p,i} z^i$ and $q(z) = \sum_{i=0}^{n} c_{q,i} z^i$ by writing out (2.2) as a matrix equation

$$FV_{n+1}c_q = V_{m+1}c_p, \tag{2.5}$$

where $F = \mathrm{diag}(f(\gamma_1), \ldots, f(\gamma_L))$ for $L = m + n + 1$ and V_{m+1}, V_{n+1} are the first $(m + 1)$ and $(n + 1)$ columns of V, the Vandermonde matrix of size L as in (1.1). To obtain (2.5), note that the (partial) Vandermonde matrices map the coefficients c_p, c_q to value space (i.e., $V_{n+1}c_q = [q(\gamma_1), \ldots, q(\gamma_L)]^\top$, $V_{m+1}c_p = [p(\gamma_1), \ldots, p(\gamma_L)]^\top$), in which “multiplication by $f(\gamma_i)$” corresponds simply to “multiplication by F”. Equation (2.5) is thus a matrix formulation of rational interpolation (2.2) in value space.

Solving (2.5) for c_p, c_q amounts to finding a null vector of the $L \times (L + 1)$ matrix

$$\left(C \begin{bmatrix} c_q \\ -c_p \end{bmatrix} := \right) \quad \begin{bmatrix} FV_{n+1} & V_{m+1} \end{bmatrix} \begin{bmatrix} c_q \\ -c_p \end{bmatrix} = 0. \tag{2.6}$$

Sometimes the matrix C has null space of dimension larger than 1; in this case all the null vectors of C give the same rational function p/q [12, Sect. V.3.A].

To find the poles of f once c_q is obtained, we find the roots of the polynomial $q = \sum_{i=0}^{n} c_{q,i} x^i$, for example by the companion linearization [22]. When a non-monomial polynomial basis $\{\phi_i(z)\}_{i=0}^{L}$ is chosen, other linearizations such as comrade and confederate are available [5,22].

The above process (2.6) can be easily extended to the oversampled case, in which $L > m + n + 1$ and the matrix C above is of size $L \times (m + n + 2)$. In this case the matrix in (2.6) has at least as many rows as columns, and does not necessarily have a null vector. Then the task is to perform a least-squares fitting, which we do by finding the right singular vector corresponding to the smallest singular value of the matrix C, which for later use we state as an optimization problem:

$$\begin{aligned} &\underset{c_p, c_q}{\text{minimize}} \left\| \begin{bmatrix} FV_{n+1} & V_{m+1} \end{bmatrix} \begin{bmatrix} c_q \\ -c_p \end{bmatrix} \right\|_2 \\ &\text{subject to } \|c_p\|_2^2 + \|c_q\|_2^2 = 1. \end{aligned} \tag{2.7}$$

Here the normalization $\|c_p\|_2^2 + \|c_q\|_2^2 = 1$ is imposed to rule out the trivial solution $c_p = 0$, $c_q = 0$.

We shall consider a scaled formulation of (2.7), which left-multiplies a suitably chosen diagonal matrix D by the matrix in the objective function as

$$\begin{aligned} &\underset{c_p, c_q}{\text{minimize}} \left\| D \begin{bmatrix} FV_{n+1} & V_{m+1} \end{bmatrix} \begin{bmatrix} c_q \\ -c_p \end{bmatrix} \right\|_2 \\ &\text{subject to } \|c_p\|_2^2 + \|c_q\|_2^2 = 1. \end{aligned} \tag{2.8}$$

Note that (2.7) and (2.8) have the same solution when the optimal objective value is zero, but otherwise they are different, and in the oversampled case $L > m+n+1$ this is usually the case. Numerically, they are vastly different even when $L = m+n+1$.

The dominant cost is in the SVD (more precisely computing the right singular vector corresponding to the smallest singular value) of $\left[FV_{n+1}\ V_{m+1}\right]$ or the scaled matrix $D\left[FV_{n+1}\ V_{m+1}\right]$, requiring $O(L(m+n)^2)$ cost.

The naive method (2.5) is mentioned for example in [10], but seems to be rarely used in practice, and we are unaware of previous work that explicitly investigate the least-squares formulation (2.8) or its scaled variant (2.7). Nonetheless, in Sect. 5 we shall show that the scaled formulation (2.8) is numerically stable for rational interpolation (i.e., computing p, q) for a suitable choice of D. In this paper we refer to (2.8) as the *scaled naive method* (or just naive method).

Another method that relies on finding a null vector of a matrix is described in [41], whose matrix elements are defined via the divided differences. Analyzing stability for this method appears to be complicated and is an open problem.

2.2 Chebfun's `ratinterp`

Chebfun [18] is a MATLAB package for working with functions based primarily on polynomial interpolation, but it also provides basic routines for rational functions. In particular, the `ratinterp` command runs a rational interpolation or least-squares fitting algorithm for the linearized equation (2.3), as outlined below.

We start again with the matrix equation in the naive method (2.6), which we rewrite as $FV_{n+1}c_q = V_{m+1}c_p$. Expanding the matrices V_{m+1}, V_{n+1} to form a full Vandermonde matrix V, the equation becomes

$$FV\begin{bmatrix} c_q \\ 0 \end{bmatrix} = V\begin{bmatrix} c_p \\ 0 \end{bmatrix}. \tag{2.9}$$

Now when the sample points are at roots of unity $\gamma_j = \exp(\frac{2\pi \mathrm{i} j}{L})$ for $j = 1, \dots, L$, and using the monomial basis $\{z^i\}_{i=0}^{L-1}$, we can use the FFT to efficiently multiply by V or $V^{-1} = \frac{1}{L}V^*$ ($V^* = \bar{V}^T$ denotes that Hermitian conjugate), and left-multiplying (2.9) by $V^{-1} = \frac{1}{L}V^*$ gives

$$\frac{1}{L}V^*FV\begin{bmatrix} c_q \\ 0_{L-(n+1)} \end{bmatrix} = \begin{bmatrix} c_p \\ 0_{L-(m+1)} \end{bmatrix}. \tag{2.10}$$

The multiplication by V^{-1} brings the equation back to coefficient space, and so unlike the naive method (2.5) given in value space, (2.10) is a formulation of rational interpolation in coefficient space. Note that the matrix $\frac{1}{L}V^*FV$ can be formed in $\mathcal{O}(L^2\log L)$ operations using the FFT. An analogous result holds for Chebyshev points $\gamma_j = \cos(\frac{\pi(j-1)}{L-1})$ using the Chebyshev polynomial basis [6], [46, Ch. 8].

By (2.10), c_q is a null vector of the bottom-left $(L-m-1)\times(n+1)$ part of V^*FV, which has one more column than rows in the interpolation case $L = m+n+1$. Then the task is to find c_q such that

$$\widetilde{V}^* F V_{n+1} c_q = 0, \tag{2.11}$$

where $\widetilde{V}$ denotes the last $L-m-1$ columns of V (as before, V_{n+1} is the first $N+1$ columns).

Again, in the oversampled case a least-squares fitting can be done by finding the smallest singular value and its right singular vector of the $(L-m-1) \times (n+1)$ matrix $\widetilde{V}^* F V_{N+1}$.

As in the naive method, `ratinterp` finds the poles by finding the roots of q via the eigenvalues of the companion (when sampled at roots of unity) or colleague (Chebyshev points) matrices.

2.3 RKFIT

The recent work by Berljafa and Güttel [7,8] introduces RKFIT, a toolbox for working with matrices and rational functions based on rational Krylov decompositions. Given matrices $F, A \in \mathbb{C}^{L\times L}$ and a vector $b \in \mathbb{C}^L$, RKFIT is designed to find a rational matrix approximant $r(A)$ to F such that $r(A)b \approx Fb$ by solving

$$\underset{r\in\mathcal{R}_{m,m}}{\text{minimize}} \left\| \tilde{D}\left(F - r(A)\right)b \right\|_2, \tag{2.12}$$

where $\tilde{D} \in \mathbb{C}^{L\times L}$ is an elementwise weight matrix, which the user can specify. The objective function in (2.12) is called the absolute misfit in [8]. In the special case where $F = \mathrm{diag}(f(\gamma_i))$, $A = \mathrm{diag}(\gamma_i)$ and $b = [1,\ldots,1]^\top$, RKFIT seeks to solve the optimization problem

$$\underset{p,q\in\mathcal{P}_m}{\text{minimize}} \left\| \tilde{D} \begin{bmatrix} f(\gamma_1) - \frac{p(\gamma_1)}{q(\gamma_1)} \\ \vdots \\ f(\gamma_L) - \frac{p(\gamma_L)}{q(\gamma_L)} \end{bmatrix} \right\|_2. \tag{2.13}$$

RKFIT solves (2.13) by an iterative process: starting with an initial guess for poles (e.g. ∞) that determines a temporary $q = q_0$, form a rational Krylov decomposition and solve (2.13) over $p \in \mathcal{P}_m$ via computing an SVD. Using the obtained solution, RKFIT then updates the pole estimates and q_0, then repeats the process until convergence is achieved. See [8] for details, which shows RKFIT can deal with more general problems, for example with multiple vectors b and matrices F.

Note that (2.13) has the flavor of dealing with the original rational approximation problem (2.1) rather than the linearized version (2.2). We observe, nonetheless, that (2.13) becomes very close to (2.8) (same except for the normalization) if we take $\tilde{D} = D\mathrm{diag}(q(\gamma_i))$. As we discuss in Sect. 5, the choice of D (and hence $\tilde{D}$) is crucial for numerical stability. Indeed, RKFIT is not stable with the default parameters when used for scalar rational approximation, but the user can input an appropriate D (which depends on f) to achieve stability.

3 Automatic type determination via oversampling

A significant feature of Chebfun's polynomial approximation process for a continuous function f is that the numerical degree can be obtained automatically by oversampling. This allows the user to obtain the polynomial approximant by taking just the function f as input, without prior information on the (numerical) degree of f.

Specifically, when the user inputs `chebfun(f)` for a function handle f, an adaptive process is executed to find the appropriate degree: Chebfun first samples f at $2^s + 1$ Chebyshev points $\{\cos \frac{j\pi}{2^s}\}_{j=0}^{2^s}$ for a modest integer s, examines the leading Chebyshev coefficients of the interpolant, and if they have not decayed sufficiently, then increments s by 1 to sample at twice as many points, and repeat until the leading Chebyshev coefficients decay to $O(u)$. For details see [2]. We emphasize the important role that oversampling plays for determining the degree; the coefficient decay is observed only after f is sampled more than necessary to obtain the polynomial interpolant.

For rational interpolation or approximation, we argue that it is possible to determine an appropriate type for a rational approximant just as in the polynomial case by oversampling, although the process is not just to look at coefficients but rather based on the SVD of a certain matrix. Related studies exist: Antoulas and Anderson [1] find a minimum-degree interpolant in the barycentric representation by examining a so-called Löwner matrix, given a set of sample points. Similar techniques have been used in Chebfun's `ratinterp` [21] and `padeapprox` [20], and in RKFIT [8], which are designed for removing spurious root-pole pairs, rather than to find a type of a rational approximant.

3.1 Type determination by oversampling and examining singular values

Suppose that we sample a rational function $f = p/q$ at sufficiently many points $\{\gamma_j\}_{j=1}^{L}$, so that $L/2$ is larger than both $\deg(p), \deg(q)$. We initially take $m = n = \lfloor (L-1)/2 \rfloor$ as tentative upper bounds for the degrees. Then, as in the naive method (2.6), we compute the null space of C (which is square or tall, corresponding to the oversampled case). In "Appendix B" we examine the rank of the matrix C as the integers m, n, L vary, which shows that assuming L is taken large enough so that $L \geq \max\{M + n, m + N\} + 1$, (recalling that (M, N) is the exact type of f)

1. If $m < M$ or $n < N$, then
$$\dim \operatorname{null}(C) = 0. \tag{3.1}$$
2. If $m \geq M$ and $n \geq N$, then
$$\dim \operatorname{null}(C) = \min(m - M, n - N) + 1 \geq 1. \tag{3.2}$$

(See [8, Thm. 3.1] for a similar result in the RKFIT setting). Note how this result gives us information about the type of a rational f: By the first result, if $\dim \mathrm{null}(C) = 0$, we need to take m, n larger, along with L. On the other hand, if $\dim \mathrm{null}(C) > 1$, then (3.2) shows how to reduce n so that there is no redundancy: $n := n - \dim \mathrm{null}(C) + 1$ should give us the correct $n(= N)$ provided that m was set large enough. Even if $m - M < n - N$, if $\ell > 1$ singular values of C are negligible then we reduce n by $\ell - 1$ and repeat the process, which will eventually give us the correct $n = N$ provided that $m > M$. Once $n = N$ is determined, we can find m as the smallest integer such that the $L \times (N + 1 + m + 1)$ matrix C has a null vector. m can be obtained also by looking at the leading coefficients of the computed c_p, but we have found this SVD-based approach to be more reliable. We emphasize the important role played by oversampling, which is necessary for (3.1) and (3.2) to hold.

The above process would find the exact type in exact arithmetic if f is rational. In practice, f may not be rational, and we compute $\dim \mathrm{null}(C)$ numerically by the number of singular values that are smaller than a tolerance $tol = O(u)$ to find a "numerical type" of f, which is the type (m, n) of a rational function $r \in \mathcal{R}_{m,n}$ such that $r \approx f$ in Ω. It is worth noting that "numerical type" is an ambiguous notion: for example, (1) $r_1 \in \mathcal{R}_{20,5}$ and $r_2 \in \mathcal{R}_{5,20}$ may be equally good as approximants to f in the domain Ω, and (2) if f is analytic in Ω, polynomials would suffice if the degree is taken large enough, but rational functions give much better approximants if singularities lie near Ω, see [3, Sect. 6]. (1) Suggests that the "smallest" m, n is not uniquely defined without further restriction. A natural approach is to find an approximant with the smallest possible n (since we do not want unnecessary poles), but (2) suggests that this may lead to an approximant p/q of excessively high $\deg(p)$.

Given f, we attempt to find a rational approximant with as few poles as possible, within a controlled amount of computational effort. Specifically, our Algorithm 3.1 below finds a rational approximant p/q of type (m, n) with the following properties:

1. There exists $p/q \in \mathcal{R}_{m,n}$ such that $\|fq - p\|_L \le tol$ and $\left\| \begin{bmatrix} c_p \\ c_q \end{bmatrix} \right\|_2 = 1$, and
2. No rational function $r = \frac{p}{q} \in \mathcal{R}_{\tilde{m},\tilde{n}}$ with $\tilde{n} < n$ and $\tilde{m} \le \lfloor (L-1)/2 \rfloor$ ($\in [\max(m,n), 2\max(m,n)]$) satisfies $\|fq - p\|_L \le tol$ and $\left\| \begin{bmatrix} c_p \\ c_q \end{bmatrix} \right\|_2 = 1$.

In other words, no rational function with lower denominator degree is a good approximant unless the numerator degree is more than doubled. In what follows, we set $tol = 10^{-14}$ unless otherwise mentioned.

Numerically in practice, we shall show in Sect. 4.2 that it is important that a preprocessing step is carried out before examining the singular values of C. Specifically, we first scale f as $f \leftarrow f/\mathrm{median}_L|f(\gamma)|$ so that the median of the scaled f is 1 in absolute value, and left-multiply a diagonal matrix so that each row of C has roughly the same norm:

$$DC = \mathrm{diag}\left(\frac{1}{\max(|f(\gamma_i)|, 1)}\right) C = [DFV_{n+1}\ DV_{m+1}]. \tag{3.3}$$

This choice of D is the same as the one we use in the scaled naive method (2.8) for stability, to be justified in Sect. 5. Diagonal scaling has the effect of reducing the condition number (when ill-conditioning is caused by the entries having widely varying magnitudes rather than the rows being linearly dependent), and a simple scaling that scales the rows to have identical norms is known to be nearly optimal [16,50]; the scaling in (3.3) achieves this approximately.

For further stability, we orthogonalize the two block columns by the "thin" QR factorizations.[1] $DFV_{n+1} = Q_{n+1}R_{n+1}$, $DV_{m+1} = \tilde{Q}_{m+1}\tilde{R}_{m+1}$, where $Q_{n+1} \in \mathbb{C}^{L\times(n+1)}$, $\tilde{Q}_{m+1} \in \mathbb{C}^{L\times(m+1)}$. Then we define

$$\tilde{C} = [Q_{n+1}\ \tilde{Q}_{m+1}] \tag{3.4}$$

and determine the rational function type by the singular values of $\tilde{C}$. Note that (3.2) continues to hold with C replaced with $\tilde{C}$ in exact arithmetic.

Summarizing, Algorithm 3.1 is the pseudocode for our type determination algorithm.

Algorithm 3.1 `[m,n,sigma] = typefind(f,tol)`: find a type for a rational approximant of f

```
1: Set $L = 8$. Scale $f \leftarrow f/\mathrm{median}_L|f(\gamma)|$.
2: Sample $f$ at the $L$th roots of unity. Set $m = \lfloor L/2 \rfloor - 1$, $n = L - m - 3$ and form the matrix $\tilde{C}$ as in (3.4).
3: Compute $\dim \mathrm{null}(\tilde{C})$ by the number of singular values smaller than $tol$, and
    (a) if $\dim \mathrm{null}(\tilde{C}) = 0$, then take $L := 2L$ and return to step 2 (if $L > 2^{12}$ then stop and return a warning).
    (b) if $\dim \mathrm{null}(\tilde{C}) \geq 1$, then $n := n - \dim \mathrm{null}(\tilde{C}) + 1$ and repeat until $\dim \mathrm{null}(\tilde{C}) = 1$.
4: Reduce $m$ to the smallest integer such that $\tilde{C}$ has a null vector.
5: Set sigma $= \sigma_{m+n+2}(\tilde{C})$.
```

In Algorithm 3.1 we increase the number of sample points by a factor 2 until $\tilde{C}$ has a nontrivial null vector. Doubling the points allows us to reuse the previously sampled values $f(\gamma_i)$ when γ_i are roots of unity; for the same reason, when sampling at Chebyshev points on $[-1, 1]$ (this variant replaces the Lth roots of unity in step 2 by L Chebyshev points), we sample at $2^s + 1$ points as in Chebfun.

We note that (3.1) and (3.2) assume that sufficiently many sample points are taken so that $L \geq \max\{M + n, m + N\} + 1$. If this does not hold, it is possible that $\dim \mathrm{null}(C) > 0$ although $m < M$ or $n < N$, causing Algorithm 3.1 to wrongly conclude f is of a lower type. Fortunately, even if $L < \max\{M + n, m + N\} + 1$, it is unlikely that $\dim \mathrm{null}(C) > 0$, as this requires that $fq \approx p$ at L points, where p, q together have $< L$ degrees of freedom. Similarly, a tall rectangular matrix is

[1] We note that these QR factorizations can be computed exploiting the Vandermonde-like structure of DFV_{n+1}, DV_{m+1}. Namely, when the basis is degree-graded, i.e., $\deg \phi_i(z) = i$, then the column space of DV_{m+1} is equal to the Krylov subspace $\mathcal{K}(\Gamma, b) = [b, \Gamma b, \Gamma^2 b, \ldots, \Gamma^{n-1} b]$ where b is the first column of DV_{m+1}. An orthonormal space for the Krylov subspace $\mathcal{K}(\Gamma, b)$ can thus be computed using the Arnoldi process [19, Sect. 10.5], as done for example in [24, App. A]. The same holds for DFV_{n+1}.

unlikely to have nontrivial null vectors: a random rectangular matrix is full rank with probability one, and well conditioned with high probability if the aspect ratio is safely above 1 [39]. The default value $n = L - m - 3$ was chosen to ensure $\tilde{C}$ is always tall rectangular.

The description of Step 3(b) is not necessarily the most efficient: we can instead take $m \leftarrow m - \ell$ for some $\ell (> \dim \operatorname{null}(\tilde{C}) - 1)$, if this results in $\dim \operatorname{null}(C) > 0$. In step 4, we can use bisection to determine the smallest integer m. The worst-case cost is thus computing $O(\log_2 m)$ SVDs.

When the evaluation of f incurs nonnegligible (relative) error, tol should be adjusted accordingly. The output `sigma` indicates the error bound; a successful degree detection implies `sigma` $< tol$.

Mathematically in exact arithmetic, the matrix C or $\tilde{C}$ having null space of dimension greater than 1 indicates the presence of a spurious root-pole pair, and in fact the coefficients of p, q obtained from *any* null vector of C are known to result in the same rational function p/q. In finite precision arithmetic, however, this property gets lost and a numerical null vector gives a function p/q that may be far from the function f. Furthermore, the accuracy of a computed singular vector is known to be inversely proportional to the gap between the corresponding singular value and the rest [43, Ch. 5]. Besides making the solution unique, finding the smallest possible degree has the additional benefit of widening the distance between the smallest and the second smallest singular values.

3.2 Experiments with oversampling for degree determination

Here we illustrate `typefind` through some numerical experiments. For illustration purposes, instead of doubling $L = 2^3, 2^4, 2^5, \ldots$ as in Algorithm 3.1, we formed $\tilde{C}$ for each integer $L = 2, 3, \ldots$ with $\gamma_j = \exp(\frac{2\pi i j}{L})$, and examined the resulting output type without doubling L. For convenience, below we refer to this process as `typefind(f,tol,L)`, where the number of sample points L is an input.

Algorithm 3.2 `[m,n,sigma] = typefind(f,tol,L)`: `typefind` with fixed sample points

```
1: Sample $f$ at the $L$th roots of unity $\{\gamma_i\}_{i=1}^{L}$, and scale $f \leftarrow f/\operatorname{median}_L|f(\gamma)|$. Set $m = \lfloor L/2 \rfloor - 1$,
   $n = \max(L - m - 3, 0)$ and form the matrix $\tilde{C}$ as in (3.4).
2: Compute $\dim \operatorname{null}(\tilde{C})$ by the number of singular values smaller than $tol$, and
   (a) if $\dim \operatorname{null}(\tilde{C}) = 0$, then output a warning that $L$ is too small; exit with sigma $= \sigma_{m+n+2}(\tilde{C})$
       ($m, n$ are unchanged).
   (b) if $\dim \operatorname{null}(\tilde{C}) \geq 1$, then determine $n, m,$ sigma as in steps 3(b)-5 of typefind(f,tol).
```

When f is a rational function We first examine the simplest case where f is a rational function

$$f(z) = \sum_{i=1}^{N} \frac{1}{z - \xi_i} = \frac{N z^{N-1}}{z^N - 0.9^N}, \tag{3.5}$$

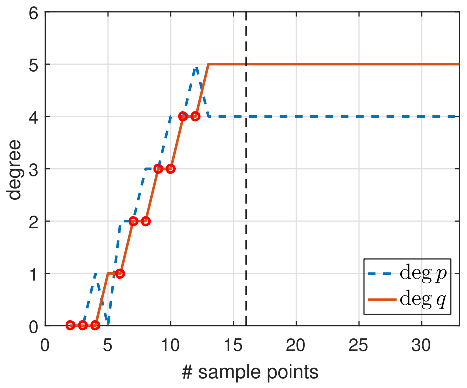

Fig. 1 Types of the rational approximants found by `typefind(f,tol,L)` for rational function (3.5), as the number of sample points L is varied (throughout, $tol = 10^{-14}$). The red circles indicate that `typefind(f,tol,L)` found that the number of sample points is insufficient. The vertical black dashed line indicates the number of sampled points L taken by the automatic degree determination process `typefind(f,tol)`; here $L = 16$

where $N = 5$ and $\xi_i = 0.9\exp(2\pi \mathrm{i} i/N)$ are equispaced on the circle of radius 0.9 centered at the origin. f is a rational function of exact type $(M, N) = (4, 5)$. Figure 1 shows the types obtained by `typefind(f,tol,L)` as we increase the number of sample points L.

Observe that with 13 sample points or more, the algorithm correctly finds the type $(4, 5)$ of the rational function f. With five sample points, however, `typefind(f,tol,L)` erroneously concludes that the function is of lower type; this is an artifact of the symmetry of the function (which disappears e.g. by changing the location of one pole), and illustrates the importance of oversampling. We will come back to this issue in Fig. 5.

Algorithm 3.1 samples at $2^4 = 16$ points to determine the type of the rational approximant. Although 16 is larger than the smallest number $M + N + 1 = 10$ of sample points to theoretically obtain the rational interpolant p/q *if* the degree were known, we believe this is a small price to pay for an automated degree-finding algorithm.[2]

When f is a meromorphic function The situation becomes more complicated when f is not rational but merely meromorphic. For example consider

$$f(z) = \frac{\exp(z)}{z - \xi_1} + \sum_{i=2}^{N} \frac{1}{z - \xi_i}. \tag{3.6}$$

We take an example again with $N = 5$. f can be regarded as being of numerical type $(\hat{M}, 5)$ where $\hat{M} \approx 20$, because the exponential function can be resolved to $O(u)$ accuracy by a degree $\approx$ 15 polynomial in the unit disk. Moreover, we expect that by increasing the denominator degree one can reduce the numerator degree for the same approximation quality, so we could also approximate f in the unit disk by a type $(20 - \delta_M, 5 + \delta_N)$ rational function where δ_M, δ_N are modest integers such as 1, 2.

[2] When reducing the number of sample points is of primary importance (i.e., when sampling f is expensive), we can proceed as follows: having sampled f at L points, take $n \leftarrow n + \ell$, $m \leftarrow m - \ell$ for integers ℓ with $|\ell| \le \frac{L}{2}$, re-form $\tilde{C}$ and examine the condition $\dim \operatorname{null}(\tilde{C}) \ge 1$; if this is satisfied for some ℓ, there exists an acceptable rational approximant of type $(n + \ell, m - \ell)$.

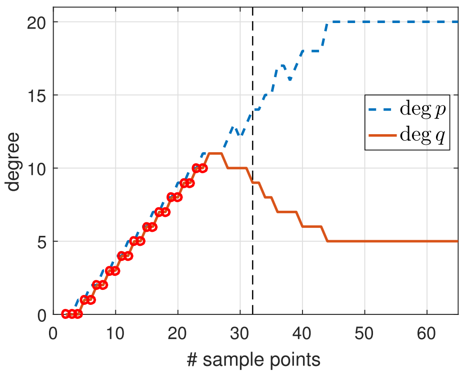

Fig. 2 Type found by `typefind(f,tol,L)` for a meromorphic function f (3.6)

Figure 2 shows the numerical degrees obtained by `typefind(f,tol,L)`, which confirms this observation. Algorithm 3.1 (i.e., `typefind(f,tol)`) outputs the type $(m, n) = (14, 9)$ by sampling at $2^5 = 32$ points. Our polefinder `ratfun` (described in Sect. 4) computes nine poles, five of which approximate the correct poles ξ_i to within 10^{-14} and four of which have absolute value larger than 10. The same is true of all the types found by `typefind(f, tol, L)` for $L \geq 25$; this example suggests they are all appropriate types, illustrating the nonunique nature of the numerical type. *When f is an analytic function with poles near Ω* Finally, we consider the function

$$f(z) = \frac{\exp(z)}{z - 1.1}, \tag{3.7}$$

which is analytic in the unit disk Ω, therefore a polynomial p exists such that $\|f - p\|_\Omega < \epsilon$ for any $\epsilon > 0$. However, as described in [3, Sect. 6], rational functions do a much better job of approximating analytic functions with a singularity lying near Ω, and (3.7) is such an example. Indeed, to achieve $\|f - p\|_\Omega < \epsilon$ for a polynomial p, we need $\deg(p) \geq 386$, whereas with rationals, $\|f - p/q\|_\Omega < \epsilon$ is achieved for a $p/q \in \mathcal{R}_{16,1}$. Figure 3 shows the types obtained by `typefind(f,tol,L)`, which outputs the type $(16, 1)$ for $L > 36$. The output would become $(\deg(p), 0)$ for $\deg(p) \approx 400$ if we take $L \geq 800$, but `typefind(f,tol)` terminates doubling the sample points once $\dim \operatorname{null}(\tilde{C}) \geq 1$ with $L = 32$, giving type $(13, 3)$. Again, the two extra poles are far outside the unit disk.

See Fig. 7 for a function with exact poles far from the unit disk, along with other experiments in Sect. 6.

3.3 Interpretations as optimization problems

We have emphasized the role of the diagonal scaling D in the discussion in Sect. 3.1. Here we reassess its significance from the viewpoint of optimization problems. Let us consider the meaning of the smallest singular value $\sigma_{m+n+2}(C)$ of C in (2.6), allowing for the oversampled case. As discussed in (2.7), it has the characterization

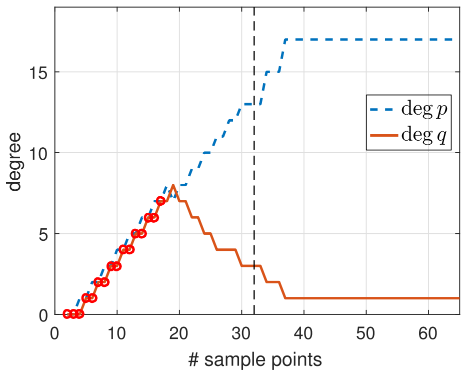

Fig. 3 Types found by `typefind(f,tol,L)` for the function f in (3.7), analytic in the unit disk

$$\sigma_{m+n+2}(C) = \min\left\{ \left\| C \begin{bmatrix} c_q \\ -c_p \end{bmatrix} \right\|_2 : \left\| \begin{bmatrix} c_q \\ -c_p \end{bmatrix} \right\|_2 = 1 \right\}, \tag{3.8}$$

and the solution $\begin{bmatrix} c_q \\ -c_p \end{bmatrix}$ is obtained by the corresponding right singular vector. Since we have $C\begin{bmatrix} c_q \\ -c_p \end{bmatrix} = [f(\gamma_j)q(\gamma_j) - p(\gamma_j)]_{j=1}^{L}$ by the definition of C, its smallest singular value σ_{m+n+2} is equal to the optimal value of the following optimization problem:

$$\begin{aligned} \underset{p,q}{\text{minimize}} \quad & \|fq - p\|_L \\ \text{subject to} \quad & p \in \mathcal{P}_m, \quad q \in \mathcal{P}_n, \qquad \|c_p\|_2^2 + \|c_q\|_2^2 = 1, \end{aligned} \tag{3.9}$$

where $\|g\|_L := \sqrt{\sum_{i=1}^{L} |g(\gamma_i)|^2}$. Note that the constraint in (3.9) changes depending on the choice of the polynomial basis, and the norm in the constraint $\|\cdot\|_2$ differs from that of the objective function $\|\cdot\|_L$.

Recall that for stability, instead of C, we work with the scaled-orthogonalized matrix $\tilde{C}$ in (3.4). We claim that the smallest singular value $\sigma_{m+n+2}(\tilde{C})$ of $\tilde{C}$ is equal to the optimal value of the following optimization problem:

$$\begin{aligned} \underset{p,q}{\text{minimize}} \quad & \|dfq - dp\|_L \\ \text{subject to} \quad & p \in \mathcal{P}_m, \quad q \in \mathcal{P}_n, \qquad \|dp\|_L^2 + \|dfq\|_L^2 = 1, \end{aligned} \tag{3.10}$$

where $d(z)$ is a function such that $d(\gamma_i)$ is equal to the ith diagonal element of D.

To verify the claim, we express $\sigma_{m+n+2}(\tilde{C})$ as

$$\begin{aligned} \sigma_{m+n+2}(\tilde{C}) &= \min\{\|\tilde{C}x\|_2 : \|x\|_2 = 1\} \\ &= \min\{\|Q_{m+1}x_2 - Q_{n+1}x_1\|_2 : \|x_1\|_2^2 + \|x_2\|_2^2 = 1\} \\ &= \min\{\|Q_{m+1}x_2 - Q_{n+1}x_1\|_2 : \|Q_{n+1}x_1\|_2^2 + \|Q_{m+1}x_2\|_2^2 = 1\}, \end{aligned} \tag{3.11}$$

where the last equality comes from the orthonormality of the columns of Q_{n+1} and Q_{m+1}. From the definition of Q_{n+1} and Q_{m+1}, we have

$$\begin{aligned}\operatorname{range}(Q_{n+1}) &= \{[d(\gamma_j)f(\gamma_j)q(\gamma_j)]_{j=1}^L : q \in \mathcal{P}_n\},\\ \operatorname{range}(Q_{m+1}) &= \{[d(\gamma_j)p(\gamma_j)]_{j=1}^L : p \in \mathcal{P}_m\}.\end{aligned}$$

Hence, $\sigma_{m+n+2}(\tilde{C})$ in (3.11) is equal to the optimal value of the problem given by (3.10). We note that, if the optimal value $\sigma_{m+n+2}(\tilde{C})$ is sufficiently small, then the optimal solutions p and q are scaled so that $\|dp\|_L^2 \approx \|dfq\|_L^2 \approx 1/2$, because $|\|dp\|_L - \|dfq\|_L| \leq \|dp - dfq\|_L = \sigma_{m+n+2}(\tilde{C})$.

We can also show similarly (and more easily) for the scaled (but not orthogonalized) matrix DC in (3.3) that $\sigma_{\min}(DC)$ is equal to the optimal value of

$$\begin{aligned}&\underset{p,q}{\text{minimize}} && \|dfq - dp\|_L\\ &\text{subject to} && p \in \mathcal{P}_m, \quad q \in \mathcal{P}_n, \qquad \|c_p\|_2^2 + \|c_q\|_2^2 = 1.\end{aligned} \tag{3.12}$$

The optimization problems (3.9), (3.10) and (3.12) differ in the following respects:

- The objective function in (3.10) and (3.12) is scaled so that $\|[d(\gamma_j), d(\gamma_j)f(\gamma_j)]\|_2 = \Theta(1)$.
- The constraint in (3.10) does not depend on the choice of the polynomial basis.
- In (3.10), the objective function and constraint employ the same norm $\|\cdot\|_L$.

The diagonal scaling in the objective function is crucial for numerical stability as we show in Sect. 5. The independence of $\tilde{C}$ from the polynomial basis is due to the QR factorization, and it "automatically" chooses polynomial bases $\{\phi_{p,i}\}_{i=0}^{L-1}$ and $\{\phi_{q,i}\}_{i=0}^{L-1}$ for p and q respectively, for which discrete orthonormality is achieved: for p, defining $v_i := \begin{bmatrix} d(\gamma_1)\phi_{p,i}(\gamma_1) \\ \vdots \\ d(\gamma_L)\phi_{p,i}(\gamma_L)\end{bmatrix}$ we have orthonormality $v_i^* v_j = \delta_{i,j}$ (the Kronecker delta, $\delta_{i,j} = 0$ if $i \neq j$ and $\delta_{i,i} = 1$), and similarly for q, the vectors $w_i := \begin{bmatrix} d(\gamma_1)f(\gamma_1)\phi_{q,i}(\gamma_1) \\ \vdots \\ d(\gamma_L)f(\gamma_L)\phi_{q,i}(\gamma_L)\end{bmatrix}$ are orthonormal $w_i^* w_j = \delta_{i,j}$. Note that the two bases for p, q are different, and they depend on the function f and sample points $\{\gamma_i\}_{i=1}^L$. Working with orthonormal matrices have numerical benefits, as we shall illustrate in Sect. 5.3. Together with the fact that the objective function and constraint are defined with respect to the same norm $\|\cdot\|_L$, this "scaled and QR'd" approach results in a natural and numerically stable interpolation. For these reasons, we argue that (3.10) is a natural way to formulate our problem.

Note, however, that the scaled naive method (2.8) works with (3.12), not (3.10). No QR factorizations is performed in (2.8), because if one uses it, the null vector of $\tilde{C}$ no longer gives the coefficients c_p, c_q as in (2.8). Although we could retrieve c_p, c_q by applying the inverse transformation with respect to the R factors in the QR factorizations, this leads to numerical instability when FV_{n+1}, V_{m+1} are ill-conditioned.

In the next section we shall overcome this difficulty by formulating an algorithm that directly computes the poles, bypassing the coefficient vector c_q. The resulting algorithm `ratfun` essentially works with (3.10), but is immune to the difficulty associated with the change of polynomial basis.

4 Polefinding via a generalized eigenvalue problem

We now describe our eigenvalue-based algorithm for finding the poles of f. Here we take m, n as given, assumed to be obtained by Algorithm 3.1 or given as inputs.

4.1 Formulating polefinding as an eigenproblem

We consider finding the poles of $f(z) = \frac{p(z)}{q(z)} \in \mathcal{R}_{m,n}$, i.e., the roots of $q(z)$. Denote the desired poles by ξ_i for $i = 1, \dots, n$.

As before we start with the linearized interpolation equation (2.2). Here we consider interpolation where $L = m+n+1$; we treat the oversampled case later in Sect. 4.3. The key idea is to make a pole ξ_k, the sought quantity, appear explicitly in the equation to be solved. To this end we rewrite $q(z)$ using $\tilde{q}(z) := \frac{q(z)}{z-\xi_k}$, which is also a polynomial, as

$$q(z) = z\tilde{q}(z) - \xi_k \tilde{q}(z). \tag{4.1}$$

We can then express (2.2) as

$$\gamma_i f(\gamma_i)\tilde{q}(\gamma_i) - p(\gamma_i) = \xi_k f(\gamma_i)\tilde{q}(\gamma_i), \quad i = 1, \dots, m+n, \tag{4.2}$$

which is the crucial guiding equation for our algorithm. The equations (4.2) can be written as a matrix equation using the Vandermonde matrix as

$$\Gamma F V_n c_{\tilde{q}} - V_{m+1} c_p = \xi_k F V_n c_{\tilde{q}}, \tag{4.3}$$

where $\Gamma = \mathrm{diag}(\gamma_i)$, $c_{\tilde{q}}$ is the vector of coefficients for the polynomial $\tilde{q}$, and as before, $F = \mathrm{diag}(f(\gamma_i))$ and V_i is the first i columns of the Vandermonde matrix. Just as in the naive method (2.5), we obtain (4.3) by mapping into value space using the Vandermonde matrix, then noting that in value space, "$f(\gamma_i)$-multiplication" is "F-multiplication" and "γ_i-multiplication" is "Γ-multiplication". Thus (4.3) formulates rational interpolation again in value space, but now with ξ_k shown explicitly.

Of course, ξ_k is unknown in (4.3), and setting it as an unknown λ we arrive at the generalized eigenvalue problem

$$\begin{bmatrix} A_1 & A_2 \end{bmatrix} x = \lambda \begin{bmatrix} B_1 & O \end{bmatrix} x, \tag{4.4}$$

where $A_1 = F\Gamma V_n$, $A_2 = V_{m+1}$ and $B_1 = FV_n$, and O is the zero matrix of size $L \times (m+1)$.

Since the matrix $[B_1 \ O]$ clearly has null space of dimension $m+1$, the eigenproblem (4.4) has $m+1$ eigenvalues at infinity. By construction, we expect the finite eigenvalues to contain information about the poles. The next result shows indeed that the finite eigenvalues of (4.4) are the poles of f.

Proposition 1 *If $f(z) = \frac{p(z)}{q(z)} \in \mathcal{R}_{m,n}$ has n poles counting multiplicities (i.e., $n = N$), then the matrix pencil $[A_1, A_2] - \lambda[B_1, O]$ is regular, and its finite eigenvalues coincide with the poles of f.*

(Proof) Since $f(z) \in \mathcal{R}_{m,n}$ has n poles, $f(z)$ has the expression

$$f(z) = \frac{p(z)}{q(z)} = \frac{a\prod_{j=1}^{d}(z-\eta_j)}{\prod_{i=1}^{n}(z-\xi_i)}$$

for some $d \le m$, where $a \ne 0$ and η_j does not coincide with any element of $\{\xi_1, \dots, \xi_n\}$ for $j = 1, \dots, d$, i.e., $\{\eta_1, \dots, \eta_d\} \cap \{\xi_1, \dots, \xi_n\} = \emptyset$. It suffices to show that $[A_1, A_2] - \lambda[B_1, O]$ is singular if and only if λ is one of the roots $\xi_1, \dots, \xi_n$ of q. We can easily confirm the "if" part as follows. Let $\lambda = \xi_k$ for a fixed integer k. Defining the coefficient vectors $c_{\tilde{q}} = [c_{\tilde{q},0}, \dots, c_{\tilde{q},n-1}]^\top$ and $c_p = [c_{p,0}, \dots, c_{p,m}]^\top$ such that $\frac{q(z)}{z-\xi_k} = \sum_{i=0}^{n-1} \tilde{c}_{q,i}\phi_i(z)$ and $p(z) = \sum_{i=0}^{m} c_{p,i}\phi_i(z)$, we have

$$\left(([A_1, A_2] - \xi_k[B_1, O]) \begin{bmatrix} c_{\tilde{q}} \\ -c_p \end{bmatrix} \right)_i$$
$$= f_i\gamma_i \frac{q(\gamma_i)}{\gamma_i - \xi_k} - p(\gamma_i) - \xi_k f_i \frac{q(\gamma_i)}{\gamma_i - \xi_k} = f_i q(\gamma_i) - p(\gamma_i) = 0$$

for $i = 1, \dots, m+n+1$, so it follows that $([A_1, A_2] - \xi_k[B_1, O])$ has a nontrivial kernel, and hence, $([A_1, A_2] - \xi_k[B_1, O])$ is singular for $k = 1, \dots, n$.

Next, for the "only if" part, suppose (4.4) holds for a nonzero $x = [x_1, x_2]^\top \in \mathbb{C}^{m+n+1}$ and $\lambda \in \mathbb{C}$, where we write $x_1 = [c_0, \dots, c_{n-1}]$ and $x_2 = [d_0, \dots, d_m]$. Then, it suffices to show that λ is one of the roots $\xi_1, \dots, \xi_n$ of q. Define polynomials $r_{x_1}(z)$ and $r_{x_2}(z)$ by $r_{x_1}(z) = \sum_{i=0}^{n-1} c_i z^i$ and $r_{x_2}(z) = \sum_{i=0}^{m} d_i z^i$, respectively. We shall show that $\lambda = \xi_i$ for some i, and that $(z-\xi_i)r_{x_1}(z) = Cq(z)$, $r_{x_2}(z) = Cp(z)$ for some nonzero scalar C. From (4.4), we have

$$zr_{x_1}(z)f(z) + r_{x_2}(z) = \lambda r_{x_1}(z)f(z)$$

for $z = \gamma_1, \dots, \gamma_{m+n+1}$. Multiplying $q(z)$ to both sides, we obtain

$$(z-\lambda)r_{x_1}(z)p(z) + r_{x_2}(z)q(z) = 0 \tag{4.5}$$

for $z = \gamma_1, \dots, \gamma_{m+n+1}$. Since the left-hand side of (4.5) is a polynomial of degree at most $m+n$ and take on the value 0 at $m+n+1$ distinct points, it must be the zero polynomial, i.e., (4.5) holds for arbitrary $z \in \mathbb{C}$. Hence, the polynomial $(z-\lambda)r_{x_1}(z)p(z)$ is equal to the polynomial $-r_{x_2}(z)q(z)$. Note that these two polynomials are not the zero polynomial since $x \ne 0$. Let $\alpha_1, \dots, \alpha_{d_1}$ be the roots of r_{x_1} and $\beta_1, \dots, \beta_{d_2}$ the roots of r_{x_2}. Since $(z-\lambda)r_{x_1}(z)p(z)$ has the same roots as $-r_{x_2}(z)q(z)$, we have $\{\lambda\} \cup \{\alpha_1, \dots, \alpha_{d_1}\} \cup \{\eta_1, \dots, \eta_d\} = \{\beta_1, \dots, \beta_{d_2}\} \cup \{\xi_1, \dots, \xi_n\}$. Since $\{\eta_1, \dots, \eta_d\} \cap \{\xi_1, \dots, \xi_n\} = \emptyset$, we have $\{\xi_1, \dots, \xi_n\} \subseteq \{\lambda\} \cup \{\alpha_1, \dots, \alpha_{d_1}\}$. Since the number d_1 of roots of r_{x_1} is at most $n-1$, we have $\{\xi_1, \dots, \xi_n\} = \{\lambda\} \cup \{\alpha_1, \dots, \alpha_{d_1}\}$, so it follows that $\lambda \in \{\xi_1, \dots, \xi_n\}$.

We have thus shown that for every λ and $x \neq 0$ such that (4.4) holds, λ must be a pole of f. It hence follows that for any $\lambda \neq \xi_i$, the matrix $[A_1, A_2] - \lambda[B_1, 0]$ is nonsingular, showing the matrix pencil is regular. □

As shown in the proof, the eigenvectors of (4.4) have a special structure: the eigenvector corresponding to ξ_i is

$$\begin{bmatrix} A_1 & A_2 \end{bmatrix} \begin{bmatrix} c_{\tilde{q}} \\ -c_p \end{bmatrix} = \xi_i \begin{bmatrix} B_1 & O \end{bmatrix} \begin{bmatrix} c_{\tilde{q}} \\ -c_p \end{bmatrix}. \tag{4.6}$$

In the appendix we give further analysis of the eigenproblem, revealing the Kronecker canonical form. It shows in particular that the orders of the poles are equal to the multiplicities of the eigenvalues.

4.2 Techniques for efficient and stable solution of eigenproblem

We now discuss how to solve (4.4) in practice. We employ techniques to remove undesired eigenvalues at ∞, and to achieve numerical stability.

Projecting out eigenvalues at infinity The generalized eigenvalue problem (4.4) has n eigenvalues ξ_i along with $m+1$ eigenvalues at infinity. These eigenvalues at infinity can be projected out easily. Let $A_2^{\perp} \in \mathbb{C}^{L\times(L-n)}$ be the orthogonal complement of A_2 such that $A_2^* A_2^{\perp} = 0$. Then

$$(A_2^{\perp})^* A_1 x = \lambda (A_2^{\perp})^* B_1 x \tag{4.7}$$

is an $n \times n$ eigenvalue problem whose eigenvalues are $\{\xi_i\}_{i=1}^n$ with corresponding eigenvectors $c_{\tilde{q}_i}$. To see this, recall (4.6) and note that (4.4) is equivalent to

$$\begin{bmatrix} A_2 & A_1 \end{bmatrix} \begin{bmatrix} -c_p \\ c_{\tilde{q}} \end{bmatrix} = \lambda \begin{bmatrix} O & B_1 \end{bmatrix} \begin{bmatrix} -c_p \\ c_{\tilde{q}} \end{bmatrix}, \tag{4.8}$$

and so taking the QR factorization $A_2 = [Q_1 \; A_2^{\perp}]R$, by left-multiplying $[Q_1 \; A_2^{\perp}]^*$ we obtain

$$\begin{bmatrix} R & Q_1^* A_1 \\ O & (A_2^{\perp})^* A_1 \end{bmatrix} \begin{bmatrix} -c_p \\ c_{\tilde{q}} \end{bmatrix} = \lambda \begin{bmatrix} O & Q_1^* B_1 \\ O & (A_2^{\perp})^* B_1 \end{bmatrix} \begin{bmatrix} -c_p \\ c_{\tilde{q}} \end{bmatrix} \tag{4.9}$$

from which we can deflate the $m+1$ eigenvalues corresponding to the top-left corner and solve for the lower-right part, to arrive at (4.7) with eigenvector $c_{\tilde{q}}$. Alternatively, (4.8) shows that the "residual" $A_1 x_1 - \lambda B_1 x_1 \in \mathrm{Span}(A_2)$, which means it is orthogonal to $A_2^{\perp}$; (4.7) is its representation.

Diagonal scaling Generally, given an eigenvalue problem $A - \lambda B$, a well known technique of balancing the elements in the presence of widely varying elements is diagonal scaling.

As with the scaled naive method (2.8), we left-multiply a diagonal matrix D and work with the pencil $D(A - \lambda B)$ so that each row of $D[A\ B]$ has about the same norm. In Sect. 5 we show that this scaling makes our approach numerically stable.

Orthogonalization As alluded to at the end of Sect. 3.3, the final technique that we use for improved stability, which is inapplicable in the naive method, is orthogonalization. As in (3.4), we take the thin QR factorizations $DA_2 = Q_{A_2}R_{A_2}$, $DB_1 = Q_{B_1}R_{B_1}$, where Q_{A_2}, Q_{B_1} have orthonormal columns and are of the same size as DA_2, DB_1. The rationale is that numerical errors are reduced by working with orthogonal matrices. These can be computed exploiting the Vandermonde structure, as explained after (3.4).

Applying scaling and orthogonalization to (4.9), the eigenvalue problem we solve becomes

$$\tilde{A}x = \lambda\tilde{B}x, \quad \text{where } \tilde{A} = (Q_{A_2}^{\perp})^* \Gamma Q_{B_1}, \tilde{B} = (Q_{A_2}^{\perp})^* Q_{B_1}x. \tag{4.10}$$

This is a $n \times n$ eigenproblem; the n eigenvalues are precisely the n sought poles.

Recall that as a consequence of this orthogonalization, the eigenvector of (4.10) goes through the change of basis with respect to $R_{B_1}^{-1}$. This severely affects the naive method (for which the singular vector is the sought quantity), but not our algorithm (for which the eigenvalues are sought).

Use of FFT? In the practically important cases where the sample points are at roots of unity or Chebyshev points, we can use the FFT to efficiently obtain the matrices in (4.7), as discussed in Sect. 2.2.

However, we shall not use the FFT in this work, for two reasons. First, while the FFT significantly speeds up the matrix-matrix multiplication, from $O(L^3)$ to $O(L^2 \log L)$, this is not essential to the overall algorithm as it inevitably invokes an eigensolver (or an SVD), which requires $O(L(m+n)^2)$ operations. Indeed [35] designs the algorithm to fascilitate the use of the FFT, but again the saving is attenuated by the $O(Ln^2)$ SVD step.

The second, more fundamental, reason is stability. We shall see in Sect. 5 and through numerical experiments that diagonal scaling is crucial for stability. Unfortunately, using the FFT makes diagonal scaling inapplicable.

Pole exactly at sample point When a pole happens to exactly coincide with a sample point, $f(\gamma_i) = \infty$ and the eigenvalue problem breaks down due to infinity elements in the matrices. However, this should be a "happy" breakdown, rather than a difficulty. In this case we can simply take γ_i to be a computed pole, and work with the function $f := (z - \gamma_i)f$, taking $n := n - 1$. An alternative and equally valid approach is to take $f(\gamma_i) = \frac{1}{u}\text{median}_L|f(\gamma)|$, and proceed as usual.

4.3 Oversampling and least-squares fitting

As with previous algorithms, it is often recommended to take advantage of the oversampled values $f(\gamma_i)$ at more than $m + n + 1$ points γ_i, and perform a least-squares fitting. This is true especially in our context, where the degree-finding process in Algorithm 3.1 has oversampled f to find the type, and it is natural to try to reuse the computed quantities $f(\gamma_i)$.

Consider finding the poles of $f(z) = \frac{p(z)}{q(z)} \in \mathcal{R}_{m,n}$ with $L > m+n+1$ sample points $(\gamma_i)_{i=1}^{L}$. We form the matrices as in the previous Sect. (4.4) with $A_1 = \Gamma F V_n \in \mathbb{C}^{L\times n}$, $A_2 = V_{m+1} \in \mathbb{C}^{L\times(m+1)}$ and $B_1 = F V_n \in \mathbb{C}^{L\times n}$. We proceed as in (4.10) and apply projection, scaling, and orthogonalization $DA_2 = Q_{A_2}R_{A_2}$, $DB_1 = Q_{B_1}R_{B_1}$, to obtain matrices $\tilde{A} = (Q_{A_2}^{\perp})^* \Gamma Q_{B_1}$, $\tilde{B} = (Q_{A_2}^{\perp})^* Q_{B_1}$. as in (4.10), but these matrices are now nonsquare, of size $(L-m-1)\times n$: they have more rows than columns since $L > m+n+1$. Under the assumption that f has n poles, there exists a nonzero $x \in \mathbb{C}^n$ with $(\tilde{A} - \lambda\tilde{B})x = 0$ if and only if λ is one of the poles of f; this can be shown as in Proposition 1. Hence, in theory, we can compute the poles of f by solving the *rectangular* eigenvalue problem

$$\tilde{A}x = \lambda\tilde{B}x, \quad x \neq 0. \tag{4.11}$$

However, traditional methods for generalized eigenvalue problems such as the QZ algorithm [19, Ch. 7], [31] are not applicable to (4.11) since the pencil $\tilde{A} - \lambda\tilde{B}$ is rectangular.

To solve the rectangular eigenvalue problem (4.11), we use the recent algorithm by Ito and Murota [25]. The idea is to find perturbations $\Delta A, \Delta B$ with smallest $\|[\Delta\tilde{A}\ \Delta\tilde{B}]\|_{\mathrm{F}}$ so that the pencil $(\tilde{A}+\Delta\tilde{A}) - \lambda(\tilde{B}+\Delta\tilde{B})$ has n eigenpairs:

$$(\tilde{A}+\Delta\tilde{A})[x_1,\ldots,x_n] = (\tilde{B}+\Delta\tilde{B})[x_1,\ldots,x_n]\begin{bmatrix}\lambda_1 & & \\ & \ddots & \\ & & \lambda_n\end{bmatrix}. \tag{4.12}$$

The resulting algorithm computes the SVD $[\tilde{A}\ \tilde{B}] = U\Sigma\begin{bmatrix}W_{11}^* & W_{21}^* \\ W_{12}^* & W_{22}^*\end{bmatrix}$, then solves the *square* $n\times n$ generalized eigenvalue problem

$$W_{11}^* x = \lambda W_{21}^* x. \tag{4.13}$$

This corresponds to taking $\tilde{A}+\Delta\tilde{A} = U\Sigma\begin{bmatrix}W_{11}^* \\ O\end{bmatrix}$ and $\tilde{B}+\Delta\tilde{B} = U\Sigma\begin{bmatrix}W_{21}^* \\ O\end{bmatrix}$, hence $\|[\Delta\tilde{A}\Delta\tilde{B}]\|_{\mathrm{F}}^2 = \sum_{i=n+1}^{\min\{2n,L-m-1\}}\sigma_i([\tilde{A}\ \tilde{B}])^2$. See [25] for details.

4.4 Pseudocode

Summarizing, the following is the pseudocode for our polefinding algorithm `ratfun`.

By default, the sample points γ_i are the roots of unity $\gamma_j = \exp(\frac{2\pi \mathrm{i} j}{L})$ (once L is specified); other choices are allowed such as Chebyshev points on $[-1, 1]$. We justify the scaling in step 2 and the choice of the diagonal matrix D in Sect. 5.2.

Algorithm 4.1 `ratfun`: Polefinding and interpolation $f(\gamma_i) \approx \frac{p(\gamma_i)}{q(\gamma_i)}$; f, $\{\gamma_i\}_{i=1}^L$ are given.

```
1: Determine $m, n$ by Algorithm 3.1, if not given as inputs.
2: Scale $f := f/\mathrm{median}_L(|f(\gamma_i)|)$.
3: Define $A_1 = \mathrm{diag}(\gamma_i f_i)V_{n+1}$, $A_2 = V_{m+1}$ and $B_1 = \mathrm{diag}(f_i)V_{n+1}$.
4: Compute the diagonal scaling matrix $D = \mathrm{diag}(\frac{1}{\max(|f(\gamma_i)|,1)})$.
5: Compute full QR factorization $DA_2 = [Q_{A_2}\ Q_{A_2}^{\perp}]\begin{bmatrix} R_{A2} \\ 0 \end{bmatrix}$ and "thin" QR factorization $DB_1 = Q_{B_1}R_{B_1}$.
6: $\tilde{A} = (Q_{A_2}^{\perp})^* \Gamma Q_{B_1}$, $\tilde{B} = (Q_{A_2}^{\perp})^* Q_{B_1}$.
7: Compute the SVD
       $$[\tilde{A}\ \tilde{B}] = U\Sigma \begin{bmatrix} W_{11}^* & W_{21}^* \\ W_{12}^* & W_{22}^* \end{bmatrix}. \qquad (4.14)$$
8: Solve the square generalized eigenvalue problem
       $$W_{11}^* x = \lambda W_{21}^* x. \qquad (4.15)$$
   The computed eigenvalues $\lambda$ are approximants to the poles $\xi_i$ of $f$.
```

When the domain of interest is far from the origin, it is recommended that one work with a shifted function $f(z - z_0)$ so that the domain becomes near the origin (this affects the Vandermonde matrix, in particular its condition number).

4.4.1 Efficiency

The dominant costs of Algorithm 4.1 are in the QR factorizations, forming $Q_{A_2}^{\perp}$, computing the SVD (4.14) and solving the eigenproblem (4.15). These are all $O(L(m+n)^2)$ or less, using standard algorithms in numerical linear algebra. This is comparable in complexity with other approaches, as we summarized in Table 1.

4.5 Input/output parameters

Combined with the degree determination process Algorithm 3.1, `ratfun` lets us find poles and rational interpolants with the minimum input requirement: just the function f. Our algorithm `ratfun` (described in detail in Sect. 4) adapts to the specifications as necessary when the user inputs more information such as the location of the sample points and type of the rational approximants. Below we detail the process for three types of inputs:

1. Minimum input requirement: `poles = ratfun(f)`.
2. Function and sample points: `poles = ratfun(f,gam)`.
3. Function, sample points and degrees: `poles = ratfun(f,gam,m,n)`.

Minimum input requirement `poles = ratfun(f)` When the function f is the only input the algorithm first determines the numerical type of the rational approximant by Algorithm 3.1, then runs the polefinding algorithm to be described in Sect. 4. By default, we take the sample points to be roots of unity; Chebyshev points can be chosen by invoking `ratfun(f,'c')`.

Inputs are function and sample points `poles = ratfun(f,gam)` When the sample points are specified by `gam` the algorithm first runs the degree finder `typefind(f,tol,L)` with $L = \texttt{length(gam)}$, and gives a warning if the number of sample points L appears to be insufficient $L < \max\{M+n, m+N\}+1$, indicated by $\sigma_{m+n+2}(\tilde{C}) > tol$. Regardless, the algorithm proceeds with solving the generalized eigenvalue problem to obtain approximate poles and p, q with $\deg(p) + \deg(q) + 2 \le L$. We note that the backward errors $\|\Delta p\|_L$ and $\|\Delta q\|_L$ have magnitudes $O(\sigma_{m+n+2}(\tilde{C}))$, which is not necessarily $O(tol)$ in this case (see Sect. 5 for details on backward errors).

Full input: function, sample points and degrees `poles = ratfun(f,gam,m,n)` When the degrees are further specified the algorithm directly solves the (rectangular or square when $L = m+n+1$) generalized eigenvalue problem to obtain the poles and p, q.

4.5.1 Outputs

The full output information is `[poles,cp,cq,type,roots]=ratfun(f)`, in which `poles` are the computed poles, `cp,cq` are the vectors of coefficients c_p, c_q of the polynomials p, q in the monomial basis, `type` is a 2-dimensional vector $[\hat{m}, \hat{n}]$ of the computed type, and `roots` are the computed roots.

We next discuss how to compute the roots and finding c_p, c_q.

4.6 Computing the roots

One situation that Algorithm 4.1 did not deal with is when the roots of f are sought. We suggest two approaches for rootfinding, depending on whether poles are also sought or not.

Finding roots only First, when only the roots are of interest, we can invoke Algorithm 4.1 to find the poles of $1/f$. Alternatively, the roots can be computed from f by defining $\tilde{p}(z) = \frac{p(z)}{z-r_k}$ and starting from the guiding equation [recall (4.2)]

$$f(\gamma_i)q(\gamma_i) = z\tilde{p}(\gamma_i) - r_k\tilde{p}(\gamma_i), \quad i = 1, \ldots, m+n, \tag{4.16}$$

which, as before, can be rewritten as a generalized eigenvalue problem with $\lambda := r_k$. For brevity we omit the details, as the formulation is analogous to that for (4.4) and (4.11).

Finding poles and roots When both the poles and roots are required, we suggest the following. First compute the poles as in Algorithm 4.1. Then we find the roots by solving for λ the equation

$$f(\gamma_i)q(\gamma_i) = (\gamma_i - \lambda)\tilde{p}(\gamma_i), \quad \tilde{p}(z) = \frac{p(z)}{z - \lambda}. \tag{4.17}$$

Here $\tilde{p}(z)$ is the same as above, and we form $q(\gamma_i)$ from the expression $q(z) = \prod_{i=1}^{n}(z - \hat{\xi}_i)$ using the poles $\hat{\xi}_i$ that have previously been computed. Equation (4.17) can be rearranged to $\gamma_i \tilde{p}(\gamma_i) - f(\gamma_i)q(\gamma_i) = \lambda \tilde{p}(\gamma_i)$, which we write in matrix form as

$$\begin{bmatrix} F\text{diag}(q(\gamma_i)) & \Gamma V \end{bmatrix} \begin{bmatrix} 1 \\ -c_{\tilde{p}} \end{bmatrix} = \lambda \begin{bmatrix} 0 & V \end{bmatrix} \begin{bmatrix} 1 \\ -c_{\tilde{p}} \end{bmatrix}. \tag{4.18}$$

This is again a rectangular generalized eigenvalue problem. This has one irrelevant eigenvalue at infinity, and the problem can again be solved via an SVD. Since the matrices involved are of smaller size than (4.4) and (4.11), this process is cheaper than finding the poles of $1/f$.

4.7 Finding c_p, c_q

To find the coefficient vectors c_p and c_q, we can take advantage of the eigenvector structure (4.6) to extract c_p from any eigenvector, along with $c_{\tilde{q}}$, from which we obtain c_q via (4.1). Note that to do this we need to give up the QR factorization in step 4 of Algorithm 4.1. Equally effective and stable is to invoke the scaled naive method (2.8), which gives c_p, c_q directly (our current code adopts this approach). A word of caution is that eigenvectors are sensitive to perturbation if (but not only if) the corresponding eigenvalues are nearly multiple.

We note that there are many other ways of representing a rational function $f = p/q$. Since `ratfun` can compute the poles and roots as described above, one effective representation is to take

$$f = c\frac{\prod_{i=1}^{m}(z - \hat{r}_i)}{\prod_{i=1}^{n}(z - \hat{\xi}_i)}, \tag{4.19}$$

in which we store the constant c and the roots $\hat{r}_i$ and poles $\hat{\xi}_i$.

4.8 Mathematical equivalence with previous algorithms: interpolation-based and Hankel eigenproblem

Here we briefly discuss the connection between our eigenvalue problem and existing ones. We shall show that the eigenproblem (4.7), when $L = m + n + 1$, is equivalent in exact arithmetic to the generalized eigenvalue problem of Hankel matrices derived in [29,40], which are in turn equivalent to Chebfun's `ratinterp` as shown in [3]. Essentially, both our algorithm and `ratinterp` find the roots of q such that $\frac{p}{q}$ interpolates f at the sample points.

We shall show that the eigenvalues and right eigenvectors of (4.7) and those of the Hankel matrix pencil are the same. Before proving this claim, we briefly review the Hankel eigenproblem approach, which originates in work of Delves and Lyness [15] and Kravanja et al. [27,28], see also [3]. In this algorithm, one computes the discretized moments

$$s_j := \frac{1}{2\pi \mathrm{i}} \oint_{|z|=1} z^j f(z) \mathrm{d}z, \quad j = 0, \ldots, 2n-1,$$

and then solves the generalized eigenvalue problem with Hankel matrices

$$\begin{bmatrix} s_1 & & s_n \\ \vdots & \iddots & \vdots \\ s_n & & s_{2n-1} \end{bmatrix} x = \lambda \begin{bmatrix} s_0 & & s_{n-1} \\ \vdots & \iddots & \vdots \\ s_{n-1} & & s_{2n-2} \end{bmatrix} x. \tag{4.20}$$

We write this as $H_1 x = \lambda H_0 x$ for simplicity. The pencil $H_1 - \lambda H_0$ can be written using a contour integral as

$$H_1 - \lambda H_0 = \oint_{|z|=1} (z-\lambda) f(z) \begin{bmatrix} 1 \\ \vdots \\ z^{n-1} \end{bmatrix} [1 \ \cdots \ z^{n-1}] \mathrm{d}z. \tag{4.21}$$

If f is meromorphic in the unit disk $\{z \in \mathbb{C} \mid |z| \le 1\}$ and has n poles $\xi_1, \ldots, \xi_n \in \{z \in \mathbb{C} \mid |z| < 1\}$, then the poles are eigenvalues of $H_1 x = \lambda H_0 x$. Indeed, defining $\tilde{q} = \prod_{l \neq k, 1 \le l \le n} (z - \xi_l)$ and letting $c_{\tilde{q}}$ be its coefficient vector as in (4.3) we obtain

$$(H_1 - \xi_k H_0) c_{\tilde{q}} = \oint_{|z|=1} (z - \xi_k) f(z) \tilde{q}(z) \begin{bmatrix} 1 \\ \vdots \\ z^{n-1} \end{bmatrix} \mathrm{d}z = 0, \tag{4.22}$$

since $(z - \xi_k) f(z) \tilde{q}(z) = f(z) \prod_{k=1}^{n} (z - \xi_k)$ is analytic in the unit disk.

The contour integral (4.22) needs to be discretized in a practical computation. If we use the standard trapezoidal rule evaluating at roots of unity $\gamma_j = \exp(2\pi \mathrm{i} j/L)$ for $j = 1, \ldots, L$ to approximate $H_1 - \lambda H_0$, the computed pencil $\hat{H}_1 - \lambda \hat{H}_0$ becomes

$$\begin{aligned} &\hat{H}_1 - \lambda \hat{H}_0 \\ &= \sum_{j=1}^{L} \frac{2\pi \mathrm{i} \gamma_j}{L} (\gamma_j - \lambda) f(\gamma_j) \begin{bmatrix} 1 \\ \vdots \\ \gamma_j^{n-1} \end{bmatrix} [1 \ \cdots \ \gamma_j^{n-1}] \\ &= \frac{2\pi \mathrm{i}}{L} \begin{bmatrix} \gamma_1 & \gamma_2 & \cdots & \gamma_L \\ \vdots & \vdots & & \vdots \\ \gamma_1^n & \gamma_2^n & \cdots & \gamma_L^n \end{bmatrix} \begin{bmatrix} (\gamma_1 - \lambda) f(\gamma_1) & & & \\ & \ddots & & \\ & & \ddots & \\ & & & (\gamma_L - \lambda) f(\gamma_L) \end{bmatrix} \begin{bmatrix} 1 & \cdots & \gamma_1^{n-1} \\ 1 & \cdots & \gamma_2^{n-1} \\ \vdots & & \vdots \\ 1 & \cdots & \gamma_L^{n-1} \end{bmatrix} \\ &= V_n^\top \Gamma^2 F V_n - \lambda V_n^\top \Gamma F V_n, = (V_n^\top \Gamma F)(\Gamma - \lambda I) V_n, \end{aligned} \tag{4.23}$$

where $F = \mathrm{diag}(f(\gamma_1), \ldots, f(\gamma_L))$ and $\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_L)$ as before. Hence if f is a rational function $f = \frac{p}{q}$, we have

$$\begin{aligned}(\hat{H}_1 - \xi_i \hat{H}_0)c_{\tilde{q}} &= (V_n^\top \Gamma)F(\Gamma - \xi_i I)[\tilde{q}(\gamma_1), \ldots, \tilde{q}(\gamma_L)]^\top \\ &= V_n^\top \Gamma [p(\gamma_1), \ldots, p(\gamma_L)]^\top \\ &= V_n^\top [\gamma_1 p(\gamma_1), \ldots, \gamma_L p(\gamma_L)]^\top.\end{aligned}$$

The ith element of the final vector is $\sum_{j=1}^{L} \gamma_j^{i+1} p(\gamma_i)$ for $i = 1, \ldots, n$, which is equal to the evaluation of $\frac{1}{2\pi \mathrm{i}} \oint_{|z|=1}^{L} z^i p(z)$. Now the L-point trapezoidal rule is exact if the integrand is polynomial of degree $L-1$ or below [49, Cor. 2.3]. Therefore, if $L \geq m+n+1$ then $(\hat{H}_1 - \xi_i \hat{H}_0)c_{\tilde{q}} = 0$. Thus also for the discretized pencil $\hat{H}_1 - \xi_i \hat{H}_0$, $c_{\tilde{q}}$ is again an eigenvector if $f = \frac{p}{q}$ with $p \in \mathcal{P}_m, q \in \mathcal{P}_n$ with $L \geq m+n+1$.

This shows that the eigenproblems $\hat{H}_1 x = \lambda \hat{H}_0 x$ and $(A_2^\perp)^* A_1 x = \lambda (A_2^\perp)^* B_1 x$ in (4.9) have the same eigenvalues and eigenvectors, thus are equivalent, i.e., there exists a nonsingular matrix W such that $W\hat{H}_1 = (A_2^\perp)^* A_1$ and $W\hat{H}_0 = (A_2^\perp)^* B_1$.

Despite the mathematical equivalence, we reiterate that the numerical behavior of the algorithms is vastly different. Crucially, the left-multiplication by $V_n^\top$ in (4.23) mixes up the magnitudes of $f(\gamma_i)$, resulting in the instability due to near-pole sampling. This will be made precise in the next section.

5 Numerical stability

A crucial aspect of any numerical algorithm is stability [23]. It is common, and often inevitable for problems that are potentially ill-conditioned, to investigate backward stability (as opposed to analyzing the forward error in the outcome itself), in which we ask whether a computed output is guaranteed to be the exact solution of a slightly perturbed input.

The great success of polynomial interpolation of a continuous function f at roots of unity (for approximation in the unit disk) or Chebyshev points (on an interval $[-1, 1]$) is due to its combined efficiency and stability: a degree-n polynomial interpolation can be done in $O(n \log n)$ operations employing the Chebyshev polynomials and FFT [6]. Moreover, since the FFT matrix has condition number 1, the process is numerically stable, and we obtain an interpolant $\hat{p}$ satisfying

$$f(\gamma_i) = \hat{p}(\gamma_i) + O(u\|f\|_L) \tag{5.1}$$

at every sample point γ_i; this holds regardless of f. Suppose further the interpolation is successful (with smooth f, good points γ_i and basis $\phi_i(z)$) in that $\|f - \hat{p}\|_\infty / \|f\|_\infty = O(u)$, where $\|f\|_\infty = \max_{z \in \Omega} |f(z)|$ for a domain Ω. Then with a stable rootfinding algorithm for $\hat{p}$, one obtains stability in the computed roots: $p(\hat{r}_i) = O(u\|f\|_\infty)$. This shows $\hat{r}_i$ are the exact roots of a slightly perturbed input f. Rootfinding algorithms with proven stability include the companion [51] (for monomials) and colleague linearizations [32] (for Chebyshev).

For rational interpolation and polefinding, to our knowledge, stability in the context of polefinding and rational interpolation has been rarely discussed; [35], which connects the inaccuracies with the presence of ill-conditioned matrices, is one of the few, but their argument does not treat the backward stability of the rational interpolants $\frac{\hat{p}}{\hat{q}}$. Here we attempt to make a step forward and analyze backward stability for rational interpolation algorithms.

First we need to elucidate our goal. The presence of poles complicates the situation because, for example $\|f-\frac{\hat{p}}{\hat{q}}\|_\infty$ is infinity unless we compute the poles exactly, and this is true even for the linearized version $\|f\hat{q}-\hat{p}\|_\infty$. For this reason, sometimes rational interpolation is thought to be inherently ill-posed for a stable computation.

There is a natural workaround here: we allow for perturbation in both the numerator and denominator polynomials $\hat{p}$ and $\hat{q}$. We then analyze whether the rational interpolation is satisfied with small backward errors, that is,

$$f(\gamma_i)=\frac{\hat{p}(\gamma_i)+\Delta p(\gamma_i)}{\hat{q}(\gamma_i)+\Delta q(\gamma_i)},\quad \frac{\|\Delta p\|_L}{\|\hat{p}\|_L}=O(u),\quad \frac{\|\Delta q\|_L}{\|\hat{q}\|_L}=O(u) \tag{5.2}$$

for $i=1,\dots,L,$. As before, we work with the linearized formulation.

Definition 1 Let f be a meromorphic function. Given sample points $\{\gamma_i\}_{i=1}^L$ and computed polynomials $\hat{p}$, $\hat{q}$, we say that $\hat{p}/\hat{q}$ is a *stable rational interpolant* of f if there exist functions $\Delta q, \Delta p : \{\gamma_i\}_{i=1}^L \to \mathbb{C}$ such that

$$\begin{aligned} &f(\gamma_i)(\hat{q}(\gamma_i)+\Delta q(\gamma_i))-(\hat{p}(\gamma_i)+\Delta p(\gamma_i))=0,\\ &\frac{\|\Delta p\|_L}{\|\hat{p}\|_L}=O(u),\quad \frac{\|\Delta q\|_L}{\|\hat{q}\|_L}=O(u). \end{aligned} \tag{5.3}$$

We note that the requirement here is a rather weak condition: for example, it does not require that $\hat{p}, \hat{q}$ are close to the correct p, q when $f=p/q$. Nonetheless, we shall see that many previous algorithms fail to satisfy them. We now give a necessary and sufficient condition for stability that is easy to work with.

Lemma 1

$$|f(\gamma_i)\hat{q}(\gamma_i)-\hat{p}(\gamma_i)|=O(u)\max(|f(\gamma_i)|\|\hat{q}\|_L,\|\hat{p}\|_L) \tag{5.4}$$

is a necessary and sufficient condition for $\hat{p}/\hat{q}$ to be a stable rational interpolant at γ_i satisfying (5.3)*, for $i=1,\dots,L$.*

(Proof) Suppose (5.4) is satisfied. Then, defining Δp and Δq by

$$(\Delta p(\gamma_i),\Delta q(\gamma_i))=\begin{cases}(f(\gamma_i)\hat{q}(\gamma_i)-\hat{p}(\gamma_i),\quad 0) & \text{if } |f(\gamma_i)|\|\hat{q}\|_L\le\|\hat{p}\|_L\\ (0,\quad -\frac{f(\gamma_i)\hat{q}(\gamma_i)-\hat{p}(\gamma_i)}{f(\gamma_i)}) & \text{if } |f(\gamma_i)|\|\hat{q}\|_L>\|\hat{p}\|_L,\end{cases} \tag{5.5}$$

we obtain (5.3). Conversely, if Δq and Δp satisfy (5.3), then we have

$$|f(\gamma_i)\hat{q}(\gamma_i)-\hat{p}(\gamma_i)|=|f(\gamma_i)\Delta q(\gamma_i)-\Delta p(\gamma_i)|=O(u)\max(|f(\gamma_i)|\|\hat{q}\|_L,\|\hat{p}\|_L). \tag{5.6}$$

This proves the claim. □

Below we analyze the stability of algorithms based on Lemma 1. In Sects. 5.1 and 5.2, to avoid the jarring complications due to the ill-conditioning of the Vandermonde matrix, we discuss the case where the sample points are the roots of unity and the polynomial basis is the monomials $\phi_i(z) = z^i$. Essentially the same argument carries over to other sets of sample points employed with an appropriate polynomial basis $\{\phi_i(z)\}$, such as Chebyshev-points sampling employing the Chebyshev polynomial basis.

5.1 Instability of previous algorithms

Here we illustrate with the example of Chebfun's `ratinterp` that previous algorithms can be numerically unstable, i.e., they do not necessarily satisfy (5.4) in Lemma 1. Recall that `ratinterp` computes c_q in (2.11) as the null vector of $\widetilde{V}^* F V_{N+1}$.

Let us explain the numerical issue here. Let $\hat{c}_q$ be the computed null vector. Consider the Eq. (2.10) left-multiplied by the Vandermonde matrix V, which is unitary times $\sqrt{L}$. Taking into account the numerical errors, the equation can be written as

$$FV\begin{bmatrix}\hat{c}_q\\0\end{bmatrix} = V\begin{bmatrix}\hat{c}_p\\ \epsilon_{L-(m+1)}\end{bmatrix}, \tag{5.7}$$

which we rewrite using $\epsilon_L = V\begin{bmatrix}0\\ \epsilon_{L-(m+1)}\end{bmatrix}$ as

$$FV\begin{bmatrix}\hat{c}_q\\0\end{bmatrix} - V\begin{bmatrix}\hat{c}_p\\0\end{bmatrix} = \epsilon_L. \tag{5.8}$$

The vectors $\epsilon_{L-(m+1)}$ and ϵ_L are zero when $\hat{c}_q$ is equal to the exact c_q, but $\epsilon_{L-(m+1)}, \epsilon_L \neq 0$ due to numerical errors. Indeed, we see that the ith element of ϵ_L is $f(\gamma_i)\hat{q}(\gamma_i) - \hat{p}(\gamma_i)$, which is precisely the linearized interpolation residual in (5.4).

Now, the computed null vector $\hat{c}_q$ of the matrix $\widetilde{V}^* F V_{N+1}$ in (2.11) obtained by a stable algorithm such as the SVD generally satisfies the normwise condition

$$(L\|\epsilon_L\|_2 =)\ \|\widetilde{V}^* F V_{N+1}\hat{c}_q\|_2 = O(u\|\widetilde{V}^* F V_{N+1}\|_2). \tag{5.9}$$

Now since $\|V\|_2 = \|V^*\|_2 = \sqrt{L} = O(1)$, we have $\|\widetilde{V}^* F V_{N+1}\|_2 = O(\max_i |f(\gamma_i)|)$. Thus $\|\epsilon_L\|_2 = O(u \max_i |f(\gamma_i)|)$, which indicates that if $\max_i |f(\gamma_i)| \gg |f(\gamma_j)|$ for some j, then the interpolation residual for the jth equation is (for a constant $c_i = O(1)$)

$$f(\gamma_j)\hat{q}(\gamma_j) - \hat{p}(\gamma_j) = c_i u \max_i |f(\gamma_i)|\|\hat{q}\|_L \gg c_i u \max(|f(\gamma_j)|\|\hat{q}\|_L, \|\hat{p}\|_L),$$

which violates the condition (5.4) for stability.

Although we do not present the details, such instability is present in most algorithms, including the *unscaled* naive method and RKFIT (with default weight $\tilde{D}$).

5.2 Diagonal scaling and stability of `ratfun` and scaled naive method

Let us reconsider the eigenvalue problem (4.4) from a similar viewpoint, and we shall show that our approach of solving (4.4) employing diagonal scaling is immune to the instability just discussed, and `ratfun` gives a stable rational interpolation.

For simplicity we rewrite the eigenvalue problem (4.4) with diagonal scaling.[3] $D\begin{bmatrix}A_1 & A_2\end{bmatrix}x = \lambda\begin{bmatrix}DB_1 & O\end{bmatrix}x$ as $DAx = \lambda DBx$. By the backward stability of the standard QZ algorithm, each computed eigenpair $(\hat{\xi}_i, \hat{x})$ satisfies

$$(DA+\Delta A)\hat{x} = \hat{\xi}_i(DB+\Delta B)\hat{x}, \quad \|\Delta A\|_2 \le \epsilon\|DA\|_2, \|\Delta B\|_2 \le \epsilon\|DB\|_2, \tag{5.10}$$

in which ϵ denotes a constant of magnitude $O(u)$.

To establish stability we need two preparations. First, we use an appropriate scaling of f. We can clearly scale $f \leftarrow \frac{1}{\kappa}f$ for any $\kappa > 0$ without changing the poles and roots, and the analysis below will show that a good choice is one such that $\|c_p\|_2 \approx \|c_q\|_2$. To be precise, it suffices to have

$$\frac{\|c_p\|_2}{\|c_q\|_2} = \Theta(1), \tag{5.11}$$

which means $\frac{\|c_p\|_2}{\|c_q\|_2} = O(1)$ and $\frac{\|c_q\|_2}{\|c_p\|_2} = O(1)$. This means we expect $f = \Theta(1)$ holds at most of the sample points. In practice, we achieve (5.11) by sampling at sufficiently many points and taking κ to be the median value $\mathrm{median}_L|f(\gamma)|$; this is adopted in the pseudocode of `ratfun`, Step 2 of Algorithm 4.1.

Second, as mentioned before, we choose the diagonal scaling matrix D as in (2.4), so that (since we scale f s.t. $\mathrm{median}_L|f(\gamma)| = 1$) the jth diagonal d_j is

$$d_j = \frac{1}{\max(|f(\gamma_i)|, 1)}, \quad j = 1, \dots, L. \tag{5.12}$$

We are now ready to state our main stability result.

Theorem 1 *Let A, B be as defined in (4.4) with $L = m+n+1$, where $\gamma_j = \exp(\frac{2\pi \mathrm{i} j}{L})$ and $\phi_i(z) = z^i$. Let $D = diag(d_j)$ be as in (5.12), and let $(\hat{\xi}_k, \hat{x})$ with $|\hat{\xi}_k| = O(1)$ be a computed eigenpair such that (5.10) holds. Partition $\hat{x} = \begin{bmatrix}\hat{c}_{\tilde{q}} \\ -\hat{c}_p\end{bmatrix}$, where $\hat{c}_{\tilde{q}} \in \mathbb{C}^n$. Defining $\hat{p} = \sum_{j=0}^{m}\hat{c}_{p,j}z^j$, $\tilde{q} = \sum_{j=0}^{n-1}\hat{c}_{\tilde{q},j}z^j$ and $\hat{q} = (z-\hat{\xi}_k)\tilde{q}$ with coefficient vector $\hat{c}_q$, suppose that $\|\hat{c}_q\|_2/\|\hat{c}_p\|_2 = \Theta(1)$. Then $\hat{p}/\hat{q}$ is a stable rational interpolant of f, that is, (5.4) is satisfied.*

[3] For simplicity, we mainly analyze the scaled version of (4.4), without employing the projection (4.7) and QR factorization.

(Proof) By (5.10) we have

$$\|DA\hat{x} - \hat{\xi}_k DB\hat{x}\|_2 = O(u(\|DA\|_2 + \hat{\xi}_k\|DB\|_2)\|\hat{x}\|_2) = O(u\|\hat{x}\|_2), \tag{5.13}$$

where we used the fact that $\|DA\|_2$, $\|DB\|_2$ and $|\xi_k|$ are all $O(1)$. Now recalling (4.2), the ith element of $DA\hat{x} - \hat{\xi}_k DB\hat{x}$ is

$$\begin{aligned}(DA\hat{x} - \hat{\xi}_k DB\hat{x})_i &= d_i(\gamma_i f(\gamma_i)\tilde{q}(\gamma_i) - \hat{p}(\gamma_i)) - d_i\xi_k f(\gamma_i)\tilde{q}(\gamma_i)\\ &= d_i(f(\gamma_i)(\gamma_i - \hat{\xi}_k)\tilde{q}(\gamma_i) - \hat{p}(\gamma_i))\\ &= d_i(f(\gamma_i)\hat{q}(\gamma_i) - \hat{p}(\gamma_i)).\end{aligned}$$

This represents a *scaled* interpolation error, which is $O(u\|\hat{x}\|_2)$ by (5.13). Since γ_i are roots of unity we have $\|\hat{p}\|_L = \sqrt{L}\|\hat{c}_p\|_2$ and $\|\hat{q}\|_L = \sqrt{L}\|\hat{c}_q\|_2$, so in view of Lemma 1, it suffices to show that

$$\frac{1}{d_i}\|\hat{x}\|_2 = O(\max(|f(\gamma_i)|\|\hat{c}_q\|_2, \|\hat{c}_p\|_2)). \tag{5.14}$$

Now since $\|\hat{x}\|_2 = \|\left[\begin{smallmatrix}\hat{c}_{\tilde{q}}\\ \hat{c}_p\end{smallmatrix}\right]\|$, using the assumption $\|\hat{c}_q\|_2/\|\hat{c}_p\|_2 = \Theta(1)$ and the fact $\|\hat{c}_q\|_2/\|\hat{c}_{\tilde{q}}\|_2 = \Theta(1)$, which follows from $|\hat{\xi}_k| = O(1)$, we have $\|\hat{x}\|_2/\|\hat{c}_p\|_2 = O(1)$ and $\|\hat{x}\|/\|\hat{c}_q\|_L = O(1)$. Using these in Eq. (5.14) divided by $\|\hat{x}\|_2$, we see that it suffices to establish $\frac{1}{d_i} = O(\max(|f(\gamma_i)|, 1))$, which indeed holds due to the choice of diagonal scaling (5.12).

Since the above argument is valid for every $i = 1, \dots, L$, we conclude from Lemma 1 that $\hat{p}/\hat{q}$ is a stable rational interpolant of f. □

We emphasize the crucial role that diagonal scaling plays in the stability analysis. We also note that the scaling such that $f \leftarrow cf$ is actually not necessary for the reduced eigenproblem (4.7) (without the diagonal scaling D), which is invariant under the scaling $f \leftarrow cf$.

Stability of scaled naive method The scaled naive method can also be proven to be stable. In this case the analysis is even simpler as the jth row of $DC\left[\begin{smallmatrix}c_q\\ -c_p\end{smallmatrix}\right]$, where C is as in (2.6), represents

$$d_j\left(\begin{bmatrix}FV_{n+1} & V_{m+1}\end{bmatrix}\begin{bmatrix}c_q\\ -c_p\end{bmatrix}\right)_j = d_j(f(\gamma_j)q(\gamma_j) - p(\gamma_j)). \tag{5.15}$$

That is, the residual of each row is exactly the scaled interpolation error. Thus a null vector $\left[\begin{smallmatrix}c_q\\ -c_p\end{smallmatrix}\right]$ computed in a stable manner under the same assumptions as above [(5.10) and (5.11)] is automatically a stable rational interpolant.

However, for finding the poles, the additional process of finding the roots of q is necessary, and this can be a cause for further numerical instability. We discuss this further in Sect. 5.3.

Barycentric formula Finally, we mention the rational interpolation based on the barycentric formula [9–11,41]

$$r(z) = \frac{\sum_{k=0}^{n} \frac{w_k}{z-\gamma_k} f_k}{\sum_{k=0}^{n} \frac{w_k}{z-\gamma_k}}, \tag{5.16}$$

where $w = [w_1, \ldots, w_n]$ are called the barycentric weights. For a general w (e.g. for randomly chosen w) the rational function $r(z)$ in (5.16) is of type (n, n). However, by choosing appropriate weights w one obtains an interpolant of desired type; Berrut and Mittelmann [10] show how such w can be found by a null vector of a matrix related to $\left[FV_{n+1}\ V_{m+1}\right]$ as in (2.6). Antoulas and Anderson [1] introduce an algorithm for computing w to interpolate $r(\gamma_i) = f(\gamma_i)$, where $(\gamma_k)_{i=1}^{n}$ are taken to be half of the sample points and hence interpolation is achieved at $2n$ points. The recent AAA algorithm [33] chooses the points $(\gamma_k)_{i=1}^{n}$ in a greedy manner reduce the linearized error in the rational approximant.

As noted in [10], at the sample points γ_k the barycentric formula (5.16) essentially gives an *exact* interpolation function, in that $r(\gamma_k) = f(\gamma_k)$ at all γ_k (this holds regardless of the choice of w as long as $w_k \neq 0$). However, this is due to the representation of the rational function; finding the poles and obtaining p, q from (5.16) would induce further numerical errors. Below we focus our attention on algorithms that work with the coefficients of the rational interpolant.

5.3 Accuracy of polefinder and effect of orthogonalization

For rational interpolation, we have described and identified two stable methods: `ratfun` and the scaled naive method.

Let us now turn to polefinding, and focus on the accuracy of the computed poles. `ratfun` finds the poles while simultaneously computing the rational approximant. By contrast, in the scaled naive method (or practically any other existing method for rational approximation) we first find the denominator polynomial q, then compute its roots. Intuitively this two-step approach should be more susceptible to numerical instability, and here we illustrate that this is indeed the case.

We compare two algorithms: scaled naive and `ratfun`. In `ratfun`, the QR factorization $DB_1 = Q_{B_1} R_{B_1}$ in step 3 of Algorithm 4.1 is implicitly performing a change of basis for the polynomial q, so that discrete weighted orthogonality is achieved in the new basis. This has the possible disadvantage that the computed eigenvector in (4.15) contains the coefficients in the changed basis. However, crucially, for polefinding, this is not an issue at all, because the eigenvalues are unaffected. The change-of-basis is rather a benefit, because in the new basis the matrices A_1, B_1 are well-conditioned, which reduces numerical errors.

By contrast, this change-of-basis cannot be done for the naive method, because it requires the coefficients of q in a polynomial basis that is easy to work with for computing the roots.

Numerical example We illustrate the above discussion with an example. Let f be a rational function of the form (3.5) with $N = 20$ poles in equispaced points on

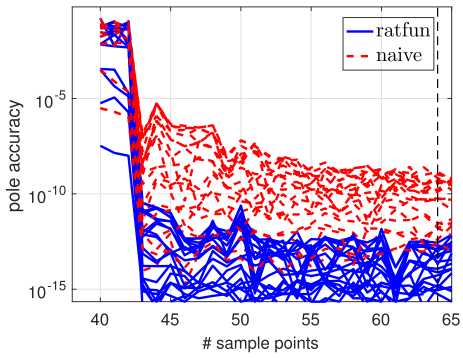

Fig. 4 Error of 20 computed poles for `ratfun` and the naive method using the monomial basis. `ratfun` implicitly and automatically uses the appropriate polynomial basis to obtain accurate poles

$[-1+\delta, 1-\delta]$ with $\delta = 10^{-3}$. We use the monomial basis but sample at Chebyshev points, whose number we vary. Therefore we are employing a "wrong" polynomial basis for the sample points; the "correct" one is the Chebyshev polynomials.

Figure 4 shows the result with the two algorithms, showing the errors in the computed poles $\min_j |\xi_i - \hat{\xi}_j|$ for $i = 1, \ldots, n$, as the number of sample points is varied. The diagonal scaling (5.12) is used for both algorithms; without this, the accuracy is significantly worse than in the figures.

`ratfun` clearly gives significantly more accurate results. The inaccuracy of the naive method is mainly due to the wrong choice of basis used to represent q. For example, by choosing the "correct" Chebyshev basis, the red plots become only slightly worse than `ratfun`. Indeed it is known that when looking for real roots of a polynomial, one is often advised to use Chebyshev polynomials instead of monomials.

The point here is that `ratfun` *automatically* finds an appropriate basis for the particular problem given: If a function f and a set of sample points are given, the QR factorization finds the appropriate basis. Indeed, if the QR factorization is not used in `ratfun`, the accuracy deteriorates significantly.

6 Numerical experiments

We present further numerical experiments to illustrate the behavior of `ratfun`. We compare

1. `ratfun`: Algorithm 4.1,
2. The scaled naive method (2.8) with diagonal scaling (5.12), shown as naive,
3. Chebfun's `ratinterp` command, shown as chebfun,
4. RKFIT [7,8] with diagonal inputs $F = \mathrm{diag}(f(\gamma_i))$, $A = \mathrm{diag}(\gamma_i)$ and $v = [1, \ldots, 1]^\top$, with the default choice $\tilde{D} = I$ and maximum number of iterations set to 10 (increasing this made no noticeable difference).

All the experiments were conducted in MATLAB R2014a using IEEE double precision arithmetic with $u \approx 1.1 \times 10^{-16}$, on a desktop machine with an Intel Core i7 Processor with four cores, and f16GB RAM.

For "easy" problems like those in Sect. 3.2, all the algorithms compute the poles and approximants reliably. Below we thus focus on more challenging problems.

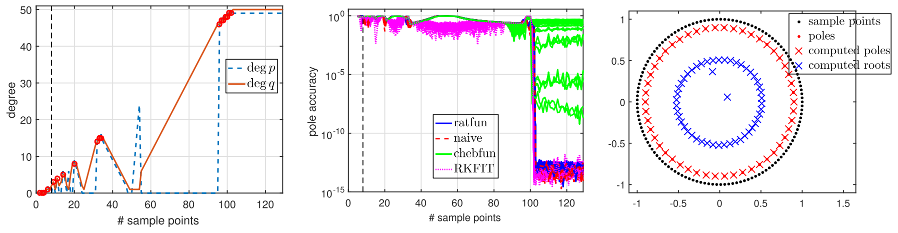

Fig. 5 $f(z) = \sum_{i=1}^{N} \frac{1}{z-\xi_i}$, with 50 poles inside the unit disk. Left: numerical degrees of the rational approximants. Black dashed line indicates the number of sampled points L in the degree determination process Algorithm 3.1; here $L = 8$. `ratfun(f)` returns the incorrect type $(0, 2)$; see comment in the text. Middle: error of computed poles $\min_j |\xi_i - \hat{\xi}_j|$ with `ratfun(f,gam)` sampling 128 points. Right: sample points, poles and roots

High-degree examples We consider a moderately high-degree example, where we take f as in (3.5) with $N = 50$. The results are in Fig. 5.

With a sufficient number of sample points $L \geq 103$, `ratfun` finds the type of the rational approximant and computes the poles and roots stably. Here and below, the roots are computed simply by finding the poles of $1/f$ by `ratfun`; the other processes described in Sect. 4.6 had similar performances.

It is worth observing that most computed roots turned out to lie on a circle of radius about 0.5. This may look bizarre at first sight, as one easily sees that the only zero of f is at $z = 0$. This can be explained by eigenvalue perturbation theory: the zero of f has multiplicity $N - 1 = 49$, so the eigenvalue problem for computing it attempts to find the eigenvalue of algebraic multiplicity 49. Now it is well known [43, Ch. 5] that the eigenvalues of algebraic multiplicity k and geometric multiplicity 1, which are essentially eigenvalues of a $k \times k$ Jordan block, can be perturbed by $O(\epsilon^{1/k})$ by a perturbation in the matrix of norm $O(\epsilon)$. The QZ algorithm computed the roots in a backward stable manner, but the small perturbation is enough to perturb them by $u^{\frac{1}{49}} \approx 0.4725$. The reason the roots appear to lie systematically on a circle is that the eigenvalues of a Jordan block are extremely sensitive to perturbation in the bottom-left element, but much less so in other positions.

Note in the left figure that with insufficient sample points $L < 100$ the type finder outputs an incorrect output. In view of Lemma 4, this happens when the number of sample points L was less than the necessary $\max\{M + n, m + N\} + 1$, but the function f and sample points γ_i happened to (e.g. by symmetry) make the matrix C rank-deficient, and so at γ_i the function behaved as if it is a lower-type rational function. The same was observed in Fig. 1, and the problem is pronounced here. Indeed the degree determination Algorithm 3.1 indicated a numerical degree of $(0, 2)$ after sampling initially at the eigth roots of unity. We have not overcome this issue completely; indeed such difficulty is present even in the polynomial case [48]; for example when a highly oscillatory function f happened to be 0 at all the initial sample points. Perhaps some further insurance policy is needed to ensure that the type obtained by Algorithm 3.1 is appropriate, such as sampling f at a few more random points [2]. One could also try `typefind(f,tol,L)` for neighboring values of L and accept only if they are the same. These remedies, while effective, cannot be proven to be fool-proof.

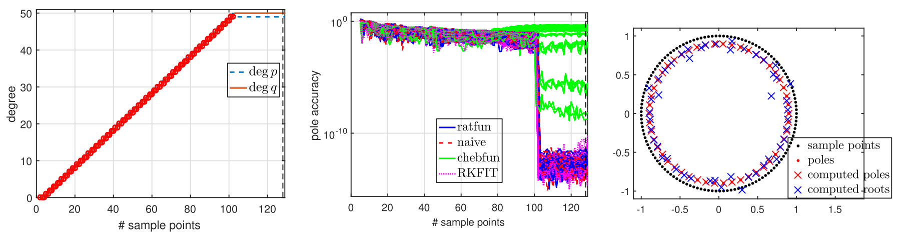

Fig. 6 $f(z) = \sum_{i=1}^{N} \frac{r_i}{z-\xi_i}$ with random residues r_i. Left: numerical degrees of the rational approximants. `ratfun(f)` samples at 2^7 points and returns the correct type (49, 50). Middle: error of computed poles $\min_j |\xi_i - \hat{\xi}_j|$. Right: sample points, poles and roots

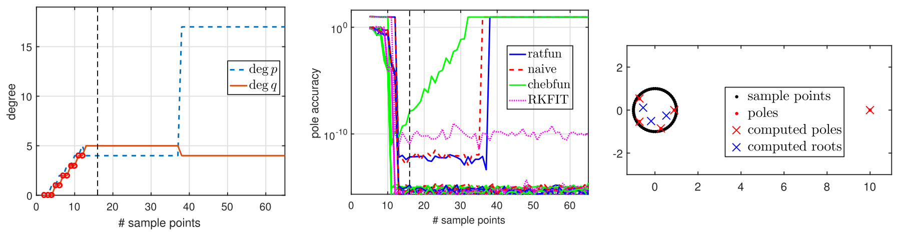

Fig. 7 f with a pole far outside the unit disk. Left: numerical degrees of the rational approximants. `ratfun(f)` samples at 2^4 points and returns the type (4, 5). Middle: error of computed poles $\min_j |\xi_i - \hat{\xi}_j|$. The pole that eventually gets lost by `ratfun` and naive corresponds to $\xi_1 = 10$, the pole far from the sample points. Right: sample points, poles and roots

Nonetheless, this is a rather contrived example with high symmetry, which generically would not happen. For example, if we take the residues of each term in (3.6) to be random numbers, we obtain Fig. 6, for which an appropriate type is chosen. In both cases, once a sufficient number of sampled points is taken, `ratfun` finds the correct poles and rational intepolant.

When f has poles far away from sample points Another possible difficulty is when f has a pole far away from the sample points.

To examine the behavior of our algorithm in such cases we take f as in (3.5) of type (4, 5), but we now set one pole to be far by taking $\xi_1 = 10$. Figure 7 shows the results.

Again, with a sufficient number of sample points we obtain a rational approximant of correct type (4, 5). In the middle error plot in Fig. 7, the poles inside the unit disk are computed accurately to machine precision. By contrast, the pole $\xi_1 = 10$ is computed with poorer accuracy. Loss of accuracy for poles outside the unit disk is a typical phenomenon, and the accuracy worsens rapidly if we take $|\xi_1|$ larger or let f be of higher type. This observation can be explained via eigenvalue conditioning analysis, which shows the condition numbers of the eigenvalues of (4.4) corresponding to poles outside the unit disk grow exponentially with base $|\xi_i|$ and exponent $m + n$, whereas those of eigenvalues inside the unit circle decrease (slowly) with $m + n$. The analysis is presented in “Appendix C”.

Recall that `ratfun` finds a numerical type by Algortihm 3.1. As explained in Sect. 3.1, there can be other numerical types for f that may be appropriate: Indeed, if we sample at many more points than necessary (i.e., `typefind(f,tol,L)` with $L \gg m+n+1$), `ratfun` eventually ignores the outlying pole and converges to a rational function of type $(m, 4)$ where m is large. That is, the computed outcome has lower denominator degree than that of the exact type 5; recall the experiment with (3.7) with a similar discussion. This can be explained as follows. By a standard result in complex analysis [34, Ch. 9], inside the unit disk a meromorphic function f can be written as

$$f(z) = \sum_{|\xi_i|\le 1} \sum_j \frac{a_{i,j}}{(z-\xi_i)^j} + p_f(z). \tag{6.1}$$

The sum is taken over the poles inside the unit disk. Here $p_f(z)$ is a power series, obtained e.g. as a Taylor series of $f(z) - \sum_{|\xi_i|\le 1}\sum_j \frac{a_{i,j}}{(z-\xi_i)^j}$, which converges inside a disk centered at the origin and of radius $|\xi_0|$, where ξ_0 is the pole closest to the origin besides those with $|\xi_i| \le 1$; here $|\xi_0| = 10$. Therefore, near the sample points (the unit circle), f behaves as if it is a sum of terms $1/(z-\xi_i)$ with $|\xi_i| \le 1$, and an analytic function.

From a practical viewpoint, this example suggests that we should locate the sample points near the poles of interest. For example, we can find the pole $\xi_1 = 10$ accurately in the above example by taking the sample points to lie on a circle centered around 10.

When a sample point is near a pole This example illustrates how existing algorithms lose accuracy when a sample point is near a pole.

We form a rational function $f(z) = \frac{\prod_{i=0}^{M}(z-r_i)}{\prod_{i=0}^{N}(z-\xi_i)}$ where the roots r_i and poles ξ_i are generated randomly to lie in the unit disk. Here we take $M = 4$, $N = 5$, and let the sample points be equispaced points on the unit circle. We then reset one pole to be $1 + 10^{-13}$, forcing it to lie close to a sample point.

`ratfun` and the naive method compute the poles much more accurately than the other methods. This is largely due to the diagonal scaling discussed in Sect. 5; indeed, if we turn off the diagonal scaling and take $D = I$, the accuracy deteriorates for both `ratfun` and the naive methods to about the same as Chebfun's `ratinterp` and RKFIT.

We believe that with RKFIT, which allows for tuning various inputs and parameters, it is possible to obtain accurate results if appropriate parameters are provided, such as $\tilde{D}$; recall the discussion in Sect. 2.3. The point here is that our analysis revealed an appropriate choice (Fig. 8).

Rational functions with poles of order > 1 When f is memorophic but has poles ξ_i of order $d_i > 1$, the generalized eigenvalue problems (4.4) and (4.15) have an eigenvalue ξ_i of the same multiplicity d_i. Here we examine the behavior of our algorithm in such cases.

We generate the function f simply by squaring the function in (3.5), that is, $f(z) = (\sum_{j=1}^{5} \frac{1}{z-\xi_j})^2$ with $\xi_j = 0.9\exp(2\pi \mathrm{i} j/5)$. Then f has 5 poles, all of which are of order 2.

Observe that all the algorithms, including `ratfun`, find the poles with accuracy $O(\sqrt{\epsilon})$, which is what one would expect from a backward stable algorithm: the poles

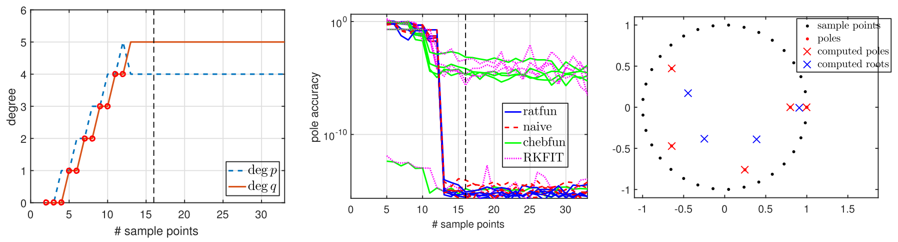

Fig. 8 f with a pole close to a sample point. `ratfun(f)` samples at 2^4 points and returns the correct type. Left: numerical degrees of the rational approximants. Middle: error of computed poles $\min_j |\xi_i - \hat{\xi}_j|$. Right: sample points, poles and roots

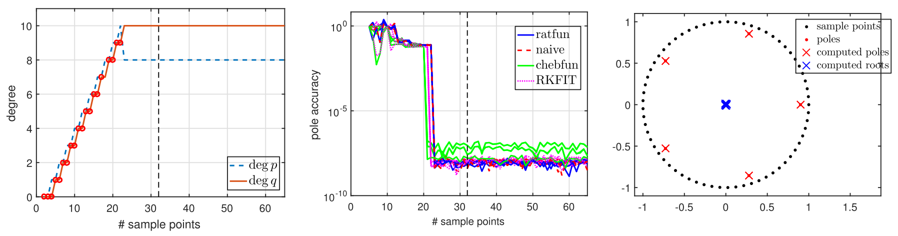

Fig. 9 f with double poles. Left: numerical degrees of the rational approximants. Middle: error of computed poles $\min_j |\xi_i - \hat{\xi}_j|$. Right: sample points, poles and roots

ξ_i of order 2 result in an eigenvalue with Jordan block of size 2, and perturbation of ϵ in the matrices perturb such eigenvalues by $O(\sqrt{\epsilon})$ (Fig. 9).

Non-meromorphic functions Although we have started our discussion assuming f is a meromorphic function in the unit disk, our algorithm can be applied to f with singularities other than poles, as long as f can be evaluated at the sample points. We now explore such cases by examining functions with a branch cut, or an essential singularity.

First let f have a log-type branch cut

$$f(z) = \log\left(z - \frac{1}{10}i\right), \tag{6.2}$$

which has a branch cut on $z = \frac{1}{10}i - \mathbb{R}_+$. Figure 10 shows the results. Observe that spurious poles and roots appear along the branch cut; we suspect this is related to a similar phenomenon known for Padé approximants of functions with branch cuts [42].

For a function with an essential singularity, we examine the standard example

$$f(z) = \exp\left(\frac{1}{z}\right). \tag{6.3}$$

The results are in Fig. 11. Again, spurious poles and roots appear near the singularity point 0, but away from the singularity f is bounded and analytic, and is well approximated by the rational interpolant. This is no surprise as $\exp(1/z)$ behaves as a completely analytic function on the unit circle.

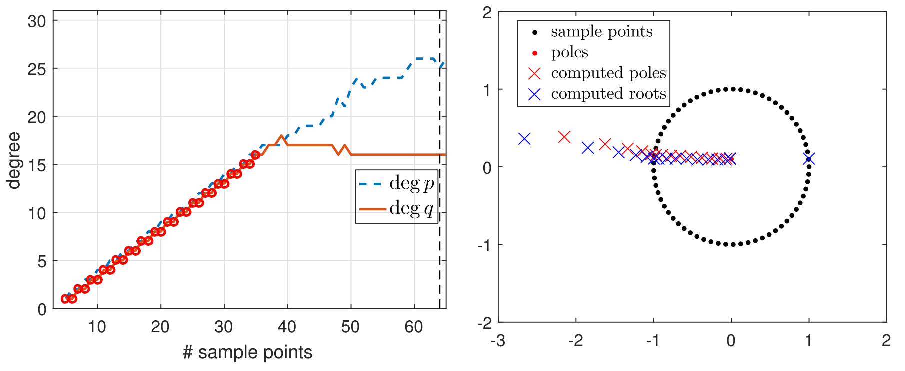

Fig. 10 f with a log-type branch cut. `ratfun(f)` samples at 2^5 points and determines the numerical type (14, 14). Left: numerical degrees of the rational approximants. Right: sample points, poles and roots

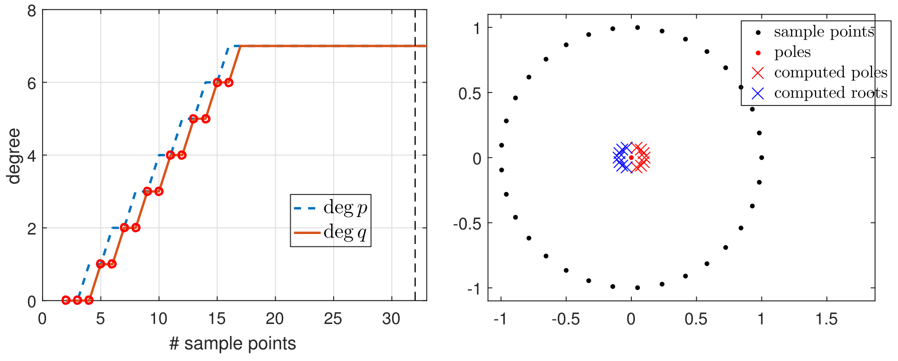

Fig. 11 f with an essential singularity. Left: numerical degrees of the rational approximants. `ratfun(f)` samples at 2^4 points and determines the numerical type (7, 7). Right: sample points, poles and roots

Sample points at Chebyshev points In this example the sample points are taken to be the Chebyshev points $\gamma_j = \cos(\frac{\pi(j-1)}{L-1})$ and the polynomial basis is Chebyshev polynomials $\phi_i(x) = T_i(x)$. This is numerically recommended when most poles lie on the real axis. In this example f is again as in (3.5), with 6 equispaced poles on $[-1 + 10^{-2}, 1 - 10^{-2}]$, along with complex poles at 0.2i and 2i. The results are in Fig. 12. For such functions, sampling at Chebyshev points give better accuracy than roots of unity.

Although not shown in the figure, the accuracy of poles far from $[-1, 1]$ worsens rapidly as the poles lie farther away, or the function type increases. This is analogous to the observation made in Fig. 7: the poles far from the sample points will eventually get ignored (here the poles that converge are those within a narrow ellipse that covers the real interval $[-1, 1]$).

Speed illustration Here we examine the speed and accuracy of `ratfun` for high-degree rational functions. We take f to be as in (3.5) with poles ξ being the Chebyshev points, scaled by $1 - 10^{-5}$, and vary the number of points (i.e., the degree of q) from 100 to 1000. We sample at the Chebyshev points. In order to examine the breakdown

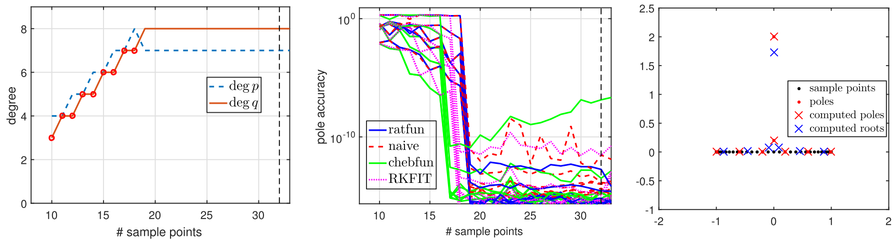

Fig. 12 Sampled at Chebyshev points. `ratfun(f)` samples at 2^5 points and determines the correct type $(8, 7)$. The pole at 2i loses accuracy as we sample more and increase m

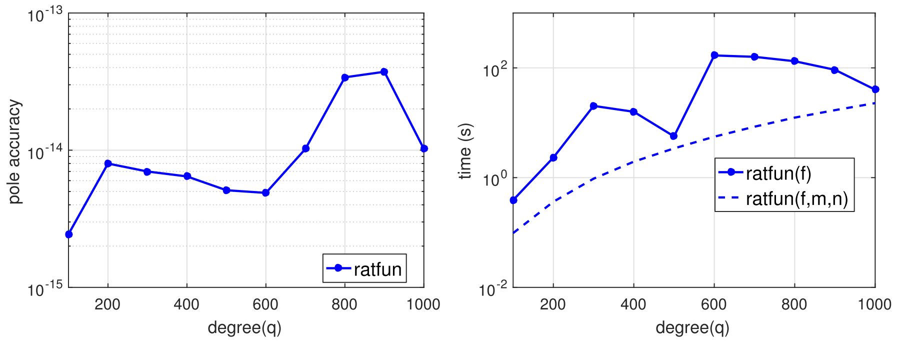

Fig. 13 High-degree example, accuracy of computed poles $\max_j |\xi_j - \hat{\xi}_j|$ (left) and runtime (right)

of the runtime we present the runtime for (1) `ratfun(f)`, which inputs only the function (hence starts by finding the type), and (2) `ratfun(f,m,n)`, which inputs the correct type (and hence bypasses the type determination). The results are in Fig. 13. This example illustrates that `ratfun` can work with rational functions of quite high degree, and that the degree determination step often takes up a dominant part of the runtime.

Eigenvalues of a matrix via the resolvent One use of rational approximation and polefinding that has been attracting recent interest [4] is in finding eigenvalues of a matrix A or matrix pencil $A - \lambda B$ via finding the poles of the projected resolvent $u^\top(A - zI)^{-1}v$ or $u^\top(A - zB)^{-1}v$, where u, v are some vectors (usually random). We have applied our algorithm to this problem, and observed that it works. However, usually it is not superior to the algorithm presented in [4], which combines a rational filter function with a block subspace whose dimension is proportional to the estimated number of eigenvalues in the region of interest. The distinct feature in [4] (and also the FEAST eigensolver [36]) is that the algorithm works with the subspaces $(A - zB)^{-1}V$ instead of the function $u^\top(A - zB)^{-1}v$, and this is crucial to overcome the difficulty associated with a nearly multiple eigenvalue. We suspect that an extension of our algorithm to work with block subspaces would be possible; we leave this for future work.

Acknowledgements We thank Nick Trefethen for providing many suggestions, particularly on clarifying what is meant by a numerical type. We are grateful to Amit Hochman for suggesting the use of Arnoldi orthogonalization for the QR factorizations, Stefan Guettel for discussions on RKFIT, and Anthony Austin and Olivier Séte for their comments on an early draft. We thank the referees for their valuable suggestions.

A The Kronecker canonical form

Here we analyze in detail the generalized eigenvalue problem (4.4) and derive its Kronecker canonical form [17, Ch. 4.5.2]. It shows in particular that multiple poles (if any) can be computed along with their multiplicities, at least in exact arithmetic.

Here, let f be a rational function $f(z) = \frac{p(z)}{q(z)} \in \mathcal{R}_{m,n}$, where $p(z)$ and $q(z)$ have no common divisors except for constants (i.e., $f = p/q$ is an *irreducible* expression). Then $p(z)$ and $q(z)$ are in the following form:

$$p(z) = \alpha \prod_{j=1}^{J} (z - \eta_j)^{c_j}, \quad q(z) = \beta \prod_{k=1}^{K} (z - \xi_k)^{d_k}, \tag{A.1}$$

where $\eta_1, \ldots, \eta_J, \xi_1, \ldots, \xi_K \in \mathbb{C}$ are distinct and $\alpha, \beta \in \mathbb{C} \setminus \{0\}$. Each η_j is a root of f of multiplicity c_j and each ξ_k is a pole of order d_k. Note that $M = \deg p = \sum_{j=1}^{J} c_j \le m$ and $N = \deg q = \sum_{k=1}^{K} d_k \le n$ since $f \in \mathcal{R}_{m,n}$. For simplicity we analyze the case where $L = m + n + 1$.

Proposition 2 *The matrix pencil $A - \lambda B = [A_1, A_2] - \lambda[B_1, O]$ is strictly equivalent to the matrix pencil*

$$F(\lambda) = \left[\begin{array}{c|c|c} \begin{matrix} H_1(\lambda) & & \\ & \ddots & \\ & & H_K(\lambda) \end{matrix} & O_{N,m+1} & O_{N,n-N} \\ \hline O_{L-N,N} & \begin{matrix} I_{M+1} \\ \\ O_{n-N,m+1} \end{matrix} & \begin{matrix} O_{M,n-N} \\ U(\lambda) \\ O_{m-M,n-N} \end{matrix} \end{array}\right], \tag{A.2}$$

where

$$H_k(\lambda) = \begin{bmatrix} \xi_k - \lambda & 1 & & \\ & \xi_k - \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \xi_k - \lambda \end{bmatrix} \in \mathbb{C}[\lambda]^{d_k \times d_k}, \quad (k = 1, \ldots, K) \tag{A.3}$$

$$U(\lambda) = \begin{bmatrix} -\lambda & & & \\ 1 & -\lambda & & \\ & 1 & \ddots & \\ & & \ddots & -\lambda \\ & & & 1 \end{bmatrix} \in \mathbb{C}[\lambda]^{(n-N+1)\times(n-N)}. \tag{A.4}$$

(Proof) It suffices to show that there exist nonsingular constant matrices $P, Q \in \mathbb{C}^{L\times L}$ satisfying $(A-\lambda B)P = QF(\lambda)$. We will construct such P and Q in two steps: (1) construct $P_k, Q_k \in \mathbb{C}^{(m+n+1)\times d_k}$ satisfying

$$(A-\lambda B)P_k = Q_k H_k(\lambda) \tag{A.5}$$

for $k = 1, \dots, K$, and (2) construct $S \in \mathbb{C}^{L\times(L-N)}$, $T_1 \in \mathbb{C}^{L\times(m+1)}$ and $T_2 \in \mathbb{C}^{L\times(n-N)}$ such that (using MATLAB notation)

$$(A-\lambda B)T_1 = S(:, 1 : m+1), \tag{A.6}$$

$$(A-\lambda B)T_2 = S(:, M+1 : n+M-N)U(\lambda). \tag{A.7}$$

Then, P and Q defined by

$$P = [P_1, \dots, P_K, T_1, T_2], \quad Q = [Q_1, \dots, Q_K, S] \tag{A.8}$$

satisfy $(A-\lambda B)P = QF(\lambda)$.

Before discussing Step (1), let us introduce some notation. For functions $g(z)$ in z and sample points $\{\gamma_i\}_{i=1}^{L}$, define the "vector of values" as

$$v(g(z)) := [g(\gamma_1), \dots, g(\gamma_L)]^\top \in \mathbb{C}^L. \tag{A.9}$$

Similarly, for $p_1(z) = \sum_{j=0}^{n-1} a_j z^j \in \mathcal{P}_{n-1}$ and $p_2(z) = \sum_{j=0}^{m} b_j z^j \in \mathcal{P}_m$, define the "vector of coefficients" as

$$c(p_1, p_2) := [a_0, \dots, a_{n-1}, b_0, \dots, b_m]^\top \in \mathbb{C}^L. \tag{A.10}$$

Then, it holds for arbitrary λ that

$$(A-\lambda B)c(p_1, p_2) = v((z-\lambda)f(z)p_1(z) + p_2(z)). \tag{A.11}$$

(1) We will construct $P_k, Q_k \in \mathbb{C}^{(L)\times d_k}$ satisfying (A.5) for each $k \in \{1, \dots, K\}$. For $d = 0, 1, \dots, d_k$, define polynomials $q_k^{(d)} \in \mathcal{P}_{n-1}$ by

$$q_k^{(d)}(z) := q(z) \sum_{j=1}^{d} \frac{1}{(z-\xi_k)^j}, \tag{A.12}$$

where we define $q_k^{(0)} = 0$. Then, for $d = 1, \dots, d_k$, the polynomial $q_k^{(d)}$ satisfies

$$\begin{aligned}(z-\lambda)f(z)q_k^{(d)}(z) - p(z) &= p(z)\left(\sum_{j=1}^{d} \frac{z-\lambda}{(z-\xi_k)^j} - 1\right)\\ &= p(z)\left(\sum_{j=1}^{d} \frac{(z-\xi_k)+(\xi_k-\lambda)}{(z-\xi_k)^j} - 1\right)\\ &= p(z)\left((\xi_k-\lambda)\sum_{j=1}^{d} \frac{1}{(z-\xi_k)^j} + \sum_{j=1}^{d} \frac{1}{(z-\xi_k)^{j-1}} - 1\right)\\ &= p(z)\left((\xi_k-\lambda)\sum_{j=1}^{d} \frac{1}{(z-\xi_k)^j} + \sum_{j=1}^{d-1} \frac{1}{(z-\xi_k)^{j}}\right)\\ &= (\xi_k-\lambda)f(z)q_k^{(d)}(z) + f(z)q_k^{(d-1)}(z).\end{aligned}$$

From this equation and (A.11), we see that $P_k, Q_k \in \mathbb{C}^{(L)\times d_k}$ defined by

$$\begin{aligned}P_k &= [c(q_k^{(1)}(z), -p(z)), \dots, c(q_k^{(d_k)}(z), -p(z))],\\ Q_k &= [v(f(z)q_k^{(1)}(z)), \dots, v(f(z)q_k^{(d_k)}(z))]\end{aligned} \tag{A.13}$$

satisfy

$$\begin{aligned}(A-\lambda B)P_k(:,1) &= (\zeta_k-\lambda)Q_k(:,1),\\ (A-\lambda B)P_k(:,j) &= (\zeta_k-\lambda)Q_k(:,j) + Q_k(:,j-1), \quad j = 2,\dots,d_k,\end{aligned}$$

which means that $(A-\lambda B)P_k = Q_k H_k(\lambda)$.

(2) Define $S \in \mathbb{C}^{L\times(L-N)}$, $T_1 \in \mathbb{C}^{L\times(m+1)}$ and $T_2 \in \mathbb{C}^{L\times(n-N)}$ by

$$T_1 = [c(0,z^0), \dots, c(0,z^{M-1}), c(0,z^0 p(z)), \dots, c(0,z^{m-M}p(z))], \tag{A.14}$$

$$T_2 = [c(z^0 q(z), 0), \dots, c(z^{n-N-1}q(z), 0)], \tag{A.15}$$

$$S = [v(z^0), \dots, v(z^{M-1}), v(z^0 p(z)), \dots, v(z^{m+n-N-M}p(z))]. \tag{A.16}$$

By substituting $p_1 = 0$ into (A.11), we obtain $(A-\lambda B)c(0, p_2(z)) = v(p_2(z))$ for an arbitrary $p_2 \in \mathcal{P}_m$, which implies (A.6). Since (A.11) gives

$$(A-\lambda B)c(z^{j-1}q(z), 0) = v((z-\lambda)f(z)z^{j-1}q(z)) = -\lambda v(z^{j-1}p(z)) + v(z^j p(z)) \tag{A.17}$$

for $j = 1, \dots, n-N$, we have (A.7).

From (A.5), (A.6) and (A.7), P and Q defined by (A.8) satisfy $(A-\lambda B)P = QF(\lambda)$.

It remains to show that P and Q are nonsingular. This is proven in the next lemma. □

Lemma 2 *The matrices P and Q defined by* (A.13)*,* (A.16)*,* (A.14)*,* (A.15) *and* (A.8) *are nonsingular.*

(Proof) Since $T_1(1:n,:)$ is a zero matrix and $T_1(n+1:m+n+1,:)$ is an upper triangular matrix with non-zero diagonal entries, $P = [P_1,\dots,P_K,T_1,T_2]$ is nonsingular if and only if $W = [P_1,\dots,P_K,T_2](1:n,:)$ is nonsingular. We shall prove that R is nonsingular by showing that $Wx = 0$ implies $x = 0$. Write $x = [x_1^{(1)},\dots,x_1^{(d_1)},x_2^{(1)},\dots,x_K^{(d_K)},y_1,\dots,y_{n-N}]^\top \in \mathbb{C}^n$. If $Wx = 0$, from the definitions (A.13), (A.15) of P_k and T_2, all the coefficients of the polynomial

$$p_x(z) = \sum_{k=1}^{K}\sum_{j=1}^{d_k} x_k^{(j)} q_k^{(j)}(z) + \sum_{i=1}^{n-N} y_i z^{i-1} q(z) \tag{A.18}$$

are equal to zero, and hence p_x is the zero polynomial. Therefore, the rational function

$$\frac{p_x(z)}{q(z)} = \sum_{k=1}^{K}\sum_{j=1}^{d_k} x_k^{(j)} \sum_{s=1}^{j} \frac{1}{(z-\xi_k)^s} + \sum_{i=1}^{n-N} y_i z^{i-1} = \sum_{k=1}^{K}\sum_{s=1}^{d_k} \frac{\sum_{j=s}^{d_k} x_k^{(j)}}{(z-\xi_k)^s} + \sum_{i=1}^{n-N} y_i z^{i-1} \tag{A.19}$$

is also the zero function. This means that

$$\sum_{j=s}^{d_k} x_k^{(j)} = 0 \quad (k = 1,\dots,K, s = 1,\dots,d_k), \tag{A.20}$$

$$y_i = 0 \quad (i = 1,\dots,n-N), \tag{A.21}$$

which implies that all the elements in x are zero. It follows from the above argument that P is nonsingular.

We prove that Q is nonsingular similarly by showing that $Qx = 0$ implies $x = 0$. Write $x = [x_1^{(1)},\dots,x_1^{(d_1)},x_2^{(1)},\dots,x_K^{(d_K)},y_1,\dots,y_M,w_1,\dots,w_{m+n-M-N+1}]^\top \in \mathbb{C}^{m+n+1}$. If $Qx = 0$, from the definitions (A.13) and (A.16) of Q_k and S, we have

$$g_x(z) := \sum_{k=1}^{K}\sum_{j=1}^{d_k} x_k^{(j)} f(z) q_k^{(j)}(z) + \sum_{i=1}^{M} y_i z^{i-1} + \sum_{l=1}^{m+n-M-N+1} w_l z^{l-1} p(z) = 0 \tag{A.22}$$

for $z = \gamma_1, \dots, \gamma_{m+n+1}$. By multiplying $q(z)$ to both sides, we have

$$\begin{aligned} h_x(z) := q(z)g_x(z) = \sum_{k=1}^{K}\sum_{j=1}^{d_k} p(z)x_k^{(j)}q_k^{(j)}(z) + \sum_{i=1}^{M} y_i z^{i-1}q(z) \\ + \sum_{l=1}^{m+n-M-N+1} w_l z^{l-1} q(z)p(z) = 0 \end{aligned} \tag{A.23}$$

for $z = \gamma_1, \dots, \gamma_{m+n+1}$. Since h_x is a polynomial of degree at most $m+n$ and take on the value 0 at $m+n+1$ distinct points, h_x must be the zero polynomial. Using this fact we obtain $x = 0$ as in the case of P, and hence Q is nonsingular. □

Corollary 1 *The Kronecker canonical form of the pencil $A - \lambda B$ is as follows:*

$$\left[\begin{array}{ccc|cccccc} H_1(\lambda) & & & & & & & \\ & \ddots & & & & & & \\ & & H_K(\lambda) & & & & & \\ \hline & & & I_{s_1} & & & & \\ & & & & J_{s_2}(\lambda) & & & \\ & & & & & L_1(\lambda) & & \\ & & & & & & \ddots & \\ & & & & & & & L_{s_3}(\lambda) \\ & & & & & & O_{s_3,2s_3} & \end{array}\right], \tag{A.24}$$

where $s_1 = M + \max\{0, m - n - M + N\}$, $s_2 = \max\{0, -m + n + M - N\} + 1$, $s_3 = \min\{m - M, n - N\}$,

$$J_{s_2}(\lambda) = \left.\begin{bmatrix} 1 & -\lambda & & \\ & 1 & \ddots & \\ & & \ddots & -\lambda \\ & & & 1 \end{bmatrix}\right\} s_2, \tag{A.25}$$

$$L_j(\lambda) = [1, -\lambda], \quad j = 1, \dots, s_3 \tag{A.26}$$

and $H_j(\lambda)$ is defined by (A.3) .

(Proof) We can transform the $(m+n-N) \times (m+n-N)$ lower-right block of $L(\lambda)$ to obtain the Kronecker canonical form as follows:

$$\begin{bmatrix} 1 &&&&&&& \\ & 1 &&& -\lambda &&& \\ && 1 && 1 & -\lambda && \\ &&& 1 && 1 & -\lambda & \\ &&&&&& 1 & -\lambda \\ &&&&&&& 1 \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \end{bmatrix} \underset{\text{erase } s_3 \text{ 1's}}{\longrightarrow} \begin{bmatrix} 1 &&&&&&& \\ & 1 &&& -\lambda &&& \\ && 1 &&& -\lambda && \\ &&& 1 &&& -\lambda & \\ &&&&&& 1 & -\lambda \\ &&&&&&& 1 \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \end{bmatrix}$$

$$\underset{\substack{\text{column}\\\text{permutation}}}{\longrightarrow} \begin{bmatrix} 1 &&&&&&& \\ &&&& 1 & -\lambda && \\ &&&&&& 1 & -\lambda \\ & 1 & -\lambda &&&&& \\ && 1 & -\lambda &&&& \\ &&& 1 &&&& \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \end{bmatrix} \underset{\substack{\text{row}\\\text{permutation}}}{\longrightarrow} \begin{bmatrix} 1 &&&&&&& \\ & 1 & -\lambda &&&&& \\ && 1 & -\lambda &&&& \\ &&& 1 &&&& \\ &&&& 1 & -\lambda && \\ &&&&&& 1 & -\lambda \\ &&&&&&& \\ &&&&&&& \\ &&&&&&& \end{bmatrix}.$$

B Rank of the rational interpolation matrix

Here we analyze the rank of the matrix in (2.6) and derive (3.2). For a rational function $f = r$ sampled at $\{\gamma_1, \ldots, \gamma_L\}$, define $C_{mn}(r) \in \mathbb{C}^{L\times(m+n+2)}$ by $C_{mn}(r) = [FV_{n+1}, V_{m+1}]$, where $F = \operatorname{diag}(r(\gamma_1), \ldots, r(\gamma_L))$. Then we have

$$\operatorname{range}(C_{mn}(r)) = \{v(rp_1 + p_2) \mid p_1 \in \mathcal{P}_n, p_2 \in \mathcal{P}_m\}, \tag{B.1}$$

where we define $v(h) = [h(\gamma_1), \ldots, h(\gamma_L)]^\top \in \mathbb{C}^L$ for functions h.

In this section we focus on the case where r is a rational function, and assume that r does not have poles coinciding with the sample points $\gamma_1, \ldots, \gamma_L$. When $r = p/q$ is an irreducible expression, the degrees M, N of p, q are uniquely determined, dependening only on r. For these M and N, we say that r is of *exact type* (M, N).

Below we summarize the properties of the matrix $C_{mn}(r)$ when r is a rational function of exact type (M, N) and has an irreducible expression $r = p/q$. Note that r is not necessarily in $\mathcal{R}_{mn}$.

From (B.1), we have

$$\operatorname{range}(C_{mn}(r)) = \{v((pp_1 + qp_2)/q) \mid p_1 \in \mathcal{P}_n, p_2 \in \mathcal{P}_m\}. \tag{B.2}$$

Let $D := \max\{M + n, m + N\}$. Defining a subspace $\mathcal{Q}$ of $\mathcal{P}_D$ by

$$\mathcal{Q} := \{pp_1 + qp_2 \mid p_1 \in \mathcal{P}_n, p_2 \in \mathcal{P}_m\}, \tag{B.3}$$

we obtain

$$\mathrm{range}(C_{mn}(r)) = \{v(p_3/q) \mid p_3 \in \mathcal{Q}\}. \tag{B.4}$$

Lemma 3 *If $r \in \mathcal{R}_{mn}$, i.e., $M \le m$ and $N \le n$, then $\mathcal{Q} = \mathcal{P}_D$.*

(Proof) It suffices to show that $\dim \mathcal{Q} \ge D + 1$. Since $\mathcal{Q} = p\mathcal{P}_n + q\mathcal{P}_m$, we have

$$\begin{aligned}\dim(\mathcal{Q}) &= \dim(p\mathcal{P}_n) + \dim(q\mathcal{P}_m) - \dim(p\mathcal{P}_n \cap q\mathcal{P}_m)\\ &= m + n + 2 - \dim(p\mathcal{P}_n \cap q\mathcal{P}_m).\end{aligned} \tag{B.5}$$

Since p and q are relatively prime, any polynomial in $p\mathcal{P}_n \cap q\mathcal{P}_m$ is a multiple of pq. Hence, we have $p\mathcal{P}_n \cap q\mathcal{P}_m \subseteq pq\mathcal{P}_d$, where $d = \min\{m - M, n - N\}$. This means $\dim(p\mathcal{P}_n \cap q\mathcal{P}_m) \le \dim \mathcal{P}_d = d + 1$, and hence by (B.5) $\dim(\mathcal{Q}) \ge m + n + 2 - (d + 1) = D + 1$. □

Lemma 4 *If r does not belong to $\mathcal{R}_{mn}$, i.e., $M > m$ or $N > n$, then* $\dim \mathcal{Q} = m+n+2$.

(Proof) This follows from (B.5) and the fact that $p\mathcal{P}_n \cap q\mathcal{P}_m = \{0\}$. □

We now prove results on the rank of the matrix $C_{mn}(r)$ that we use for finding the type of the rational approximant r in our algorithm. The first result shows that if we take m, n large enough so that $m \ge M, n \ge N$, then the rank of $C_{mn}(r)$ gives the information on M, N.

Proposition 3 *Assume that $r \in \mathcal{R}_{mn}$ is of exact type (M, N) and $r = p/q$ is an irreducible expression of r. Then, we have*

$$\mathrm{range}(C_{mn}(r)) = \{v(p_1/q) \mid p_1 \in \mathcal{P}_D\}, \tag{B.6}$$

where $D =$ $\max\{M + n, m + N\}$.

(Proof) This follows immediately from (B.4) and Lemma 3. □

Note that $m \ge M, n \ge N$ is implied by the assumptions in Proposition 3. From (B.6) we see that the rank of $C_{mn}(r)$ is

$$\mathrm{rank}(C_{mn}(r)) = D + 1, \tag{B.7}$$

as long as $L \ge D+1$, which always holds in our algorithm in which $L \ge m+n+1 \ge D+1$. Thus we obtain $\dim \mathrm{null}(C) = m+n+2-(D+1) = \min(m-M, n-N)+1$, which is (3.2).

The next result shows that if we do not take m, n large enough then this is indicated by $C_{mn}(r)$ not having a null vector, provided we still sample at sufficiently many points.

Proposition 4 *Suppose that r of exact type (M, N) does not belong to $\mathcal{R}_{mn}$, i.e., $M > m$ or $N > n$. If $L \ge$ $\max\{M + n, m + N\} + 1$, then the rank of $C_{mn}(r)$ is equal to $m + n + 2$, i.e., $C_{mn}(r)$ has full column rank.*

(Proof) This follows from (B.4) and Lemma 4. □

The above two propositions indicate that we can obtain the type (M, N) of r by combining (1) sufficient sample points, and (2) adjusting m, n so that $C_{mn}(r)$ has null space of exactly dimension 1. This is the crux of the degree determination process described in Sect. 3.1.

B.1 Analysis of our generalized eigenvalue problem

The above results provide information on the building-block eigenvalue problem (4.4) in terms of its regularity and eigenvalues. We say that a (possibly rectangular) matrix pencil $A - \lambda B$ is *regular* if the matrix $A - \lambda B$ has full column rank for some value of λ.

Proposition 5 *Suppose that f is a rational function of exact type (M, N) with $M \le m$ and $N \le n$, and $L \ge m + n + 1$. If $M = m$ or $N = n$, then the matrix pencil $[A_1, A_2] - \lambda[B_1, O]$ is regular and its finite eigenvalues coincide with the poles of f.*

(Proof) The matrix pencil $[A_1, A_2] - \lambda[B_1, O]$ is equal to $C_{m,n-1}((z - \lambda)f(z))$.

If λ is a pole of f, then $(z - \lambda)f(z)$ is of exact type $(M, N - 1)$, and consequently, we have

$$\operatorname{rank}(C_{m,n-1}((z - \lambda)f(z))) = \max\{M + n - 1, m + N - 1\} + 1 = m + n$$

by Proposition 3, hence $C_{m,n-1}((z - \lambda)f(z))$ is rank deficient.

Conversely, if λ is not a pole of f, then $(z - \lambda)f(z)$ is of exact type $(M + 1, N)$, and hence by Proposition 4, $C_{m,n-1}((z - \lambda)f(z)))$ has full column rank. □

We are thus able to correctly compute the poles of f provided that we take one of m, n to be the correct value M, N.

C Condition number of eigenvalues

Here we analyze the condition number of the eigenvalues of the matrix pencil $[A_1, A_2] - \lambda[B_1, O]$, which here we write simply as $A - \lambda B$. For simplicity we focus on the case where the sample points are roots of unity, and examine the conditioning as n is fixed and the number of sample points grow along with m. We shall show that the eigenvalues outside the unit disk become increasingly ill-conditioned, which explains the observation in Fig. 7.

Assume that $f = p/q$ is irreducible and f has N simple poles $\xi_1, \dots, \xi_N$. We consider the square generalized eigenvalue problem (4.4) where $n = N$ (i.e., the denominator degree is fixed to the correct value), $L = m + n + 1$ and $\{\gamma_j\}_{j=1}^{L} = \{\exp(2\pi \mathrm{i} \cdot j/L)\}_{j=1}^{L}$. For $m \ge M$, $A - \lambda B$ has eigenvalue equal to ξ_k for each k. We will investigate how the condition number of each eigenvalue ξ_k change, when we increase m (and hence also $L = m + n + 1$).

The absolute condition number of a simple eigenvalues of a matrix pencil $A - \lambda B$ is known [45] to be proportional to $\|x\|\|y\|/|y^\top Bx|$, where y and x are the corresponding left and right eigenvectors. Thus we need to identify the eigenvectors for ξ_k.

The right eigenvector x_k such that $[A_1, A_2]x_k = \xi_k[B_1, O]x_k$ is given by $x_k = [c_{\tilde{q}}^\top, c_p^\top]^\top$, where $\tilde{q}(z) = q(z)/(z-\xi_k)$. This x_k satisfies $Bx_k = v(f\tilde{q})$.

To find the left eigenvector, first note that as in Proposition 3 we have

$$\mathrm{range}(A - \xi_k B) = \{v(p_1/\tilde{q}) \mid p_1 \in \mathcal{P}_{L-2}\}.$$

Now since $\{\gamma_j\}$ are Lth roots of unity, the vector

$$y_k^\top = [\tilde{q}(\gamma_1)\gamma_1, \dots, \tilde{q}(\gamma_L)\gamma_L]$$

satisfies $y_k^\top(A - \xi_k B) = 0$, indicating y_k is the left eigenvector.

Hence, we have

$$\begin{aligned} y_k^\top B x_k = y_k^\top B_1 c_{\tilde{q}} = \sum_{j=1}^{L} f(\gamma_j)\tilde{q}(\gamma_j)^2\gamma_j = \sum_{j=1}^{L} \frac{p(\gamma_j)\tilde{q}(\gamma_j)^2\gamma_j}{q(\gamma_j)} \\ = \sum_{j=1}^{L} \frac{p(\gamma_j)\tilde{q}(\gamma_j)\gamma_j}{\gamma_j - \xi_k}. \end{aligned} \tag{C.1}$$

This implies that $y_k^\top Bx_k/L$ is an approximate value of

$$I = \frac{1}{2\pi \mathrm{i}} \int_{|z|=1} \frac{p(z)\tilde{q}(z)}{z-\xi_k} \mathrm{d}z = \begin{cases} 0 & (|\xi_k| > 1) \\ p(\xi_k)\tilde{q}(\xi_k) & (|\xi_k| < 1). \end{cases} \tag{C.2}$$

In fact, the trapezoidal rule approximation to I with sample points $\{\gamma_1, \gamma_2, \dots, \gamma_L\}$ is given by

$$I_L = \frac{1}{2\pi \mathrm{i}} \sum_{j=1}^{L} \frac{p(\gamma_j)\tilde{q}(\gamma_j)}{\gamma_j - \xi_k} \frac{2\pi \mathrm{i}\gamma_j}{L} = \frac{1}{L}\sum_{j=1}^{L} \frac{p(\gamma_j)\tilde{q}(\gamma_j)\gamma_j}{\gamma_j - \xi_k} = \frac{y_k^\top B x_k}{L}. \tag{C.3}$$

Now suppose that $|\xi_k| > 1$, that is, ξ_k lies outside the unit disk. Then, the integrand is analytic in the disc $|z| < |\xi|$ and have a simple pole on the circle $|z| = |\xi|$, and hence the trapezoidal rule approximation satisfies (see [49] for the trapezoidal rule and its properties)

$$|I_L - I| = O(|\xi_k|^{-L}). \tag{C.4}$$

From (C.2), (C.3) and (C.4), we have

$$y_k^\top B x_k = O(L|\xi_k|^{-L}). \tag{C.5}$$

Furthermore, since $\|y_k\| = \Theta(\sqrt{L})$ and $\|x_k\| = \Omega(1)$, the condition number $\kappa(\xi_k)$ of the eigenvalue ξ_k satisfies

$$\kappa(\xi_k) = \frac{\|y_k\|\|x_k\|}{|y_k B x_k|} = \Omega(L^{-\frac{1}{2}}|\xi_k|^L) \quad \text{for } |\xi_k| > 1. \tag{C.6}$$

This means that the condition numbers of eigenvalues outside the unit circle grow exponentially as we use more sample points.

On the other hand, if the eigenvalue ξ_k is inside the unit disk, i.e., $|\xi_k| < 1$, then the condition number of ξ_k behaves differently. Indeed, since we have

$$\lim_{L\to\infty} \frac{y_k^\top B x_k}{L} = I \neq 0, \tag{C.7}$$

we see that the condition number of the eigenvalue ξ_k satisfies

$$\kappa(\xi_k) = \frac{\|y_k\|\|x_k\|}{|y_k B x_k|} = O(L^{-\frac{1}{2}}) \quad \text{for } |\xi_k| < 1. \tag{C.8}$$

This is in sharp contrast to (C.6). Equations (C.6) and (C.8) imply that, when the number of sample points grows, the computed eigenvalues outside the unit disk lose accuracy exponentially, while those inside do not.

We confirmed this in our numerical experiments as the figures in Sect. 6 show.

References

1. Antoulas, A.C., Anderson, B.D.Q.: On the scalar rational interpolation problem. IMA J. Math. Control Inf. **3**(2–3), 61–88 (1986)
2. Aurentz, J.L., Trefethen, L.N.: Chopping a Chebyshev Series. ArXiv e-prints 1512.01803. Submitted to ACM Trans. Math. Softw. (2015)
3. Austin, A.P., Kravanja, P., Trefethen, L.N.: Numerical algorithms based on analytic function values at roots of unity. SIAM J. Numer. Anal. **52**(4), 1795–1821 (2014)
4. Austin, A.P., Trefethen, L.N.: Computing eigenvalues of real symmetric matrices with rational filters in real arithmetic. SIAM J. Sci. Comput. **37**(3), A1365–A1387 (2015)
5. Barnett, S.: Polynomials and Linear Control Systems. Marcel Dekker Inc., New York (1983)
6. Battles, Z., Trefethen, L.N.: An extension of MATLAB to continuous functions and operators. SIAM J. Sci. Comput. **25**(5), 1743–1770 (2004)
7. Berljafa, M., Güttel, S.: Generalized rational Krylov decompositions with an application to rational approximation. SIAM J. Matrix Anal. Appl. **36**(2), 894–916 (2015)
8. Berljafa, M., Güttel, S.: The RKFIT algorithm for nonlinear rational approximation, MIMS EPrint 2015.38 (2015)
9. Berrut, J.-P., Baltensperger, R., Mittelmann, H.D.: Recent developments in barycentric rational interpolation. In: Trends and Applications in Constructive Approximation, pp. 27–51. Springer (2005)
10. Berrut, J.-P., Mittelmann, H.D.: Matrices for the direct determination of the barycentric weights of rational interpolation. J. Comput. Appl. Math. **78**(2), 355–370 (1997)
11. Berrut, J.-P., Trefethen, L.N.: Barycentric Lagrange interpolation. SIAM Rev. **46**(3), 501–517 (2004)
12. Braess, D.: Nonlinear Approximation Theory. Springer, Berlin (1986)
13. Cauchy, A.L.: Sur la formule de Lagrange relative á l'interpolation. *Analyse algebraique*, Paris (1821)
14. Dahlquist, G., Björck, A., Anderson, N.: Numerical Methods. Prentice-Hall, Englewood Cliffs (1974)
15. Delves, L.M., Lyness, J.N.: A numerical method for locating the zeros of an analytic function. Math. Comput. **21**, 543–560 (1967)

16. Demmel, J.: The condition number of equivalence transformations that block diagonalize matrix pencils. SIAM J. Numer. Anal. **20**(3), 599–610 (1983)
17. Demmel, J.: Applied Numerical Linear Algebra. SIAM, Philadelphia (1997)
18. Driscoll, T.A., Hale, N., Trefethen, L.N.: Chebfun Guide. Pafnuty Publications, Oxford (2014)
19. Golub, G.H., Van Loan, C.F.: Matrix Computations, 4th edn. The Johns Hopkins University Press, Baltimore (2012)
20. Gonnet, P., Güettel, S., Trefethen, L.N.: Robust padé approximation via SVD. SIAM Rev. **19**(2), 160–174 (2013)
21. Gonnet, P., Pachón, R., Trefethen, L.N.: Robust rational interpolation and least-squares. Electron. Trans. Numer. Anal. **38**, 146–167 (2011)
22. Good, I.J.: The colleague matrix, a Chebyshev analogue of the companion matrix. Q. J. Math. **12**(1), 61–68 (1961)
23. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
24. Hochman, A., Leviatan, Y., White, J.K.: On the use of rational-function fitting methods for the solution of 2d laplace boundary-value problems. J. Comput. Phys. **238**, 337–358 (2013)
25. Ito, S., Murota, K.: An algorithm for the generalized eigenvalue problem for nonsquare matrix pencils by minimal perturbation approach. SIAM J. Matrix Anal. Appl. **37**(1), 409–419 (2016)
26. Jacobi, C.G.J.: Über die Darstellung einer Reihe gegebner Werthe durch eine gebrochne rationale Function. Journal für die reine und angewandte Mathematik **30**, 127–156 (1846)
27. Kravanja, P., Sakurai, T., Van Barel, M.: On locating clusters of zeros of analytic functions. Bit Numer. Math. **39**(4), 646–682 (1999)
28. Kravanja, P., Van Barel, M.: A derivative-free algorithm for computing zeros of analytic functions. Computing **63**(1), 69–91 (1999)
29. Kravanja, P., Van Barel, M.: Computing the Zeros of Analytic Functions. Number Lecture Notes in Math, p. 1727. Springer, Berlin (2000)
30. Martins, N., Lima, L.T.G., Pinto, H.J.C.P.: Computing dominant poles of power system transfer functions. IEEE Trans. Power Syst. **11**(1), 162–170 (1996)
31. Moler, C.B., Stewart, G.W.: An algorithm for generalized matrix eigenvalue problems. SIAM J. Numer. Anal. **10**(2), 241–256 (1973)
32. Nakatsukasa, Y., Noferini, N.: On the stability of computing polynomial roots via confederate linearizations, MIMS EPrint 2014.49. To appear in Math. Comput. (2014)
33. Nakatsukasa, Y., Sète, O., Trefethen, L.N.: The AAA algorithm for rational approximation. Technical report. Submitted to SIAM J. Sci. Comput. (2016)
34. Needham, T.: Visual Complex Analysis. Oxford University Press, Oxford (1998)
35. Pachón, R., Gonnet, P., Van Deun, J.: Fast and stable rational interpolation in roots of unity and Chebyshev points. SIAM J. Numer. Anal. **50**(3), 1713–1734 (2012)
36. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B **79**(11), 115112 (2009)
37. Powell, M.J.D.: Approximation Theory and Methods. Cambridge University Press, Cambridge (1981)
38. Rommes, J., Martins, N.: Efficient computation of transfer function dominant poles using subspace acceleration. IEEE Trans. Power Syst. **21**(3), 1218 (2006)
39. Rudelson, M., Vershynin, R.: Non-asymptotic theory of random matrices: extreme singular values. In: Proceedings of the International Congress of Mathematicians, vol. III, pp. 1576–1602. Hindustan Book Agency, New Delhi (2010). ArXiv:1003.2990
40. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. J. Comput. Appl. Math. **159**(1), 119–128 (2003)
41. Schneider, C., Werner, W.: Some new aspects of rational interpolation. Math. Comput. **47**(175), 285–299 (1986)
42. Stahl, H.: The convergence of Padé approximants to functions with branch points. J. Approx. Theory **91**(2), 139–204 (1997)
43. Stewart, G.W., Sun, J.-G.: Matrix Perturbation Theory (Computer Science and Scientific Computing). Academic Press, Cambridge (1990)
44. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer, Berlin (2002)
45. Tisseur, F.: Backward error and condition of polynomial eigenvalue problems. Linear Algebra Appl. **309**(1), 339–361 (2000)
46. Trefethen, L.N.: Spectral Methods in MATLAB. SIAM, Philadelphia (2000)
47. Trefethen, L.N.: Approximation Theory and Approximation Practice. SIAM, Philadelphia (2013)

48. Trefethen, L.N.: The doublelength flag. Chebfun examples (2015). http://www.chebfun.org/examples/cheb/DoublelengthFlag.html
49. Trefethen, L.N., Weideman, J.A.C.: The exponentially convergent trapezoidal rule. SIAM Rev. **56**(3), 385–458 (2014)
50. Van Der Sluis, A.: Condition numbers and equilibration of matrices. Numer. Math. **14**(1), 14–23 (1969)
51. Van Dooren, P., Dewilde, P.: The eigenstructure of an arbitrary polynomial matrix: computational aspects. Linear Algebra Appl. **50**, 545–579 (1983)
52. Wilkinson, J.H.: The perfidious polynomial. In: Golub, G.H. (ed.) Studies in Numerical Analysis, pp. 1–28. Mathematical Association of America, Washington, DC (1984)