

Improving performance of contour integral-based nonlinear eigensolvers with infinite GMRES

Yuqi Liu¹, Jose E. Roman², and Meiyue Shao^{3,4}

¹*School of Mathematical Sciences, Fudan University, Shanghai 200433, China*

²*D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain*

³*School of Data Science, Fudan University, Shanghai 200433, China*

⁴*Shanghai Key Laboratory for Contemporary Applied Mathematics, Fudan University, Shanghai 200433, China*

March 29, 2024

Abstract

In this work, the infinite GMRES algorithm, recently proposed by Correnty *et al.*, is employed in contour integral-based nonlinear eigensolvers, avoiding the computation of costly factorizations at each quadrature node to solve the linear systems efficiently. Several techniques are applied to make the infinite GMRES memory-friendly, computationally efficient, and numerically stable in practice. More specifically, we analyze the relationship between polynomial eigenvalue problems and their scaled linearizations, and provide a novel weighting strategy which can significantly accelerate the convergence of infinite GMRES in this particular context. We also adopt the technique of TOAR to infinite GMRES to reduce the memory footprint. Theoretical analysis and numerical experiments are provided to illustrate the efficiency of the proposed algorithm.

Keywords: Krylov methods, infinite Arnoldi, nonlinear eigenvalue problem, contour integration, companion linearization.

AMS subject classifications (2020). 65F10, 65F15, 65F50

1 Introduction

By nonlinear eigenvalue problem (NEP), we refer to

$$T(\lambda)v = 0, \quad v \in \mathbb{C}^n \setminus \{0\}, \quad \lambda \in \Omega, \quad (1)$$

where $\Omega \subseteq \mathbb{C}$ is a connected region with a smooth boundary, and $T(\xi): \Omega \rightarrow \mathbb{C}^{n \times n}$ is a ξ -dependent matrix [17, 26]. In the particular case that $T(\xi)$ is a matrix polynomial in ξ , (1) is also called a polynomial eigenvalue problem (PEP). Many well-known problems are of this type, such as the Orr–Sommerfeld equation [33] for the incompressible Navier–Stokes equation, and the vibration analysis of the damped beams [19].

Nowadays, an increasing number of non-polynomial NEPs arise from physics, chemistry, and industrial applications. These problems have remarkable practical and theoretical value. But

they are far more complicated to solve due to their large scale and nonlinearity. One emerging example appears in quasi normal mode (QNM) analysis [28], where the computation of a small set of eigentriplets of large, sparse rational eigenvalue problems

$$\sum_{j=1}^p r_j(\lambda) L_j v = 0, \quad r_j(\cdot): \mathbb{C} \rightarrow \mathbb{C}, \quad L_j \in \mathbb{C}^{n \times n}, \quad j = 1, \dots, p$$

is needed for modal expansions, whose results can be used for the physical understanding of nanophotonic devices. Actually, solving rational eigenvalue problems for QNM analysis is currently a very active field; see also [7, 24, 27]. Examples with other nonlinear functions can be found in [20], where nonlinear eigenvalue problems of the form

$$(A + \lambda B + \lambda^2 C)v = e^{i\lambda\tau} S v \quad (2)$$

are to be solved. Here, A , B , C , and S are constant matrices, and τ is a scalar. The solution of (2) can be used to obtain the acoustic modes of the Helmholtz wave equation with high efficiency and accuracy.

To address these problems, one can consider employing Krylov-based methods. This kind of algorithms reformulate the NEP as a generalized eigenvalue problem, and subsequently apply a Krylov algorithm to solve it. Depending on the specific approximations of $T(\xi)$, there are Taylor expansion-based algorithms [22], Chebyshev interpolation-based algorithms [23] and, more complicated, rational approximation-based algorithms [8, 18, 36].

Nevertheless, the Krylov-based methods suffer from the drawback that the eigenvalues have to be distributed in certain patterns. For example, Taylor expansion proves unsatisfactory when some eigenvalues are located far away from the expansion point. Chebyshev interpolation, on the other hand, is specifically designed for the eigenvalues lying exactly on the real axis or some pre-specified curves [12]. In some practical applications, users seek for the eigenvalues lying in certain regions, without prior knowledge of their number and distribution. In these cases, to capture all the eigenvalues by a single Taylor expansion point is almost impossible, not to say connecting them with a pre-specified curve. To overcome this, people may employ the rational Krylov method to target multiple points instead of just one. However, there is still no guarantee that all eigenvalues lying in the region of interest can be obtained.

Contour integral-based algorithms are another type of widely-used eigensolvers that are originally developed for linear eigenvalue problems. One of the most representative algorithms, the Sakurai–Sugiura method [30], transforms the original problems into a generalized eigenvalue problem of two small Hankel matrices from which it extracts eigenvalue approximations using standard dense eigensolvers. Nevertheless, computing these eigenvalues can be numerical unstable because the Hankel matrices, formed by higher moments, are usually ill-conditioned. Thus, people developed projection methods like FEAST [29, 32] and CIRR [31]. Similar to the Sakurai–Sugiura method, these methods approximate the characteristic subspace by integration rules, but after that, apply a projection on this subspace to avoid generating Hankel matrices.

When dealing with non-linear eigenvalue problems, the previously mentioned algorithms have corresponding extensions. In the case of the Sakurai–Sugiura method, all its theories follow and can be directly adapted to NEP cases [2]. However, the same ease does not apply to FEAST or CIRR, because computing an approximate solution of a general NEP on a projection subspace is no longer an easy task. Hence, nonlinear FEAST [15] or CIRR [37] need additionally an auxiliary solver for solving projected problems. To avoid the numerical instability of the Sakurai–Sugiura method and the uncertainty of nonlinear FEAST and CIRR, we will take Beyn’s algorithm [6] as the eigensolver for this work. It can be regarded as a nonlinear extension of the Sakurai–Sugiura algorithm using only the first two moments. We will make a detailed introduction later.

These contour integral-based algorithms share the same advantages that their efficiency does not depend too much on the distribution of eigenvalues. Hence they are well-suited for extracting all eigenvalues lying in the domain of interest. However, these algorithms require solving a series of linear systems, which could be a heavy workload, especially for large, sparse problems.

Our interest in this work is to solve these linear parameterized systems efficiently. If T is linear with respect to ξ , nested Krylov methods, such as multi-shift GMRES [13] or multi-shift QMRIDR [3] can be applied to reuse the Arnoldi basis [1]. Nevertheless, these techniques are designed for linear cases, and cannot be easily extended when the system is not linear with respect to the parameter. In this paper, we employ infinite GMRES [10, 21] to overcome these difficulties. In simple terms, infinite GMRES uses a companion linearization to transform the parameterized systems to a form that is linear with respect to the parameter, and then, use multi-shift GMRES to solve multiple systems together.

Briefly, our algorithm solves the NEP using contour integral-based algorithms where the Arnoldi process is employed to solve the linear systems efficiently. It not only avoids the drawback of poor parallelism in Krylov-based algorithms by achieving partial parallelism in matrix factorizations and solving least squares problems, but also significantly reduces the number of LU decompositions required for the contour integral. This makes our algorithm suitable for finding eigenvalues lying within certain contours for large, sparse problems.

The remainder of this paper is organized as follows. In Section 2, we provide a review of Beyn's algorithm and the infinite GMRES, along with some of their useful properties. In Section 3 we introduce the weighting technique and the two-level orthogonalization technique, and propose the structure of our algorithm. Implementation details, including the selection of some parameters and corresponding analysis, will be discussed in Section 4. Finally, numerical experiments will be presented in Section 5 to illustrate the efficiency of our algorithm.

2 Preliminaries

2.1 Nonlinear eigenvalue problems

In this work, we focus on nonlinear eigenvalue problems (1), where $T: \Omega \rightarrow \mathbb{C}^{n \times n}$ is holomorphic in the domain $\Omega \subset \mathbb{C}$ with sufficiently smooth boundary. Our goal is to find all eigenvalues $\lambda_1, \dots, \lambda_k$ lying in Ω , as well as their corresponding (right) eigenvectors v_1, \dots, v_k .

We say λ_j is a *simple* eigenvalue if $\ker(T(\lambda_j)) = \text{span}\{v_j\}$ while $T'(\lambda_j)v_j \notin \text{Range}(T(\lambda_j))$. For convenience, in the following, we always assume that there are finitely many eigenvalues lying in Ω , and all of them are simple.

2.2 Beyn's algorithm

Beyn's algorithm [6] can be regarded as a special case of the Sakurai–Sugiura algorithm where only the zeroth moment

$$\mathcal{M}_0 = \frac{1}{2\pi i} \int_{\partial\Omega} T(\xi)^{-1} Z \, d\xi$$

and first moment

$$\mathcal{M}_1 = \frac{1}{2\pi i} \int_{\partial\Omega} \xi T(\xi)^{-1} Z \, d\xi$$

are involved. For almost any $Z \in \mathbb{C}^{n \times k}$, if we perform the singular value decomposition (SVD), $\mathcal{M}_0 = V_0 \Sigma_0 W_0^*$, and construct $\tilde{\mathcal{M}}_1 = V_0^* \mathcal{M}_1 W_0 \Sigma_0^{-1}$, it can be proved that $\lambda_1, \dots, \lambda_k$ are exactly the eigenvalues of $\tilde{\mathcal{M}}_1$. Furthermore, we can diagonalize $\tilde{\mathcal{M}}_1$ as $\tilde{\mathcal{M}}_1 = S \Lambda S^{-1}$, where $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_k\}$ so that $V_0 S = [v_1, \dots, v_k]$ consists of the corresponding eigenvectors.

In Beyn's algorithm, numerical quadrature rules are used to approximate these moments. We assume the boundary of the region, $\partial\Omega$, can be parameterized as

$$\varphi \in C^1[0, 2\pi], \quad \varphi(\theta + 2\pi) = \varphi(\theta).$$

Taking N equidistant quadrature nodes as $\theta_j = 2j\pi/N$ (for $j = 0, \dots, N-1$) and applying the trapezoidal rule, we will obtain

$$\mathcal{M}_{0,N} = \frac{1}{iN} \sum_{j=0}^{N-1} \varphi'(\theta_j) T(\varphi(\theta_j))^{-1} Z, \quad \mathcal{M}_{1,N} = \frac{1}{iN} \sum_{j=0}^{N-1} \varphi(\theta_j) \varphi'(\theta_j) T(\varphi(\theta_j))^{-1} Z. \quad (3)$$

In this work, we always use an ellipse contour

$$\varphi(\theta) = c + a \cos(\theta) + ib \sin(\theta)$$

as well as the trapezoidal rule. This is a usual choice in many works [2, 15, 37], and has been proved to converge exponentially; see [34].

Though we take Beyn's algorithm as a framework in this paper, we remark here that the technique we shall propose is not dependent on a particular eigensolver, but can be applied to any contour integral-based algorithm where several moments need to be computed.

2.3 Infinite GMRES

To approximate the moments by (3), we have to evaluate $T(\xi_j)^{-1}Z$ for several quadrature nodes of the form $\xi_j = \varphi(\theta_j)$. With infinite GMRES (infGMRES) [10], we can solve them efficiently.

Suppose we need to compute $T(\xi)^{-1}z$ for several values of ξ around $\xi = 0$ all at once with infGMRES. We will firstly approximate T by a Taylor expansion as

$$T(\xi) \approx \sum_{j=0}^p \frac{\xi^j}{j!} T^{(j)}(0),$$

and linearize it to $\mathcal{L}_0 - \xi \mathcal{L}_1$, where

$$\mathcal{L}_0 = \begin{bmatrix} T(0) & \frac{T^{(1)}(0)}{1!} & \frac{T^{(2)}(0)}{2!} & \dots & \frac{T^{(p)}(0)}{p!} \\ & I & & & \\ & & I & & \\ & & & \ddots & \\ & & & & I \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} 0 & & & & \\ I & 0 & & & \\ & I & \ddots & & \\ & & \ddots & 0 & \\ & & & I & 0 \end{bmatrix}. \quad (4)$$

It can be proved that the first n elements of $(\mathcal{L}_0 - \xi \mathcal{L}_1)^{-1} \text{vec}_0(z)$ are exactly equal to

$$\left(\sum_{j=0}^p \frac{\xi^j}{j!} T^{(j)}(0) \right)^{-1} z, \quad (5)$$

where $\text{vec}_0(z) = [z^*, 0, \dots, 0]^*$. Under the assumption that the Taylor expansion is sufficiently accurate, in order to compute $T(\xi)^{-1}z$, we just compute $(\mathcal{L}_0 - \xi \mathcal{L}_1)^{-1} \text{vec}_0(z)$.

After extracting \mathcal{L}_0^{-1} , a multi-shift GMRES can be employed to solve linear systems $(I - \xi \mathcal{L}_1 \mathcal{L}_0^{-1})^{-1} \text{vec}_0(z)$ for several ξ 's easily. This is because once we obtain

$$\mathcal{L}_1 \mathcal{L}_0^{-1} \mathcal{U}_m = \mathcal{U}_{m+1} \underline{H}_m$$

Algorithm 1 infGMRES

Input: Maximum iteration m , the parameter-dependent matrix $T(\xi): \mathbb{C} \rightarrow \mathbb{C}^{n \times n}$, the right-hand side $z \in \mathbb{C}^n$ and the points to be solved ξ_j for $j = 0, \dots, N-1$

Output: Approximations $x_{0,j} \approx T(\xi_j)^{-1}z$ for $j = 0, \dots, N-1$

1: Linearize T to

$$\mathcal{L}_0 = \begin{bmatrix} T(0) & \frac{T^{(1)}(0)}{1!} & \dots & \frac{T^{(p)}(0)}{p!} \\ & I & & \\ & & \ddots & \\ & & & I \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} 0 & & & \\ I & 0 & & \\ & \ddots & \ddots & \\ & & I & 0 \end{bmatrix}, \quad p > m$$

2: Perform Arnoldi process on $(\mathcal{L}_1 \mathcal{L}_0^{-1}, \text{vec}_0(z))$ to obtain $\mathcal{L}_1 \mathcal{L}_0^{-1} \mathcal{U}_m = \mathcal{U}_{m+1} \underline{H}_m$

3: Set $y_j \leftarrow \arg \min_y \|(I_m - \xi_j \underline{H}_m)y - \|z\|e_1\|_2$ for $j = 0, \dots, N-1$

4: Set $x_{0,j} \leftarrow T(0)^{-1} [I \quad -T^{(1)}(0) \quad \dots \quad -T^{(p)}(0)/p!] \mathcal{U}_m y_j$ for $j = 0, \dots, N-1$

by the Arnoldi process, we also have

$$(I - \xi \mathcal{L}_1 \mathcal{L}_0^{-1}) \mathcal{U}_m = \mathcal{U}_{m+1} (\underline{I}_m - \xi \underline{H}_m),$$

which is exactly the Arnoldi decomposition of $(I - \xi \mathcal{L}_1 \mathcal{L}_0^{-1})$. Here, \mathcal{U}_m is the matrix whose columns are Arnoldi vectors, \underline{I}_m is the $m \times m$ identity with an extra zero row at the bottom, and $\underline{H}_m \in \mathbb{C}^{(m+1) \times m}$ is an upper Hessenberg matrix. Thus, as we already obtained the Hessenberg matrix \underline{H}_m by an Arnoldi process on $\mathcal{L}_1 \mathcal{L}_0^{-1}$, we need only to tackle a small-scale least squares problem

$$\min_y \|(\underline{I}_m - \xi \underline{H}_m)y - \|z\|e_1\|_2\|_2, \quad (6)$$

for any ξ . Since $\mathcal{U}_m y$ is the approximate solution of $(I - \xi \mathcal{L}_1 \mathcal{L}_0^{-1})^{-1} \text{vec}_0(z)$, we take $\mathcal{L}_0^{-1} \mathcal{U}_m y$ as the approximate solution of $(\mathcal{L}_0 - \xi \mathcal{L}_1)^{-1} \text{vec}_0(z)$. Then, the first n elements of $\mathcal{L}_0^{-1} \mathcal{U}_m y$ become the solution for that particular ξ . For reference, we list our infGMRES briefly in Algorithm 1.

From Algorithm 1, we know that applying \mathcal{L}_0^{-1} to several vectors involves essentially only one matrix factorization of $T(0)$. Hence, infGMRES is efficient when solving with many ξ 's. Furthermore, in [10], it is proved that under certain mild assumptions the action \mathcal{L}_0^{-1} can be applied approximately without affecting the convergence of the algorithm. This feature makes infGMRES even more attractive, if implemented properly.

Additionally, we note here that the order of the Taylor expansion, p , does not need to be determined in advance because at the j th iteration of GMRES only $T^{(s)}(0)$'s with $s \leq j$ are involved. If we process up to some certain iterations, say j , and find that the desired accuracy is not achieved, we can just provide the algorithm with a new $T^{(j+1)}(0)$ and continue. In practice, we can always assume $p = \infty$ and terminate once the accuracy is satisfactory. That is also why these algorithms are usually called infinite, or hold dynamic polynomial approximation properties [35]. For simplicity, we will always use a finite order linearization with $p > m$ in this paper. But readers should keep in mind that p and m are in fact infinite.

3 Proposed algorithm

With the methods we mentioned above, our idea is to solve linear systems in Beyn's method by infGMRES. A framework is summarized in Algorithm 2. In the following paragraphs, we will

Algorithm 2 Beyn's method with infGMRES

Input: The parameter-dependent matrix $T(\xi): \mathbb{C} \rightarrow \mathbb{C}^{n \times n}$, the initial guess $Z = [z_1, \dots, z_k] \in \mathbb{C}^{n \times k}$, the contour φ and quadrature nodes θ_j 's for $j = 0, \dots, N-1$

Output: Approximate eigenvalues Λ and eigenvectors V

- 1: **for** $s = 1, \dots, k$ **do**
 - 2: Use infGMRES to solve $T(\varphi(\theta_j))^{-1} z_s$ for $j = 0, \dots, N-1$ simultaneously
 - 3: **end for**
 - 4: Set $\mathcal{M}_{0,N} \leftarrow \frac{1}{iN} \sum_{j=0}^{N-1} \varphi'(\theta_j) T(\varphi(\theta_j))^{-1} Z$
 - 5: Set $\mathcal{M}_{1,N} \leftarrow \frac{1}{iN} \sum_{j=0}^{N-1} \varphi(\theta_j) \varphi'(\theta_j) T(\varphi(\theta_j))^{-1} Z$
 - 6: Singular value decomposition $\mathcal{M}_{0,N} = V_0 \Sigma_0 W_0^*$
 - 7: Set $\check{\mathcal{M}}_{1,N} \leftarrow V_0^* \mathcal{M}_{1,N} W_0 \Sigma_0^{-1}$
 - 8: Eigenvalue decomposition $\check{\mathcal{M}}_{1,N} = S \Lambda S^{-1}$
 - 9: Set $V \leftarrow V_0 S$
-

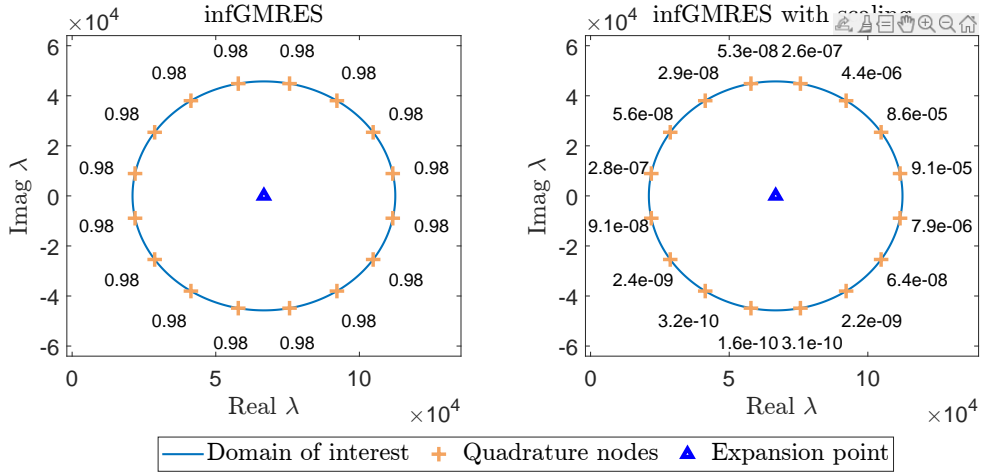


Figure 1: Running infinite GMRES with a single expansion point on the center of a circular contour (center at 66762 with a radius 45738) to solve 16 linear parameterized systems of the **gun** problem lying equidistantly on the contour. The value $m = 32$ is used in both figures. In practice, for approximating eigenpairs, we usually need these linear systems to be solved to an accuracy higher than 10^{-10} . The original infinite GMRES method (left) failed at all 16 points, whereas the infinite GMRES applied to the variable-substituted system $T(5a\tilde{\xi} + c)$ (right), although still not accurate enough, demonstrates a seemingly improved performance.

discuss how to employ infGMRES economically and efficiently in Step 2 of Algorithm 2.

One may consider expanding the Taylor series of T on the center of the contour to approximate all the linear systems on the contour at once. Unfortunately, the original implementation of infGMRES is usually not accurate enough, especially when the ellipse is relatively large or some eigenvalues lie close to the quadrature nodes [10]. To illustrate this, we take the **gun** problem from the NLEVP collection [5] as an example. We apply infGMRES on the center of a circular contour and solve the linear systems at each quadrature node; see Figure 1 (left). With 32 GMRES iterations, we obtained completely incorrect solutions.

Though this failure is partly caused by using too few iterations, it is not unavoidable within

the same number of iterations. If we instead run infGMRES on the variable-substituted system $T(5a\xi + c)$ to solve the same linear systems, the accuracy can be improved apparently within the same number of iterations; see Figure 1 (right). Here, we take $a = 45738$ as the radius of the circular contour and $c = 66762$ as the center of the contour. The reason why we take such a variable substitution will be explained in Section 4.1. For this section, we shall first provide a more general description of this technique to construct our algorithm.

The other problem is the redundant memory usage of \mathcal{U}_m . Storing the whole \mathcal{U}_m , as classical GMRES does, requires $\mathcal{O}(m^2n)$ memory. This will greatly limit the maximum iteration count GMRES can take, especially when n is very large. Following [23], we will apply a TOAR-like technique [25] to compress the memory footprint to $\mathcal{O}(mn + m^3)$. A generalized framework can be found in [36].

3.1 Weighting

In [10, Remark 6.2], it has been mentioned that an appropriate scaling can accelerate the convergence of infGMRES. But no proof or theoretical analysis is provided to justify the essential cause. In this subsection, we provide a more general implementation of the scaling. Further insights on this technique and its specific application for accelerating the convergence of infGMRES will be covered in Section 4.1.

When using infGMRES to solve $T(\xi_0)^{-1}z$ with a Taylor expansion centered at 0, we are actually solving

$$(\mathcal{L}_0 - \xi_0 \mathcal{L}_1)^{-1} \text{vec}_0(z) = \begin{bmatrix} T_0 & T_1 & \cdots & T_p \\ -\xi_0 I & I & & \\ & \ddots & \ddots & \\ & & -\xi_0 I & I \end{bmatrix}^{-1} \begin{bmatrix} z \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where for simplicity we denote $T_j = T^{(j)}(0)/j!$. To apply a scaling here, firstly notice that solving $T(\xi)^{-1}z$ at $\xi = \xi_0$ is equivalent to solving $\tilde{T}(\tilde{\xi})^{-1}z$ at $\tilde{\xi} = \xi_0/\rho$, where $\tilde{T}(\tilde{\xi}) = T(\rho\tilde{\xi})$. Repeating the same linearization process on $\tilde{T}(\tilde{\xi})$ yields alternative companion matrices of the form

$$\tilde{\mathcal{L}}_0 = \begin{bmatrix} T_0 & \rho T_1 & \rho^2 T_2 & \cdots & \rho^p T_p \\ & I & & & \\ & & I & & \\ & & & \ddots & \\ & & & & I \end{bmatrix}, \quad \tilde{\mathcal{L}}_1 = \mathcal{L}_1. \quad (7)$$

To solve the original system at $\tilde{\xi} = \xi_0/\rho$, we solve

$$\left(\tilde{\mathcal{L}}_0 - \frac{\xi_0}{\rho} \tilde{\mathcal{L}}_1\right)^{-1} \text{vec}_0(z) = \begin{bmatrix} T_0 & \rho T_1 & \cdots & \rho^p T_p \\ -\frac{\xi_0}{\rho} I & I & & \\ & \ddots & \ddots & \\ & & -\frac{\xi_0}{\rho} I & I \end{bmatrix}^{-1} \begin{bmatrix} z \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Comparing (4) and (7), we notice that

$$\tilde{\mathcal{L}}_0 = D_\rho^{-1} \mathcal{L}_0 D_\rho, \quad \frac{1}{\rho} \tilde{\mathcal{L}}_1 = D_\rho^{-1} \mathcal{L}_1 D_\rho, \quad \tilde{\mathcal{L}}_0 - \frac{\xi_0}{\rho} \tilde{\mathcal{L}}_1 = D_\rho^{-1} (\mathcal{L}_0 - \xi_0 \mathcal{L}_1) D_\rho,$$

where

$$D_\rho = \begin{bmatrix} I & & & \\ & \rho I & & \\ & & \ddots & \\ & & & \rho^p I \end{bmatrix}.$$

This reminds us the balancing techniques [9] for eigenvalue problems, where we may take

$$D = \begin{bmatrix} d_0 I & & & \\ & d_1 I & & \\ & & \ddots & \\ & & & d_p I \end{bmatrix} \quad (8)$$

with arbitrary d_j 's. In fact, we can also obtain the approximate solution (5) by solving $(D^{-1}(\mathcal{L}_0 - \xi \mathcal{L}_1)D)^{-1} \text{vec}_0(z)$; see Lemma 1.

Lemma 1. *Suppose $d_j \in \mathbb{C} \setminus \{0\}$ and $T_j \in \mathbb{C}^{n \times n}$ for $j = 0, \dots, p$, and*

$$D = \begin{bmatrix} d_0 I & & & \\ & d_1 I & & \\ & & \ddots & \\ & & & d_p I \end{bmatrix},$$

$$\mathcal{L}_0 = \begin{bmatrix} T_0 & T_1 & T_2 & \cdots & T_p \\ & I & & & \\ & & I & & \\ & & & \ddots & \\ & & & & I \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} 0 & & & & \\ I & 0 & & & \\ & I & \ddots & & \\ & & \ddots & 0 & \\ & & & I & 0 \end{bmatrix}.$$

Then, for any scalar $\xi \in \mathbb{C}$ and vector $z \in \mathbb{C}^n$,

$$(D^{-1}(\mathcal{L}_0 - \xi \mathcal{L}_1)D)^{-1} \text{vec}_0(z) = \begin{bmatrix} (\sum_{j=0}^p \xi^j T_j)^{-1} z \\ * \\ \vdots \\ * \end{bmatrix}.$$

Proof. The proof of Lemma 1 follows from [10, Theorem 3.1]. We do not repeat it here. \square

Lemma 1 indicates that we can perform infGMRES on any balanced companion linearization $(D^{-1}\mathcal{L}_0 D, D^{-1}\mathcal{L}_1 D)$, and a scaling can be regarded as a special case where $D = D_\rho$. Additionally, since $D^{-1}\mathcal{L}_j D = (d_0^{-1}D)^{-1}\mathcal{L}_j(d_0^{-1}D)$, we can always assume $d_0 = 1$ without loss of generality.

3.2 Compact representation of the Arnoldi basis

TOAR [25] is a memory-efficient algorithm for the Arnoldi process of quadratic eigenvalue problems (QEP). The most remarkable insight of TOAR is that the Arnoldi vectors of a companion linearization of the QEP can be represented in terms of a common basis for both upper and lower halves. In [36], it is proved that similar techniques can be applied to more general cases. In this

work, we apply a similar two-level orthogonalization for infGMRES to reduce the increasingly larger memory usage.

Firstly, partition the Arnoldi vectors \mathcal{U}_m by row blocks to

$$\mathcal{U}_m = \begin{bmatrix} U_{m,0} \\ \vdots \\ U_{m,p} \end{bmatrix}, \quad U_{m,j} \in \mathbb{C}^{n \times m}, \quad j = 0, \dots, p.$$

Then, there is a $Q_m \in \mathbb{C}^{m \times n}$, $Q_m^* Q_m = I$, such that $U_{m,j} = Q_m \check{U}_{m,j}$ for some $\check{U}_{m,j} \in \mathbb{C}^{m \times m}$ (for $j = 0, \dots, p$). Furthermore, the Arnoldi process can be rearranged into

$$\mathcal{L}_1 \mathcal{L}_0^{-1} \mathcal{U}_m = \mathcal{L}_1 \mathcal{L}_0^{-1} \begin{bmatrix} Q_m \check{U}_{m,0} \\ \vdots \\ Q_m \check{U}_{m,p} \end{bmatrix} = \begin{bmatrix} Q_m \check{U}_{m,0} & [Q_m, q_{m+1}] \check{u}_{m+1,0} \\ \vdots & \vdots \\ Q_m \check{U}_{m,p} & [Q_m, q_{m+1}] \check{u}_{m+1,p} \end{bmatrix} \underline{H}_m = \mathcal{U}_{m+1} \underline{H}_m,$$

where the columns of both $[Q_m, q_{m+1}]$ and

$$\begin{bmatrix} Q_m \check{U}_{m,0} & [Q_m, q_{m+1}] \check{u}_{m+1,0} \\ \vdots & \vdots \\ Q_m \check{U}_{m,p} & [Q_m, q_{m+1}] \check{u}_{m+1,p} \end{bmatrix}$$

are orthonormal. Thus, by setting $Q_{m+1} = [Q_m, q_{m+1}]$ and

$$\check{U}_{m+1,j} = \begin{bmatrix} \check{U}_{m,j} & \vdots \\ 0 & \check{u}_{m+1,j-1} \end{bmatrix},$$

the Arnoldi vectors \mathcal{U}_{m+1} share the same structure

$$\mathcal{U}_{m+1} = \begin{bmatrix} Q_{m+1} \check{U}_{m+1,0} \\ \vdots \\ Q_{m+1} \check{U}_{m+1,p} \end{bmatrix}.$$

Therefore, we only need to store Q_m and $\check{U}_{m,j}$ in memory for the Arnoldi process. Additionally, thanks to the pattern of $\text{vec}_0(z)$, we have $\check{U}_{m,j} = 0$ for $j \geq m$. Overall, the memory usage will be $\mathcal{O}(mn + m^3)$. However, since \mathcal{U}_m is not explicitly formed, additional care should be taken when orthogonalizing, using the so called two-level orthogonalization. Combined with weighting, we list our weighted two-level orthogonalization infGMRES in Algorithm 3 for reference.

Remark 1. For simplicity and consistency, we initialize $\check{U}_{j,s}$'s for $s = 0, \dots, p$ from the beginning. However, readers should remember that $\check{U}_{j,s} = 0$ for $s \geq j$. Hence, $\check{U}_{j,s}$'s are neither computed nor stored in memory.

Remark 2. Notice that in Step 4 of Algorithm 1, we have to apply

$$Q_{\log} = T(0)^{-1} \begin{bmatrix} I & -T^{(1)}(0) & \dots & -T^{(p)}(0)/p! \end{bmatrix} \mathcal{U}_m$$

on each y_j to recover the approximate solution $x_{0,j}$, which may involve extra workload. Fortunately, as mentioned in [10], we can store Q_{\log} column by column during the infGMRES process, to make this cheaper; see Step 6 and Step 19 in Algorithm 3.

Algorithm 3 Weighted two-level orthogonalization infGMRES

Input: Maximum iterations m , the matrices $T_j \in \mathbb{C}^{n \times n}$ for $j = 0, \dots, p$, $p > m$, the initial guess $z \in \mathbb{C}^n$ and the weights d_1, \dots, d_p

Output: Function $f_x(\xi)$ returning the approximate solution of $T(\xi)^{-1}z$

```

1: function  $[f_x] = \text{WTinfGMRES}(T_0, \dots, T_p, d_1, \dots, d_p, z)$ 
2:  $Q_1 \leftarrow z/\|z\|_2$ ,  $H_0 \leftarrow [\ ]$ ,  $Q_{\log} \leftarrow [\ ]$ 
3:  $\check{U}_{1,0} \leftarrow 1$ ,  $\check{u}_{1,0} \leftarrow 1$ ,  $\check{U}_{1,s} \leftarrow 0$ ,  $\check{u}_{1,s} \leftarrow 0$ ,  $s = 1, \dots, p$ 
4: for  $j = 1, \dots, m$  do
5:    $q_{j+1} \leftarrow d_1^{-1} T_0^{-1} (Q_j \check{u}_{j,0} - \sum_{s=1}^p d_s T_s Q_j \check{u}_{j,s})$ 
6:    $Q_{\log} \leftarrow [Q_{\log}, d_1 q_{j+1}]$                                      % Record  $\mathcal{L}_0^{-1} Q_j$ 
7:    $l_j \leftarrow Q_j^* q_{j+1}$                                            % First level orthogonalization
8:    $q_{j+1} \leftarrow q_{j+1} - Q_j l_j$ 
9:    $\alpha_j \leftarrow \|q_{j+1}\|_2$ 
10:   $q_{j+1} \leftarrow q_{j+1}/\alpha_j$ 
11:   $Q_{j+1} \leftarrow [Q_j, q_{j+1}]$ 
12:   $h_j = \check{U}_{j,1}^* l_j + \sum_{s=2}^p (d_{s-1}/d_s) \check{U}_{j,s}^* \check{u}_{j,s-1}$            % Second level orthogonalization
13:   $\check{u}_{j+1,0} \leftarrow 0$ ,  $\check{u}_{j+1,1} \leftarrow \begin{bmatrix} l_j - \check{U}_{j,1}^* h_j \\ \alpha_j \end{bmatrix}$ ,  $\check{u}_{j+1,s} \leftarrow \begin{bmatrix} \check{u}_{j,s-1} - \check{U}_{j,s}^* h_j \\ 0 \end{bmatrix}$ ,  $s = 2, \dots, p$ 
14:   $\beta_j \leftarrow \sqrt{\sum_{s=0}^j \check{u}_{j+1,s}^* \check{u}_{j+1,s}}$ 
15:   $\check{u}_{j+1,s} \leftarrow \check{u}_{j+1,s}/\beta_j$ ,  $s = 0, \dots, p$ 
16:   $H_j \leftarrow \begin{bmatrix} H_{j-1} & h_j \\ 0 & \beta_j \end{bmatrix}$                                      % Update  $U$  and  $H$ 
17:   $\check{U}_{j+1,s} \leftarrow \begin{bmatrix} \check{U}_{j,s} & \check{u}_{j+1,s} \\ 0 & \end{bmatrix}$ ,  $s = 0, \dots, p$ 
18: end for
19:  $f_x(\xi) = \|z\|_2 Q_{\log} (I - \xi H_m)^\dagger e_1$ 
20: end function

```

Table 1: Application of Algorithm 3 with/without reorthogonalization to all 9 test examples from Table 2. The last two columns show the orthogonality of Q_m and \check{U}_m . The test is performed on all the expansion points, and only the worst values are shown here.

	Problem	$\ Q_m^* Q_m - I\ _2$	$\ \check{U}_m^* \check{U}_m - I\ _2$
w/o reorthog.	spring	2.3×10^1	6.3×10^{-7}
	acoustic_wave_2d	2.1×10^{-8}	5.8×10^{-11}
	butterfly	7.8×10^{-4}	1.4×10^{-3}
	loaded_string	1.6×10^1	1.5×10^{-13}
	photonics	1.9×10^1	7.1×10^{-11}
	railtrack2_rep	1.7×10^1	4.0×10^{-14}
	hadelar	1.2×10^{-14}	2.0×10^{-14}
	gun	1.8×10^1	4.0×10^{-14}
	canyon_particle	1.9×10^1	8.1×10^{-6}
with reorthog.	spring	1.1×10^{-15}	5.0×10^{-16}
	acoustic_wave_2d	6.8×10^{-16}	5.0×10^{-16}
	butterfly	6.3×10^{-16}	6.7×10^{-16}
	loaded_string	1.1×10^{-15}	7.0×10^{-16}
	photonics	7.7×10^{-16}	8.1×10^{-16}
	railtrack2_rep	5.7×10^{-16}	4.3×10^{-16}
	hadelar	9.7×10^{-16}	3.3×10^{-16}
	gun	9.1×10^{-16}	7.4×10^{-16}
	canyon_particle	9.6×10^{-16}	6.7×10^{-16}

Remark 3. We remark here that even with the TOAR-like technique, Q_m and

$$\check{U}_m = \begin{bmatrix} \check{U}_{m,0} \\ \vdots \\ \check{U}_{m,p} \end{bmatrix}$$

may not be fully orthogonal in finite precision arithmetic. As we show in Table 1, Algorithm 3 loses the orthogonality of Q_m in almost every test example and sometimes loses the orthogonality of \check{U}_m as well. However, the loss of orthogonality does not always destroy the convergence of GMRES; see [16]. In fact, we can perform a reorthogonalization on Q_m and \check{U}_m to make them orthogonal; see Table 1. Nevertheless, the accuracy of infGMRES will not increase, even though both Q_m and \check{U}_m are numerically orthogonal. Therefore, we will not use reorthogonalization in our numerical experiments.

4 Implementation details of the algorithm

While our modified infGMRES has been fully described in previous sections, the implementation process involves determining numerous parameters, including the scalars d_j , the locations for applying Taylor expansion ξ_j , and others. In this section, we will reveal the connections among these parameters and the convergence of infGMRES, offering practical guidance on their selection, and hence finishing the last details needed in Algorithm 2.

4.1 Scaling is essentially a weighting strategy on GMRES

In Section 3.1, we have already showed that a scaling can be generalized into a balanced companion linearization, say $D^{-1}\mathcal{L}_0D$ and $D^{-1}\mathcal{L}_1D$. Here, we declare that this technique is essentially a weighting strategy for the least squares problems within GMRES and, if used appropriately, can help to find a more accurate solution in a certain search space.

To see this, we have to look into the process of GMRES. First notice that if the Krylov subspace $\mathcal{K}_m(I - \xi_0\mathcal{L}_1\mathcal{L}_0^{-1}, \text{vec}_0(z))$ is spanned by \mathcal{U}_m , by induction, we can prove that $\mathcal{K}_m(D^{-1}(I - \xi_0\mathcal{L}_1\mathcal{L}_0^{-1})D, \text{vec}_0(z))$ is spanned by $D^{-1}\mathcal{U}_m$. Thus, at the m th iteration, infGMRES will give the solution by

$$\begin{aligned} y_* &= \arg \min_y \|D^{-1}(I - \xi_0\mathcal{L}_1\mathcal{L}_0^{-1})DD^{-1}\mathcal{U}_m y - \text{vec}_0(z)\|_2 \\ &= \arg \min_y \|D^{-1}((I - \xi_0\mathcal{L}_1\mathcal{L}_0^{-1})\mathcal{U}_m y - \text{vec}_0(z))\|_2, \end{aligned} \quad (9)$$

where the last equality follows because $d_0 = 1$.

Equation (9) provides us with several insights. Firstly, whichever the weighting matrix D is chosen, the final approximation of $T(\xi_0)^{-1}z$ will be selected from the same search space. That is because the approximate solution will be the first n elements of $D^{-1}\mathcal{L}_0^{-1}\mathcal{U}_m y_*$ and $d_0 = 1$. The role D plays in infGMRES is actually a weighting matrix in the least squares problems (9). We can choose D appropriately to guide GMRES to pick a more accurate solution from \mathcal{U}_m .

One may feel confused because GMRES already provides the best solution that minimizes the residual norm. However, what we really care about is not the residual of GMRES (9), but

$$\|r_N\|_2 = \|T(\xi_0)x_0 - z\|_2,$$

or

$$\|r_P\|_2 = \left\| \sum_{j=0}^p \xi_0^j T_j x_0 - z \right\|_2, \quad (10)$$

where x_0 is the approximate solution of $T(\xi_0)^{-1}z$. In an ideal scenario, we choose the weighting matrix D so that (9) returns the best solution in the sense of (10). Therefore, it is crucial to reveal the relationship between (9) and (10).

Lemma 2. Suppose z , \mathcal{U}_m , \mathcal{L}_0 , \mathcal{L}_1 and T_j (for $j = 0, \dots, p$) are all defined as before. The vector $y_* \in \mathbb{C}^m$ is the solution of (9) for some weighting matrix D . Then, the polynomial-wise residual $r_P = \sum_{j=0}^p \xi_0^j T_j x_0 - z$ can be represented as

$$r_P = \left[I, -\sum_{j=1}^p \xi_0^{j-1} T_j, -\sum_{j=2}^p \xi_0^{j-2} T_j, \dots, -T_p \right] \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_p \end{bmatrix}, \quad (11)$$

where

$$\begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_p \end{bmatrix} = (I - \xi_0\mathcal{L}_1\mathcal{L}_0^{-1})\mathcal{U}_m y_* - \text{vec}_0(z). \quad (12)$$

Proof. We know that if y_* is the solution of (9), infGMRES will give $x_* = D^{-1}\mathcal{L}_0^{-1}\mathcal{U}_m y_*$ as the approximate solution of $(D^{-1}(\mathcal{L}_0 - \xi_0\mathcal{L}_1)D)^{-1}\text{vec}_0(z)$. Then, by Lemma 1, the first n elements

of x_* , denoted by $x_0 \in \mathbb{C}^n$, will be taken as the approximate solution of $T(\xi_0)^{-1}z$. In other words, we have

$$D^{-1}\mathcal{L}_0^{-1}\mathcal{U}_m y_* = x_* = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix},$$

where x_0 here is same as the one in (10).

Therefore, we can represent (12) in terms of x_* instead of y_* :

$$\begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_p \end{bmatrix} = (I - \xi_0 \mathcal{L}_1 \mathcal{L}_0^{-1})\mathcal{U}_m y_* - \text{vec}_0(z) = (\mathcal{L}_0 - \xi_0 \mathcal{L}_1)Dx_* - \text{vec}_0(z). \quad (13)$$

Just listing out the equations in (13)

$$\begin{aligned} d_0 T_0 x_0 + d_1 T_1 x_1 + \cdots + d_p T_p x_p &= d_0 \tilde{r}_0, \\ -\xi_0 d_0 x_0 + d_1 x_1 &= d_1 \tilde{r}_1, \\ &\vdots \\ -\xi_0 d_{p-1} x_{p-1} + d_p x_p &= d_p \tilde{r}_p, \end{aligned}$$

and substituting all other equations into the first one yields (11). \square

Remember that the solution y_* minimizes (9), so that r_j 's minimize

$$\|D^{-1}((I - \xi_0 \mathcal{L}_1 \mathcal{L}_0^{-1})\mathcal{U}_m y - \text{vec}_0(z))\|_2 = \left\| D^{-1} \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_p \end{bmatrix} \right\|_2 = \sqrt{\sum_{j=0}^p \frac{\|r_j\|_2^2}{d_j^2}},$$

which means we can adjust the magnitude of r_j by setting d_j properly. If we set d_j to be larger, then, $\|r_j\|_2$ must be more modest.

On the other hand, we know from Lemma 2 that

$$\|r_P\| \leq \|r_0\|_2 + \left\| \sum_{j=1}^p \xi_0^{j-1} T_j \right\|_2 \|r_1\|_2 + \cdots + \|T_p\|_2 \|r_p\|_2.$$

Therefore, intuitively, if we make $\|r_j\|_2$ relatively small for larger $\|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2$, and relatively large for smaller $\|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2$, overall, $\|r_P\|$ could be modest.

With this insight, we may set the weights as $d_0 = 1$ and $d_s = \|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2^{-1}$ for $s = 1, \dots, p$. However, there are cases where $\|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2 \ll 1$ or $\|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2 \gg 1$, which will make D extremely ill-conditioned. Thus, in practice, we prefer to balance d_j by

$$d_j \leftarrow d_j / \left(\frac{d_2^2}{d_3} \right), \quad j > 0.$$

Consequently, in our algorithm, the weights d_j 's will be taken as

$$d_0 = 1, \quad d_s = \frac{\gamma}{\|\sum_{j=s}^p \xi_0^{j-s} T_j\|_2}, \quad \gamma = \frac{\|\sum_{j=2}^p \xi_0^{j-2} T_j\|_2^2}{\|\sum_{j=3}^p \xi_0^{j-3} T_j\|_2}, \quad s = 1, \dots, p. \quad (14)$$

Remark 4 (How to determine ξ_0). *Since several linear systems will share the same expansion points, we cannot set a specified D for each linear system, so that we can neither set a specified ξ_0 . In practice, we prefer to use a ξ_0 that is a little bit larger than the radius of a circle that encloses the integration points we want to associate to it. For example, if we expand at $\xi = 0$ and want to solve linear systems within $|\xi| < \nu$, then, $\xi_0 = 2\nu$ will be a satisfying choice.*

Remark 5. *Our analysis does not depend on specific styles of the companion linearizations. For example, if we use the classical companion linearization*

$$\mathcal{L}_0 = \begin{bmatrix} A_1 & A_2 & \cdots & A_p \\ I & & & \\ & \ddots & & \\ & & I & \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} A_0 & & & \\ & -I & & \\ & & \ddots & \\ & & & -I \end{bmatrix},$$

it can also be proved that

$$D_\rho^{-1}(\xi \mathcal{L}_1^{-1} \mathcal{L}_0 - I) D_\rho = \frac{\xi}{\rho} \tilde{\mathcal{L}}_1^{-1} \tilde{\mathcal{L}}_0 - I,$$

where $\tilde{\mathcal{L}}_0$ and $\tilde{\mathcal{L}}_1$ are the companion linearization matrices corresponding to the scaled system \tilde{T} . Other analyses on D will follow.

4.2 On polynomial eigenvalue problems

Since infGMRES is originally designed for non-polynomial eigenvalue problems, additional care must be taken when dealing with a PEP, or in other words when

$$T(\xi) = T_0 + \xi T_1 + \cdots + \xi^g T_g.$$

In these cases, it is not necessary for us to use infGMRES. On the contrary, just linearizing it to

$$\check{\mathcal{L}}_0 = \begin{bmatrix} T_0 & T_1 & T_2 & \cdots & T_g \\ & I & & & \\ & & I & & \\ & & & \ddots & \\ & & & & I \end{bmatrix}, \quad \check{\mathcal{L}}_1 = \begin{bmatrix} 0 & & & & \\ I & 0 & & & \\ & I & \ddots & & \\ & & \ddots & 0 & \\ & & & I & 0 \end{bmatrix}, \quad (15)$$

and using multi-shift GMRES to solve $(\check{\mathcal{L}}_0 - \xi \check{\mathcal{L}}_1)^{-1} \text{vec}_0(z)$ is sufficient to solve the problem.

However, if one insists on employing infGMRES, the linearization becomes

$$\mathcal{L}_0 = \begin{bmatrix} T_0 & T_1 & \cdots & T_g & 0 & \cdots \\ & I & & & & \\ & & \ddots & & & \\ & & & I & & \\ & & & & I & \\ & & & & & \ddots \end{bmatrix}, \quad \mathcal{L}_1 = \begin{bmatrix} 0 & & & & & \\ I & 0 & & & & \\ & I & \ddots & & & \\ & & \ddots & 0 & & \\ & & & I & 0 & \\ & & & & \ddots & \ddots \end{bmatrix}, \quad (16)$$

where we use 0 to fill the position of T_j , $j > g$ because $T^{(j)}(\xi) = 0$, $j > g$. It is interesting to note that $(\mathcal{L}_0 - \xi \mathcal{L}_1)^{-1} \text{vec}_0(z)$ also gives the true solution of $T(\xi)^{-1}z$ on the first n elements. One question that naturally comes to one's mind is: (15) or (16), which linearization is better?

From the point of view of computational cost and memory usage, in the j th iteration of (15), the cost of orthogonalization will be $\mathcal{O}(jn + j^2g)$ and the total memory usage is also $\mathcal{O}(jn + j^2g)$. As for (16), they are $\mathcal{O}(jn + j^3)$. For general cases $m > g$, so that (15) is cheaper. However, since $g < m \ll n$, there is not much difference.

Things become interesting if we look from the perspective of convergence rate. Equation (15) can actually be regarded as (16) with the weighting we mentioned in Section 4.1. Remember that, with our weighting strategy, the weighting matrix D will be

$$D = \begin{bmatrix} d_0 I & & & & & \\ & d_1 I & & & & \\ & & \ddots & & & \\ & & & d_g I & & \\ & & & & \infty I & \\ & & & & & \ddots \end{bmatrix}.$$

Then, if we regard ∞ as a very large number that can be used in arithmetic computation, we will have the weighted system

$$D^{-1}\mathcal{L}_0 D = \begin{bmatrix} T_0 & T_1 & \cdots & T_g & 0 & \cdots \\ & I & & & & \\ & & \ddots & & & \\ & & & I & & \\ & & & & I & \\ & & & & & \ddots \end{bmatrix},$$

$$D^{-1}\mathcal{L}_1 D = \begin{bmatrix} 0 & & & & & \\ \frac{d_0}{d_1} I & 0 & & & & \\ & \ddots & \ddots & & & \\ & & \ddots & \frac{d_{g-1}}{d_g} I & 0 & \\ & & & 0 & 0 & \\ & & & & I & 0 \\ & & & & & \ddots \end{bmatrix}.$$

Notice that $D^{-1}\mathcal{L}_1\mathcal{L}_0^{-1}D$ is a diagonal block matrix now. Since the right-hand side $\text{vec}_0(z)$ has non-zero elements only on the first n dimensions, solving $(D^{-1}(I - \xi\mathcal{L}_1\mathcal{L}_0^{-1})D)^{-1}\text{vec}_0(z)$ and $(D_g^{-1}(I - \xi\check{\mathcal{L}}_1\check{\mathcal{L}}_0^{-1})D_g)^{-1}\text{vec}_0(z)$ with GMRES will result in exactly the same process, where $D_g = \text{diag}\{d_0 I, \dots, d_g I\}$ is the truncated D .

Since both the cost and the convergence of the two methods are similar, we also use infGMRES in Section 5 for PEP to keep results uniform.

4.3 How to choose expansion points

Even with the weighting strategy, it may still take too many iterations for infGMRES to converge, especially for quadrature nodes that are far from expansion points. Therefore, it is sensible to use more than one expansion point in practice. However, a good choice of the expansion points depends on the singularities, eigenvalues and many other things. Therefore, adaptively selecting the expansion points is usually impractical, and it is not the focus of interest in this paper. In

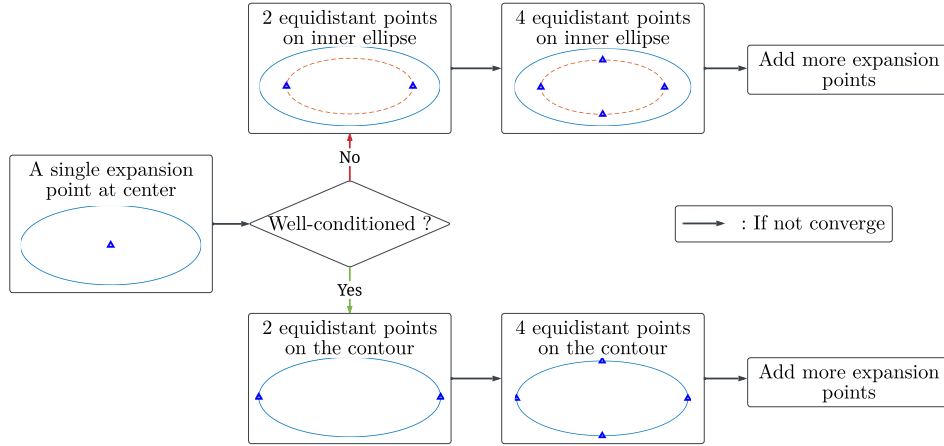


Figure 2: A brief guideline on choosing expansion points: If a single point at the center does not work, choose expansion points equidistantly on the contour or on an inner ellipse, based on the condition number of the problem. Continue doubling the number of expansion points till the accuracy is satisfactory.

this subsection, we provide a flowchart for users to heuristically determine the distribution of expansion points; see Figure 2.

At this point, we can fully describe the computation needed for Step 2 of Algorithm 2; see Algorithm 4.

Algorithm 4 Step 2 of Algorithm 2

Input: The parameter-dependent matrix $T(\xi): \mathbb{C} \rightarrow \mathbb{C}^{n \times n}$, the right-hand side $z \in \mathbb{C}^n$, the contour φ and quadrature nodes θ_j 's for $j = 0, \dots, N-1$

Output: Approximations $x_{0,j} \approx T(\varphi(\theta_j))^{-1} z$ for $j = 0, \dots, N-1$

- 1: Determine expansion points $\xi_0, \dots, \xi_{n_{ep}}$ as in Figure 2
 - 2: **for** $j = 0, \dots, n_{ep}$ **do**
 - 3: Set $T_{s,j} \leftarrow T^{(s)}(\xi_j)/s!$ for $s = 0, \dots, p$
 - 4: Compute weights $d_{s,j}$'s as in Section 3.1
 - 5: Set $f_{x,j}(\cdot) \leftarrow \text{WTinfGMRES}(T_{0,j}, \dots, T_{p,j}, d_{0,j}, \dots, d_{p,j}, z)$
 - 6: **end for**
 - 7: **for** $j = 0, \dots, N-1$ **do**
 - 8: Find $\xi_s \in \{\xi_0, \dots, \xi_{n_{ep}}\}$ closest to $\varphi(\theta_j)$
 - 9: Set $x_{0,j} \leftarrow f_{x,s}(\varphi(\theta_j))$
 - 10: **end for**
-

5 Numerical experiments

In this section we present experimental results of Algorithms 3. All numerical experiments were performed using MATLAB R2022b on a Linux server with two 16-core Intel Xeon Gold 6226R 2.90 GHz CPUs and 1024 GB main memory.

Table 2: Information of test problems. Here, n is the size of the problem and k is the number of the eigenvalues to be computed. The number of quadrature nodes, expansion points are represented by N and n_{ep} , respectively. The times consumed by Beyn’s method with MATLAB backslash, t_s , and with infGMRES t_{iG} are also listed in the last columns.

Problem	Type	n	k	N	n_{ep}	$t_s(\text{s})$	$t_{\text{iG}}(\text{s})$	$(t_s - t_{\text{iG}})/t_s$
spring	QEP	3000	32	1024	6	3.472	15.72	−353%
acoustic_wave_2d	QEP	9900	10	512	5	28.14	9.475	66%
butterfly	PEP	5000	9	512	9	11.54	8.453	27%
loaded_string	REP	20000	10	128	4	1.07	9.805	−816%
photonics	REP	20363	16	3060	18	600	209.7	65%
railtrack2_rep	REP	35955	2	128	1	533.4	71.95	87%
hadelers	NEP	5000	13	32	1	40.2	46.6	−16%
gun	NEP	9956	21	1024	10	501.2	148.9	70%
canyon_particle	NEP	16281	5	256	4	40.94	7.652	81%

5.1 Experiment settings

Most of our test examples are chosen from the NLEVP collections [5] except **photonics**, which is similar to the one described in [11], but with a more general model for the permittivity as in [14]. We set the maximum number of the outer iterations of infGMRES as 32 for all these examples, while the number of quadrature nodes and expansion points varies from case to case. Details regarding these examples, including their respective types, the problem size n , the number of quadrature nodes N , and the number of expansion points n_{ep} , are listed in Table 2. For the exact distribution of these points or the contour, readers can check Figure 3.

For computed approximate nonlinear eigenpair $(\hat{\lambda}_j, \hat{v}_j)$, the convergence criterion is

$$\|T(\hat{\lambda}_j)\hat{v}_j\|_2 \leq \text{tol} \cdot \|T(\hat{\lambda}_j)\|_2 \|\hat{v}_j\|_2, \quad (17)$$

where **tol** is the user-specified tolerance. Unless otherwise stated, all the test results presented in this section achieved an accuracy of $\text{tol} = 10^{-12}$.

As we mentioned in Section 2.3, it is possible to apply the action T_0^{-1} in Algorithm 3 in an inexact way, e.g., using algebraic multigrid or GMRES, to make the algorithm even faster. However, to emphasize the effect of our algorithm, we use LU decomposition to solve all the linear systems T_0^{-1} exactly in our numerical experiments. Readers should keep in mind that, in practice, inexact linear solvers can be implemented here for further acceleration.

5.2 Overall performance on the test examples

To illustrate the efficiency of our algorithm, we solve the linear systems by MATLAB backslash (`\`) as comparative experiments. The times consumed by Beyn’s method with MATLAB backslash (t_s) and with infGMRES (t_{iG}) are listed, respectively, in the last columns of Table 2. Except for the **spring**, **loaded_string** and **hadelers**, our algorithm achieved a speedup of at least 27%. The acceleration rate increases when the problems size becomes larger, where our algorithm can benefit from taking fewer matrix decompositions and reach a speedup over 80%.

However, there are two scenarios where the advantages of our algorithm are less evident. In the cases of **spring** and **loaded_string**, MATLAB implements specialized optimizations for tridiagonal matrices, making it several times faster than the general MATLAB backslash operation. On the other hand, in the case of **hadelers**, two out of its three component matrices

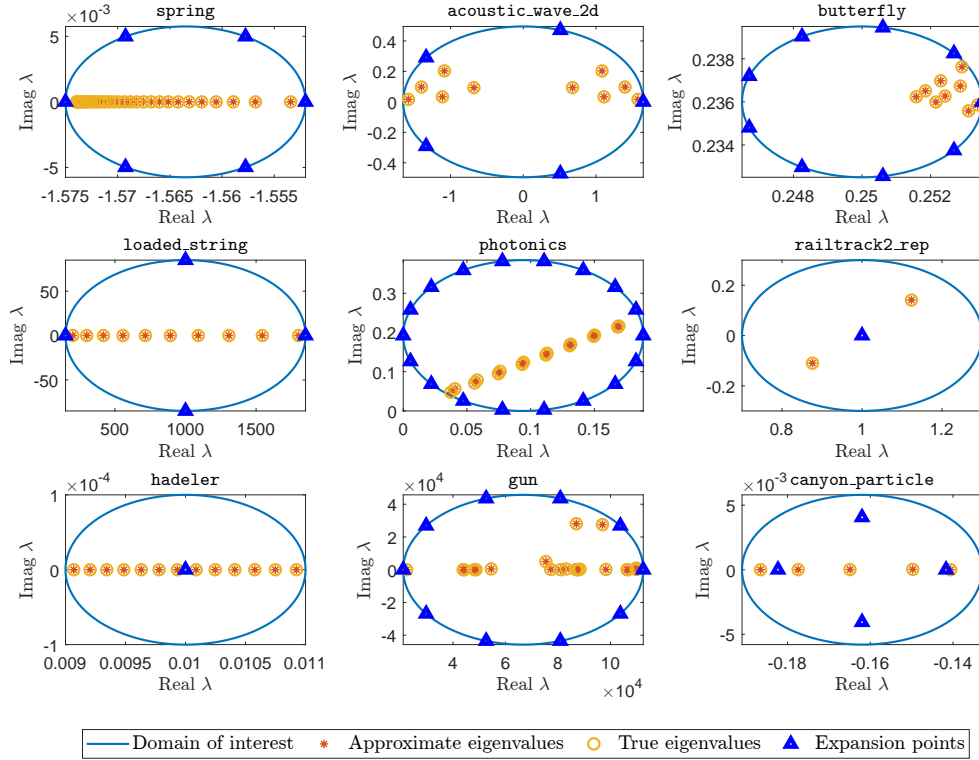


Figure 3: Interest field, expansion points and eigenvalues of different test problems.

are dense. Generally, iterative methods are not as efficient for dense matrices, leading to reduced effectiveness in these specific instances.

To obtain a more detailed analysis, we have extracted the time proportions of each operation, as illustrated in Figure 4. It can be found that a significant portion of time is dedicated to the Arnoldi process (matrix–vector multiplications and orthogonalization) in our algorithm. Even in the challenging problem **photonics**, where over 3000 quadrature nodes are needed to be solved, solving least squares problems consume comparatively less time. This implies that our algorithm proves especially advantageous in cases involving a large number of quadrature nodes.

One thing to note from Figure 4 is that the time consumed by LU decomposition is relatively higher for **railtrack2_rep**. That is because the size of this problem is obviously larger, bringing about more expensive factorizations. However, it is actually a good news for our algorithm, which means that when the problem size become larger, our algorithm can benefit more from solving these factorizations in parallel.

5.3 Comparison on different weighting strategies

In Section 3.1, we introduce a weighting strategy for infGMRES. Here, we illustrate its efficiency under different circumstances with numerical experiments. The **gun** problem is taken as the test problem. We compute the Taylor expansion on the leftmost and rightmost points of the contour, respectively, to approximate the linear systems corresponding to quadrature nodes around. The difference between these two points is that there are singularities close to the left side one, leading to a more ill-conditioned problem. The full picture can be found in Figure 5 (a), where

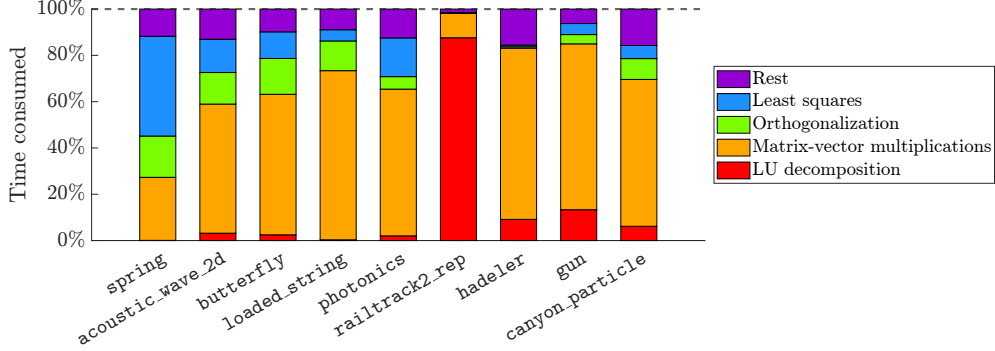


Figure 4: The proportional chart of Beyn’s methods with infinite GMRES. The times consumed by different operations are illustrated.

we illustrate the corresponding relationships between expansion points and quadrature nodes. The number of outer iterations of infGMRES is fixed to 32 in all experiments. The residuals obtained with or without weighting can be found in Figures 5 (b), (c), (f), (g). Our weighting strategy can bring significant boost on the accuracy in both cases.

As a comparison, we refer to the scaling strategy from [4, Section 6]. In that algorithm, a scalar

$$\rho = \left(\frac{\|T_0\|}{\|T_p\|} \right)^{1/p} \quad (18)$$

is used to make the norm of $\rho^j T_j$ as similar as possible. Their consideration is that solving a PEP by applying a backward stable algorithm is backward stable if

$$\|T_0\| = \|T_1\| = \dots = \|T_p\|.$$

Even though this algorithm is designed for eigenvalue problems but not for solving linear systems, it shares similar motivations with our algorithm. We implement it in Figure 5 (d), (e). It can be found that, both strategies perform well for well-conditioned cases. Nevertheless, the accuracy of (18) decays rapidly in the ill-conditioned case; see Figure 5 (d); while our algorithm performs obviously better; see Figure 5 (f).

6 Conclusion

In this work, we introduce infGMRES to reduce the cost of solving linear systems in contour integral-based nonlinear eigensolvers. We have worked out all the details including the convergence-accelerating weighting strategy, the memory-friendly TOAR-like trick, and the selection of the parameters. With these ingredients, we proposed a robust and efficient implementation of infGMRES. While our numerical experiments are carried out in Beyn’s method, this technique can actually be applied to all contour integral-based nonlinear eigensolvers, where several moments are needed to be approximated.

Our method reduces the computational cost by reducing the number of required matrix factorizations. More precisely, it requires as many factorizations as the number of expansion points, which is usually much smaller than the number of quadrature nodes. This is especially relevant for difficult problems, where the quadrature rule demands a large number of quadrature nodes or the scale is extremely large, making a matrix decomposition very expensive.

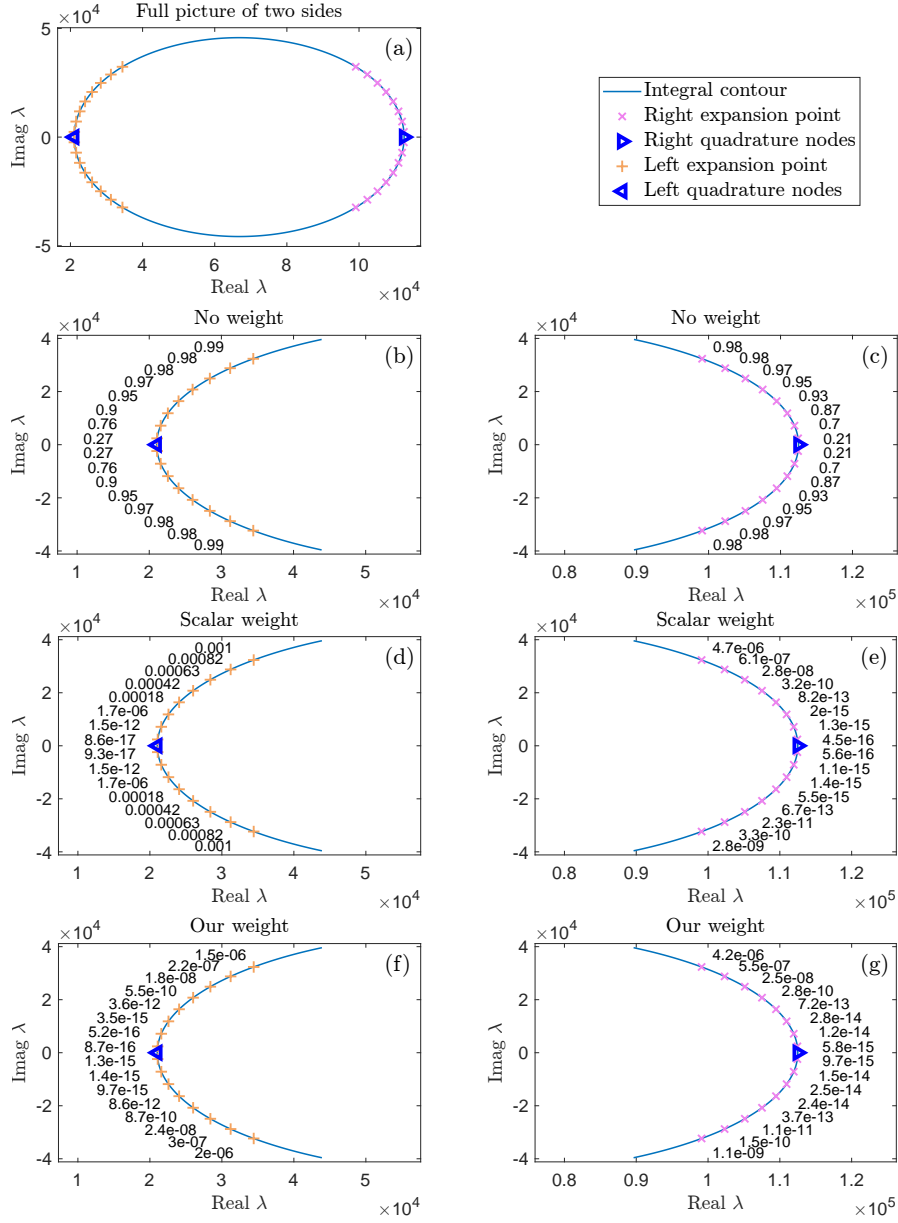


Figure 5: Solving linear systems of the **gun** problem by infGMRES without weighting (b, c), with scaling weighting (18) (d, e) and with our weighting (14) (f, g). (a) shows a full picture of the distribution of the expansion points and quadrature nodes.

Our future work may involve developing a machine learning-based adaptive strategy for selecting expansion points automatically. Additionally, we may implement a block variant of infGMRES to efficiently solve all right-hand sides.

Acknowledgments

We would like to thank Guillaume Demésy for providing us with the data from the photonics test case. Y. Liu and M. Shao were partly supported by the National Natural Science Foundation of China under grant No. 92370105. J. E. Roman was supported by grant PID2022-139568NB-I00 funded by MCIN/AEI/ 10.13039/501100011033 and by “ERDF A way of making Europe”. This work was carried out while Y. Liu was visiting Universitat Politècnica de València.

References

- [1] Junko Asakura, Hiroto Sakurai, Tetsuya Tadano, Tsutomu Ikegami, and Kinji Kimura. A numerical method for polynomial eigenvalue problems using contour integral. *Japan J. Indust. Appl. Math.*, 27(1):73–90, 2010. doi:10.1007/s13160-010-0005-x.
- [2] Junko Asakura, Tetsuya Sakurai, Hiroto Tadano, Tsutomu Ikegami, and Kinji Kimura. A numerical method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett.*, 1:52–55, 2009. doi:10.14495/jsiaml.1.52.
- [3] Manuel Baumann and Martin B. Van Gijzen. Nested Krylov methods for shifted linear systems. *SIAM J. Sci. Comput.*, 37(5):S90–S112, 2015. doi:10.1137/140979927.
- [4] Timo Betcke. Optimal scaling of generalized and polynomial eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 30(4):1320–1338, 2009. doi:10.1137/070704769.
- [5] Timo Betcke, Nicholas J. Higham, Volker Mehrmann, Christian Schröder, and Françoise Tisseur. NLEVP: A collection of nonlinear eigenvalue problems. *ACM Trans. Math. Software*, 39(2):1–28, 2013. doi:10.1145/2427023.2427024.
- [6] Wolf-Jürgen Beyn. An integral method for solving nonlinear eigenvalue problems. *Linear Algebra Appl.*, 436(10):3839–3863, 2012. doi:10.1016/j.laa.2011.03.030.
- [7] Felix Binkowski, Fridtjof Betz, Rémi Colom, Patrice Genevet, and Sven Burger. Poles and zeros in non-Hermitian systems: Application to photonics. *Phys. Rev. B*, 109(4):045414, 2024. doi:10.1103/PhysRevB.109.045414.
- [8] Carmen Campos and Jose E. Roman. NEP: a module for the parallel solution of nonlinear eigenvalue problems in SLEPc. *ACM Trans. Math. Software*, 47(3):1–29, 2021. doi:10.1145/3447544.
- [9] Tzu-Yi Chen and James W. Demmel. Balancing sparse matrices for computing eigenvalues. *Linear Algebra Appl.*, 309(1-3):261–287, 2000. doi:10.1016/S0024-3795(00)00014-8.
- [10] Siobhán Correnty, Elias Jarlebring, and Kirk M. Soodhalter. Preconditioned infinite GMRES for parameterized linear systems. *SIAM J. Sci. Comput.*, pages S120–S141, 2023. doi:10.1137/22M1502380.
- [11] Guillaume Demésy, André Nicolet, Boris Gralak, Christophe Geuzaine, Carmen Campos, and Jose E. Roman. Non-linear eigenvalue problems with GetDP and SLEPc: Eigenmode computations of frequency-dispersive photonic open structures. *Comput. Phys. Commun.*, 257:107509, 2020. doi:10.1016/j.cpc.2020.107509.
- [12] Cedric Effenberger and Daniel Kressner. Chebyshev interpolation for nonlinear eigenvalue problems. *BIT. Numer. Math.*, 52:933–951, 2012. doi:10.1007/s10543-012-0381-5.

- [13] Andreas Frommer and Uwe Glässner. Restarted GMRES for shifted linear systems. *SIAM J. Sci. Comput.*, 19(1):15–26, 1998. doi:10.1137/S1064827596304563.
- [14] Mauricio Garcia-Vergara, Guillaume Demésy, and Frédéric Zolla. Extracting an accurate model for permittivity from experimental data: hunting complex poles from the real line. *Optics Letters*, 42(6):1145–1148, 2017. doi:10.1364/OL.42.001145.
- [15] Brendan Gavin, Agnieszka Międlar, and Eric Polizzi. FEAST eigensolver for nonlinear eigenvalue problems. *J. Comput. Sci.*, 27:107–117, 2018. doi:10.1016/j.jocs.2018.05.006.
- [16] Anne Greenbaum, Miroslav Rozložník, and Zdenek Strakoš. Numerical behaviour of the modified Gram–Schmidt GMRES implementation. *BIT. Numer. Math.*, 37(3):706–719, 1997. doi:10.1007/BF02510248.
- [17] Stefan Güttel and Françoise Tisseur. The nonlinear eigenvalue problem. *Acta Numer.*, 26:1–94, 2017. doi:10.1017/S0962492917000034.
- [18] Stefan Güttel, Roel Van Beeumen, Karl Meerbergen, and Wim Michiels. NLEIGS: A class of fully rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Sci. Comput.*, 36(6):A2842–A2864, 2014. doi:10.1137/130935045.
- [19] Nicholas J. Higham, D. Steven Mackey, Françoise Tisseur, and Seamus D. Garvey. Scaling, sensitivity and stability in the numerical solution of quadratic eigenvalue problems. *Int. J. Numer. Method Engineering*, 73(3):344–360, 2008. doi:10.1002/nme.2076.
- [20] Varun Hiremath and Jose E. Roman. Acoustic modal analysis with heat release fluctuations using nonlinear eigensolvers. *Appl. Math. Comput.*, 458:128249, 2023. doi:10.1016/j.amc.2023.128249.
- [21] Elias Jarlebring and Siobhán Correnty. Infinite GMRES for parameterized linear systems. *SIAM J. Matrix Anal. Appl.*, 43(3):1382–1405, 2022. doi:10.1137/21M1410324.
- [22] Elias Jarlebring, Wim Michiels, and Karl Meerbergen. A linear eigenvalue algorithm for the nonlinear eigenvalue problem. *Numer. Math.*, 122(1):169–195, 2012. doi:10.1007/s00211-012-0453-0.
- [23] Daniel Kressner and Jose E. Roman. Memory-efficient Arnoldi algorithms for linearizations of matrix polynomials in Chebyshev basis. *Numer. Linear Algebra Appl.*, 21(4):569–588, 2014. doi:10.1002/nla.1913.
- [24] Philippe Lalanne, Wei Yan, Alexandre Gras, Christophe Sauvan, J.-P. Hugonin, Mondher Besbes, Guillaume Demésy, M. D. Truong, B. Gralak, F. Zolla, A. Nicolet, F. Binkowski, L. Zschiedrich, S. Burger, J. Zimmerling, R. Remis, P. Urbach, H. T. Liu, and T. Weiss. Quasinormal mode solvers for resonators with dispersive materials. *J. Opt. Soc. Am. A*, 36(4):686–704, 2019. doi:10.1364/JOSAA.36.000686.
- [25] Ding Lu, Yangfeng Su, and Zhaojun Bai. Stability analysis of the two-level orthogonal Arnoldi procedure. *SIAM J. Matrix Anal. Appl.*, 37(1):195–214, 2016. doi:10.1137/151005142.
- [26] Volker Mehrmann and Heinrich Voss. Nonlinear eigenvalue problems: A challenge for modern eigenvalue methods. *GAMM-Mitteilungen*, 27(2):121–152, 2004. doi:10.1002/gamm.201490007.

- [27] Xianshun Ming. Quasinormal mode expansion method for resonators with partial-fraction material dispersion, 2023. [arXiv:2312.11048](#), [doi:10.48550/arXiv.2312.11048](#).
- [28] André Nicolet, Guillaume Demésy, Frédéric Zolla, Carmen Campos, Jose E. Roman, and Christophe Geuzaine. Physically agnostic quasi normal mode expansion in time dispersive structures: From mechanical vibrations to nanophotonic resonances. *Eur. J. Mech. A/Solids*, 100:104809, 2023. [doi:10.1016/j.euromechsol.2022.104809](#).
- [29] Eric Polizzi. Density-matrix-based algorithms for solving eigenvalue problems. *Phys. Rev. B*, 79:115112, 2009. [doi:10.1103/physrevb.79.115112](#).
- [30] Tetsuya Sakurai and Hiroshi Sugiura. A projection method for generalized eigenvalue problems using numerical integration. *J. Comput. Appl. Math.*, 159(1):119–128, 2003. [doi:10.1016/S0377-0427\(03\)00565-X](#).
- [31] Tetsuya Sakurai and Hiroshi Sugiura. CIRRR: a Rayleigh–Ritz type method with contour integral for generalized eigenvalue problems. *Hokkaido Math. J.*, 36(4):745–757, 2007. [doi:10.14492/hokmj/1272848031](#).
- [32] Ping Tak Peter Tang and Eric Polizzi. FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. *SIAM J. Matrix Anal. Appl.*, 35(2):354–390, 2014. [doi:10.1137/13090866X](#).
- [33] Françoise Tisseur and Nicholas J. Higham. Structured pseudospectra for polynomial eigenvalue problems, with applications. *SIAM J. Matrix Anal. Appl.*, 23(1):187–208, 2001. [doi:10.1137/S0895479800371451](#).
- [34] Lloyd N. Trefethen and J. A. C. Weideman. The exponentially convergent trapezoidal rule. *SIAM Rev.*, 56(3):385–458, 2014. [doi:10.1137/130932132](#).
- [35] Roel Van Beeumen, Karl Meerbergen, and Wim Michiels. A rational Krylov method based on Hermite interpolation for nonlinear eigenvalue problems. *SIAM J. Sci. Comput.*, 35(1):A327–A350, 2013. [doi:10.1137/120877556](#).
- [36] Roel Van Beeumen, Karl Meerbergen, and Wim Michiels. Compact rational Krylov methods for nonlinear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 36(2):820–838, 2015. [doi:10.1137/140976698](#).
- [37] Shinnosuke Yokota and Tetsuya Sakurai. A projection method for nonlinear eigenvalue problems using contour integrals. *JSIAM Lett.*, 5:41–44, 2013. [doi:10.14495/jsiaml.5.41](#).