NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2016; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla

A FEAST Algorithm with oblique projection for generalized eigenvalue problems

Guojian Yin[1*†], Raymond H. Chan[2] and Man-Chung Yeung[3]

[1]*School of Mathematics, Sun Yat-sen University, Guangzhou, P. R. China*
[2]*Department of Mathematics, The Chinese University of Hong Kong, Shatin, Hong Kong*
[3] *Department of Mathematics, University of Wyoming, Dept. 3036, 1000 East University Avenue, Laramie, USA*

SUMMARY

The contour-integral based eigensolvers are the recent efforts for computing the eigenvalues inside a given region in the complex plane. The best-known members are the Sakurai-Sugiura (SS) method, its stable version CIRR, and the FEAST algorithm. An attractive computational advantage of these methods is that they are easily parallelizable. The FEAST algorithm was developed for the generalized Hermitian eigenvalue problems. It is stable and accurate. However, it may fail when applied to non-Hermitian problems. Recently, a dual-subspace FEAST algorithm was proposed to extend the FEAST algorithm to non-Hermitian problems. In this paper, we instead use the oblique projection technique to extend FEAST to the non-Hermitian problems. Our approach can be summarized as follows: (i) construct a particular contour integral to form a search subspace containing the desired eigenspace, and (ii) use the oblique projection technique to extract desired eigenpairs with appropriately chosen test subspace. The related mathematical framework is established. Comparing to the dual-subspace FEAST algorithm, our method can save the computational cost roughly by a half if only the eigenvalues or the eigenvalues together with their right eigenvectors are needed. We also address some implementation issues such as how to choose a suitable starting matrix and design efficient stopping criteria. Numerical experiments are provided to illustrate that our method is stable and efficient. Copyright © 2016 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: generalized eigenvalue problems, contour integral, spectral projection

1. INTRODUCTION

Consider the generalized eigenvalue problem

$$A\mathbf{x} = \lambda B\mathbf{x}, \tag{1}$$

where $A, B \in \mathbb{C}^{n\times n}$. The scalars $\lambda \in \mathbb{C}$ and the associated vectors $\mathbf{x} \in \mathbb{C}^n, \mathbf{x} \neq 0$, are called the eigenvalues and right eigenvectors (or simply eigenvectors), respectively. In this paper, we are concerned with computing the eigenvalues of (1) that are located inside a given region in the complex plane together with their eigenvectors.

Large-scale generalized eigenvalue problems arise in various areas of science and engineering, such as dynamic analysis of structures [21], determination of the linearized stability of 3-D fluid flows [7], the electron energy and position problems in quantum chemistry [15], the widely used principal component analysis [42], and the linear discriminant analysis in statistical data analysis

*Correspondence to: School of Mathematics, Sun Yat-sen University, Guangzhou, P. R. China.
† E-mail: guojianyin@gmail.com

[11]. In some applications, it is not the whole spectrum but rather a significant part of it is of interest to the users. For example, in the electronic structure calculations of materials [36], it is required to compute the lowest portion of the spectrum of (1); and in the model reduction of a linear dynamical system, one only needs to know the response over a range of frequencies, see [4, 17].

Solving (1) is a very challenging problem, even if there are various practical methods and software available, see [4]. When A and B have no special structures and the whole spectrum is required, the QZ method [31] is the most widely used method. It uses a sequence of unitary equivalence transformations to reduce the original pair (A, B) to generalized Schur form. The algorithm is numerically stable. However its computational cost is expensive, requiring about $46n^3$ floating point operations [19].

There are several methods for computing only part of the spectrum of (1). The rational Krylov subspace method approximates all eigenvalues in a union of regions around the chosen shifts [34]. However, it needs locking, purging and implicit restart techniques, which create difficulties in practical implementation. The divide-and-conquer approaches, which are based on the sign-function or inverse-free techniques, are also popular choices [5]. However, these approaches always suffer from slow convergence or poor stability [32]. When A and B are Hermitian, a shifted block Lanczos algorithm was proposed in [21] to compute the eigenvalues contained in a given interval. The authors designed a shift strategy combining with the LDL decomposition to guarantee that all eigenvalues in the given interval can be found. But the shift strategy is complicated in practical implementation and its efficiency depends on the distribution of the spectrum.

The methods based on contour integral are recent efforts for solving partial spectrum of (1). When (1) is a diagonalizable and non-degenerate system, a contour integral method, called the Sakurai-Sugiura (SS) method, was proposed in [38] for evaluating the eigenvalues inside a specific domain. In this method, the original problem (1) is reduced to a small eigenproblem with Hankel matrices. However, since Hankel matrices are usually ill-conditioned [6], the SS method always suffers from numerical instability [3, 39]. Later in [39], Sakurai *et al.* used the Rayleigh-Ritz procedure to replace the Hankel matrix approach to get a more stable algorithm called CIRR. In [25] and [24] the block versions of SS and CIRR were proposed respectively to make SS and CIRR also available for degenerate systems. It was shown that SS and CIRR, as well as their corresponding block variants, can be regarded as the Krylov subspace techniques [22, 25]. In [8], Beyn formulated a contour-integral based method for solving the eigenvalues inside a given region of a nonlinear eigenvalue problem.

Recently in [33], Polizzi proposed another eigenproblem solver based on contour integral to solve (1) under the assumptions that A and B are Hermitian and B is positive definite, in which case the eigenvalues of (1) are real-valued. His algorithm, called FEAST, computes all eigenvalues inside a given interval, along with their associated eigenvectors. The FEAST algorithm is accurate and reliable, see [28] for more details. It was shown that the FEAST algorithm can be understood as an accelerated subspace iteration algorithm with the Rayleigh-Ritz procedure [44].

The FEAST algorithm originally was proposed for Hermitian problems. We observe that when it is applied to non-Hermitian problems, it may fail to find the desired eigenvalues. A simple example (Example 3.1) will be given later to illustrate this. Motivated by these facts, our goal is to generalize the FEAST algorithm to non-Hermitian problems and establish the related mathematical framework. The only requirement in our method is that the corresponding matrix pencil $zB - A$ is regular, i.e., $\det(zB - A)$ is not identically zero for all $z \in \mathbb{C}$. In other words, our new method can deal with the most common generalized eigenvalue problems [4]. Unlike FEAST which uses the Rayleigh-Ritz procedure to extract the desired eigenpairs, our generalized FEAST algorithm uses the oblique projection technique to find them.

We should point out that a dual-subspace FEAST algorithm was recently developed in [27, 29] for dealing with non-Hermitian problems. If both the left and the right eigenvectors corresponding to the eigenvalues inside the prescribed region are of interest, both our method and the dual-space method need to construct two subspaces to contain the desired left and right eigenspaces by two contour integrals. As a result, both methods require nearly the same cost. However, if only the

DOI: 10.1002/nla

eigenvalues or the eigenvalues together with their right eigenvectors are needed, then the cost our method will be about half of that of the dual-subspace FEAST algorithm.

One of the main drawbacks of the contour-integral based algorithms, including ours, is that the information about the number of desired eigenvalues has to be known a priori. This is because we need this information (i) to choose an appropriate size for the starting matrix to start the methods, and (ii) to determine whether all desired eigenvalues are captured when the methods stop. In this paper, we use a way similar to that proposed in [37] to find an estimator of an upper bound of the number of eigenvalues inside the target region. It will help to choose the starting matrix. We also provide stopping criteria to capture the desired eigenpairs and to provide their accuracy. With these efforts, our method is applicable in practical implementation. Comparisons with MATLAB's `eig` command, Beyn's method [8] and the block version of CIRR [24, 37] show that our method is an efficient and stable solver for large generalized eigenvalue problems.

The outline of the paper is as follows. In Section 2, we briefly describe two typical contour-integral based eigensolvers: the CIRR method [39] and the FEAST algorithm [33]. In Section 3, we extend the FEAST algorithm to non-Hermitian problems and establish the related mathematical framework. In Section 4, we present a way to find an estimator of an upper bound of the number of eigenvalues inside the prescribed region and give the stopping criteria. Then we present the complete algorithm of our method. In Section 5, numerical experiments are reported to illustrate the efficiency and stability of our method.

Throughout the paper, we use the following notation and terminology. The subspace spanned by the columns of a matrix X is denoted by $\mathrm{span}\{X\}$. The rank and conjugate transpose of X are denoted by $\mathrm{rank}(X)$ and X^* respectively. We denote the submatrix consisting of the first i rows and the first j columns of X by $X_{(1:i,1:j)}$, the submatrices consisting of the first j columns of X and the first i rows of X by $X_{(:,1:j)}$ and $X_{(1:i,:)}$ respectively. The algorithms are presented in MATLAB style.

2. TWO TYPICAL CONTOUR-INTEGRAL BASED EIGENSOLVERS

In this section, we briefly introduce two typical contour-integral based eigensolvers: the CIRR method and the FEAST algorithm. Before starting our discussion, we present some facts about matrix pencil. These facts will play important roles in the derivation of these methods.

Recall that the matrix pencil $zB - A$ is regular if $\det(zB - A)$ is not identically zero for all $z \in \mathbb{C}$. The Weierstrass canonical form for regular matrix pencils is given below.

Theorem 1 ([18])
Let $zB - A$ be a regular matrix pencil of order n. Then there exist nonsingular matrices S and $T \in \mathbb{C}^{n\times n}$ such that

$$TAS = \begin{bmatrix} J_d & 0 \\ 0 & I_{n-d} \end{bmatrix} \quad \text{and} \quad TBS = \begin{bmatrix} I_d & 0 \\ 0 & N_{n-d} \end{bmatrix}, \tag{2}$$

where J_d is a $d \times d$ matrix in Jordan canonical form with its diagonal entries corresponding to the eigenvalues of $zB - A$, N_{n-d} is an $(n-d) \times (n-d)$ nilpotent matrix also in Jordan canonical form, and I_d denotes the identity matrix of order d.

Let J_d be of the form

$$J_d = \begin{bmatrix} J_{d_1}(\lambda_1) & 0 & \cdots & 0 \\ 0 & J_{d_2}(\lambda_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & J_{d_m}(\lambda_m) \end{bmatrix} \tag{3}$$

DOI: 10.1002/nla

where $\sum_{i=1}^{m} d_i = d$ and $J_{d_i}(\lambda_i)$ are $d_i \times d_i$ matrices of the form

$$J_{d_i}(\lambda_i) = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & & 0 & \lambda_i \end{bmatrix}, \qquad i = 1, 2, \ldots, m$$

with λ_i being the eigenvalues. Here the λ_i are not necessarily distinct and can be repeated according to their multiplicities.

Let N_{n-d} be of the form

$$N_{n-d} = \begin{bmatrix} N_{d'_1} & 0 & \cdots & 0 \\ 0 & N_{d'_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & N_{d'_{m'}} \end{bmatrix},$$

where $\sum_{i=1}^{m'} d'_i = n - d$ and $N_{d'_i}$ are $d'_i \times d'_i$ matrices of the form

$$N_{d'_i} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & & 0 & 0 \end{bmatrix}, \qquad i = 1, 2, \ldots, m'.$$

Let us partition S into block form

$$S = [S_1, S_2, \ldots, S_m, S_{m+1}], \tag{4}$$

where $S_i \in \mathbb{C}^{n \times d_i}$, $1 \le i \le m$ and $S_{m+1} \in \mathbb{C}^{n \times (n-d)}$. One can easily verify by (2) (or see (19) for details) that the first column in each S_i, $i = 1, \ldots, m$, is the eigenvector associated with the eigenvalue λ_i of (1).

2.1. *The CIRR method*

In [38], Sakurai *et al.* used a moment-based technique to formulate a contour-integral based method, which now is known as SS, for finding the eigenvalues of (1) inside a given region. Since the SS method always suffers from numerical instability, a stable version, called the CIRR method, was later developed. The CIRR method uses the Rayleigh-Ritz procedure to extract desired eigenpairs [24, 39].

Below we show how to use the CIRR method to compute the eigenvalues inside Γ, which is a given positively oriented simple closed curve in the complex plane. Without loss of generality, let the set of eigenvalues of (1) enclosed by Γ be $\{\lambda_1, \ldots, \lambda_l\}$, and $s := d_1 + d_2 + \cdots + d_l$ be the number of eigenvalues inside Γ with multiplicity taken into account. Define the contour integrals

$$F_i := \frac{1}{2\pi\sqrt{-1}} \oint_{\Gamma} z^i (zB - A)^{-1} dz, \quad i = 0, 1, \ldots. \tag{5}$$

It was shown in [24] that

$$F_i = S_{(:,1:s)} (J_{(1:s,1:s)})^i T_{(1:s,:)}, \tag{6}$$

where S and T are given by Theorem 1. The CIRR method uses the Rayleigh-Ritz procedure to extract the eigenpairs inside Γ [39]. Originally, it was derived under the assumptions that (1) is

DOI: 10.1002/nla

a Hermitian system and the desired eigenvalues are distinct, i.e., they are non-degenerate [39]. However, based on the following theorem, the CIRR method was extended to non-Hermitian cases in [24].

Theorem 2 ([24])
Let $L,\ D \in \mathbb{C}^{n\times t},\ t \geq s$, be arbitrary matrices, and $R = F_0 D$, where F_0 is defined in (5). A projected matrix pencil $z\bar{B} - \bar{A}$ is defined by $\bar{B} = L^* B R$ and $\bar{A} = L^* A R$. If the ranks of both $L^*(T^{-1})_{(:,1:s)}$ and $T_{(1:s,:)}D$ are s, the non-singular part of the projected matrix pencil is equivalent to $zI_s - (J_d)_{(1:s,1:s)}$.

Theorem 2 says that the desired eigenvalues $\{\lambda_i\}_{i=1}^{s}$ can be solved by computing the eigenvalues of the projected eigenproblem $z\bar{B} - \bar{A}$, if the ranks of both $L^*(T^{-1})_{(:,1:s)}$ and $T_{(1:s,:)}D$ are s.

Define the columns of D to be

$$D_{(:,i)} = (T^{-1})_{(:,1:s)}(J_{(1:s,1:s)})^{i-1}T_{(1:s,:)}v, \quad i = 1, 2, \ldots, t. \tag{7}$$

It was shown in [25] that the rank of $T_{(1:s,:)}D$ is s, if all the elements of $T_{(1:s,:)}v$ are non-zero and there is no degeneracy in $J_{(1:s,1:s)}$. By (5) and (7), we have

$$R_{(:,i)} = F_0 D_{(:,i)} = F_i v, \quad i = 1, \ldots, t. \tag{8}$$

Based on Theorem 2, the right Ritz space is spanned by the vectors $\{F_i v\}_{i=0}^{t-1}$. As for the left Ritz space which is required in the CIRR for non-Hermitian problems, it is chosen to be the same as the right one.

In order to remove the restriction on the non-degeneracy in $J_{(1:s,1:s)}$, a block CIRR method was also proposed in [24], where the random vector v in (8) is replaced by a random matrix $Y \in \mathbb{C}^{n\times h}$ of appropriate dimension. The right and the left Ritz spaces are spanned by the vectors $\{F_i Y\}_{i=0}^{g-1}$, where g is a positive integer satisfying $hg \geq s$. Then all eigenvalues of h'-order degeneracy, $h' \leq h$, can be found, see [24, 37]. Obviously, the main task of the block CIRR method is to evaluate $\{F_i Y\}_{i=0}^{g-1}$. In practice, $F_i Y$ are computed approximately by a quadrature scheme according to (5). Below is the block CIRR algorithm for non-Hermitian problems.

Algorithm 1
Input matrices A and B, a random matrix $Y \in \mathbb{C}^{n\times h}$, and a positive integer g satisfying $t = hg \geq s$. The function "BLOCK_CIRR" computes eigenpairs of (1) that are located inside Γ, and they are output in the vector Λ_s and the matrix X_s.

Function $[\Lambda_s, X_s]$ = BLOCK_CIRR(A, B, Y, g, Γ)
1. Compute $R_i = F_i Y, i = 0, 1, \ldots, g-1$, approximately by a quadrature scheme.
2. Compute the singular value decomposition: $[R_0, \ldots, R_{g-1}] = U\Sigma V^*$.
3. Set $\bar{A} = U^* A U$ and $\bar{B} = U^* B U$.
4. Solve the generalized eigenproblem of size t: $\bar{A}\mathbf{y} = \bar{\lambda}\bar{B}\mathbf{y}$, to obtain the eigenpairs $\{(\bar{\lambda}_i, \mathbf{y}_i)\}_{i=1}^{t}$.
5. Compute $\bar{\mathbf{x}}_i = U\mathbf{y}_i, i = 1, 2, \ldots t$, and select s approximate eigenpairs.

We see that for Algorithm 1, in order to choose the parameters h and g, one needs to know s and the degrees of degeneracy of the desired eigenvalues. In the recent article [37], Sakurai *et al.* gave a method for choosing a suitable h for fixed g. We will describe the method in Section 4.1. Also in [37], the authors suggested to perform iterative refinement in case the eigenpairs computed by Algorithm 1 cannot attain the prescribed accuracy.

2.2. *The FEAST algorithm*

In this section, we give a brief introduction to the FEAST algorithm [33] due to Polizzi. The FEAST algorithm was formulated under the assumptions that A and B are Hermitian and B is positive definite, in which case the eigenvalues of (1) are real-valued [4]. It is used to find all eigenvalues of (1) within a specified interval, say $[\sigma_1, \sigma_2]$, and their associated eigenvectors. Here we also assume

DOI: 10.1002/nla

that the desired eigenvalues are $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_s$. Let Γ be any contour that contains $\{\lambda_i\}_{i=1}^s$ inside. For example, Γ can be the circle with center at $c = (\sigma_1 + \sigma_2)/2$ and radius $r = (\sigma_2 - \sigma_1)/2$.

When $zB - A$ is a definite matrix pencil, the Weierstrass canonical form (2) is reduced to

$$TAS = \Lambda = \mathrm{diag}([\lambda_1, \lambda_2 \cdots, \lambda_n]) \quad \text{and} \quad TBS = I_n. \tag{9}$$

It is easy to see that $T = S^*$, and the columns $S_{(:,i)}$ are the eigenvectors corresponding to λ_i, $i = 1, \ldots, n$. Therefore according to (6), we have

$$F_0 = S_{(:,1:s)} T_{(1:s,:)} = S_{(:,1:s)} (S_{(:,1:s)})^*. \tag{10}$$

Let $W := F_0 Y = S_{(:,1:s)} (S_{(:,1:s)})^* Y$, where Y is an $n \times t$ matrix with $t \ge s$. Then W forms a basis for the desired eigenspace $\mathrm{span}\{S_{(:,1:s)}\}$ if $(S_{(:,1:s)})^* Y$ is full-rank. In the FEAST algorithm, the elements of Y are chosen to be random numbers to increase the chance that W may form a basis for $\mathrm{span}\{S_{(:,1:s)}\}$. According to the Rayleigh-Ritz procedure [33, 43], the problem (1) is transformed to the problem of computing the eigenpairs of the smaller generalized eigenvalue problem

$$\hat{A}\mathbf{y} = \lambda \hat{B}\mathbf{y}$$

of size $t \times t$, where $\hat{A} = W^* A W$, $\hat{B} = W^* B W$, and $t < n$.

Now the task is to get the basis $W = F_0 Y$. Since $S_{(:,1:s)}$ is unknown, one cannot use (10) to compute W. Instead W is computed by (5) numerically using a quadrature scheme such as the Gauss-Legendre quadrature rule [12]. The complete FEAST algorithm is given as follows.

Algorithm 2
Input Hermitian matrices A and B with B being positive definite, a random matrix $Y \in \mathbb{R}^{n \times t}$, where $t \ge s$, the circle Γ enclosing the interval $[\sigma_1, \sigma_2]$, and a convergence tolerance ϵ. The function "FEAST" computes eigenpairs $(\hat{\lambda}_i, \hat{\mathbf{x}}_i)$ of (1) that satisfy

$$\hat{\lambda}_i \in [\sigma_1, \sigma_2] \quad \text{and} \quad \sum_{i=1}^{s} \hat{\lambda}_i < \epsilon, \tag{11}$$

and they are output in the vector Λ_s and the matrix X_s.

Function $[\Lambda_s, X_s]$ = FEAST$(A, B, Y, \Gamma, \epsilon)$
1. Compute $W = F_0 Y$ approximately by the Gauss-Legendre quadrature rule.
2. Set $\hat{A} = W^* A W$ and $\hat{B} = W^* B W$.
3. Solve the generalized eigenproblem of size t: $\hat{A}\mathbf{y} = \hat{\lambda}\hat{B}\mathbf{y}$, to obtain the eigenpairs $\{(\hat{\lambda}_i, \mathbf{y}_i)\}_{i=1}^t$.
4. Compute $\hat{\mathbf{x}}_i = W\mathbf{y}_i, i = 1, 2, \ldots t$.
5. Check if $\{(\hat{\lambda}_i, \hat{\mathbf{x}}_i)\}_{i=1}^t$ satisfy the convergence criteria (11). If s eigenpairs satisfy (11), stop. Otherwise, set $X_t = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_t]$ and $Y = BX_t$, then go back to Step 1.

The FEAST algorithm can be understood as an accelerated subspace iteration combining with the Rayleigh-Ritz procedure [44]. It is an accurate and reliable technique [28]. However, like CIRR, in practice we have to know s in advance in order to choose t for the starting matrix Y and to determine whether all desired eigenpairs are found. In [44], a technique was presented to find an estimation of s under the conditions that A and B are Hermitian and B is positive definite. The estimation can help to select an appropriate starting matrix Y.

3. A FEAST METHOD WITH OBLIQUE PROJECTION

The FEAST method was originally developed for generalized Hermitian eigenvalue problems. We note that it may fail when applied to non-Hermitian problems, as is shown by the following example.

DOI: 10.1002/nla

EXAMPLE 3.1: Let A and B be defined as follows:

$$A = \begin{pmatrix} 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0.2 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Then (2) holds with

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \text{and} \quad S = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

In fact, $TAS = \mathrm{diag}([5, 2, 0.5, 0.2])$ and $TBS = I_4$. Suppose we want to find the eigenvalues of $A\mathbf{x} = \lambda B\mathbf{x}$ lying inside the unit circle. Obviously, the eigenvalues of interest are 0.2 and 0.5, and the corresponding eigenvectors are $S_{(:,4)} = [1, 0, 0, 0]^*$ and $S_{(:,3)} = [0, 1, 0, 0]^*$. By (6) and (10), it is easy to check that both projected matrices $\hat{A}$ and $\hat{B}$ in the FEAST algorithm are zero for any given random matrix Y. Hence any complex number will be an eigenvalue of the corresponding projected eigenproblem $\hat{A}\mathbf{y} = \lambda\hat{B}\mathbf{y}$!

In view of the above example, we now extend the FEAST algorithm to non-Hermitian problems. The only requirement of our method is that the corresponding matrix pencil $zB - A$ is regular. Hence our method can deal with the most common generalized eigenvalue problems [4]. Below we establish the related mathematical framework.

Again without loss of generality, we let the eigenvalues of (1) enclosed by Γ be $\{\lambda_1, \ldots, \lambda_l\}$, and $s := d_1 + d_2 + \cdots + d_l$ be the number of eigenvalues inside Γ with multiplicity taken into account. Define the contour integral

$$Q := F_0 B = \frac{1}{2\pi\sqrt{-1}} \oint_\Gamma (zB - A)^{-1} B dz, \tag{12}$$

here $(zB - A)^{-1}B$ is the generalized resolvent. For $z \neq \lambda_i$, according to (3) the matrix $zI_d - J_d$ is invertible. Hence by (2), the generalized resolvent is given by

$$\begin{aligned} (zB - A)^{-1}B &= S \begin{bmatrix} (zI_d - J_d)^{-1} & 0 \\ 0 & (zN_{n-d} - I_{n-d})^{-1} \end{bmatrix} TB \\ &= S \begin{bmatrix} (zI_d - J_d)^{-1} & 0 \\ 0 & (zN_{n-d} - I_{n-d})^{-1} \end{bmatrix} \begin{bmatrix} I_d & 0 \\ 0 & N_{n-d} \end{bmatrix} S^{-1} \\ &= S \begin{bmatrix} (zI_d - J_d)^{-1} & 0 \\ 0 & (zN_{n-d} - I_{n-d})^{-1} N_{n-d} \end{bmatrix} S^{-1}. \end{aligned} \tag{13}$$

Notice that the first diagonal block $(zI_d - J_d)^{-1}$ in (13) is of the block diagonal form where each diagonal sub-block is of the form:

$$(zI_{d_i} - J_{d_i}(\lambda_i))^{-1} = \begin{bmatrix} \frac{1}{z-\lambda_i} & \frac{1}{(z-\lambda_i)^2} & \cdots & \frac{1}{(z-\lambda_i)^{d_i}} \\ 0 & \frac{1}{z-\lambda_i} & \cdots & \frac{1}{(z-\lambda_i)^{d_i-1}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{z-\lambda_i} \end{bmatrix}. \tag{14}$$

DOI: 10.1002/nla

Similarly, the second diagonal block in (13) is also of the block diagonal form where each diagonal sub-block is of the form:

$$(zN_{d_i'} - I_{d_i'})^{-1}N_{d_i'} = \begin{bmatrix} 0 & -1 & -z & \cdots & -z^{d_i'-2} \\ 0 & 0 & -1 & \cdots & -z^{d_i'-3} \\ 0 & 0 & 0 & \cdots & -z^{d_i'-4} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{15}$$

Then, according to the residue theorem in complex analysis [1], it follows from (13)–(15) that

$$Q = \frac{1}{2\pi\sqrt{-1}}\oint_\Gamma (zB - A)^{-1}B dz = S\begin{bmatrix} I_s & 0 \\ 0 & 0 \end{bmatrix} S^{-1} = S_{(:,1:s)}(S^{-1})_{(1:s,:)}. \tag{16}$$

Using the remark following (4), we know that $\mathcal{K} := \mathrm{span}\{S_{(:,1:s)}\}$ contains the eigenspace corresponding to the eigenvalues $\{\lambda_1, \ldots, \lambda_l\}$. Since $Q^2 = Q$, Q is a spectral projector onto $\mathcal{K}$. Define $U := QY$, where Y is an appropriately chosen matrix so that U forms a basis for $\mathcal{K}$. As in FEAST, we choose Y randomly, and we show below that the resulting U does form a basis for $\mathcal{K}$.

Lemma 1
Let $Y \in \mathbb{R}^{n\times s}$. If the entries of Y are random numbers from a continuous distribution and that they are independent and identically distributed (i.i.d.), then with probability 1, the matrix $(S^{-1})_{(1:s,:)}Y$ is nonsingular.

Proof
Let $Z = (S^{-1})_{(1:s,:)}Y$. Consider $|\det(Z)|^2$, the square of the absolute value of $\det(Z)$, as a real coefficient polynomial in the elements of Y. We now show that the polynomial is non-zero, i.e. $|\det(Z)|^2 \not\equiv 0$.

Since the rank of $(S^{-1})_{(1:s,:)}$ is s, it has an $s \times s$ nonsingular submatrix. Without loss of generality, let the left $s \times s$ submatrix of $(S^{-1})_{(1:s,:)}$ be nonsingular. Then set $Y = [I_s, 0]^T$, we have $\det(Z) = \det((S^{-1})_{(1:s,:)}Y) \neq 0$. Therefore the polynomial $|\det(Z)|^2$ is not identically zero. Hence the set $\mathcal{Z}$ of zeros of the polynomial $|\det(Z)|^2$ is of measure zero in $\mathbb{R}^{ns}$ according to [26, Prop. 4]. When Y is randomly picked, it is with probability 1 that $Y \notin \mathcal{Z}$, or equivalently, $|\det((S^{-1})_{(1:s,:)}Y)|^2 \neq 0$. □

Since

$$U = QY = S_{(:,1:s)}(S^{-1})_{(1:s,:)}Y, \tag{17}$$

and $(S^{-1})_{(1:s,:)}Y$ is nonsingular by Lemma 1, the columns of U form a basis for $\mathcal{K}$. Our next step is to project the original problem (1) onto a small subspace where we can extract the desired eigenpairs. Unlike CIRR and FEAST which use the Rayleigh-Ritz procedure to extract the desired eigenpairs, here we resort to the oblique projection method, namely, imposing the Petrov-Galerkin condition [4, 35]. Since $\mathcal{K}$ contains the eigenspace corresponding to the eigenvalues inside Γ, it is natural to choose $\mathcal{K}$ as the search subspace. Our next task is to seek an appropriate test subspace.

Let us further partition each S_i in (4) into $S_i = [\mathbf{s}_1^i, \mathbf{s}_2^i, \ldots, \mathbf{s}_{d_i}^i]$ with $\mathbf{s}_j^i \in \mathbb{C}^n$, $1 \le i \le m, 1 \le j \le d_i$. Notice that by (2), we have for any eigenvalue λ_i, $1 \le i \le m$,

$$\begin{aligned} (\lambda_i B - A)S\begin{bmatrix} I_d & 0 \\ 0 & N_{n-d} \end{bmatrix} &= T^{-1}\begin{bmatrix} \lambda_i I_d - J_d & 0 \\ 0 & (\lambda_i N_{n-d} - I_{n-d})N_{n-d} \end{bmatrix} \\ &= T^{-1}\begin{bmatrix} I_d & 0 \\ 0 & N_{n-d} \end{bmatrix}\begin{bmatrix} \lambda_i I_d - J_d & 0 \\ 0 & \lambda_i N_{n-d} - I_{n-d} \end{bmatrix} \\ &= BS\begin{bmatrix} \lambda_i I_d - J_d & 0 \\ 0 & \lambda_i N_{n-d} - I_{n-d} \end{bmatrix}. \end{aligned} \tag{18}$$

By comparing the first d columns on both sides above, we get

$$(\lambda_i B - A)\mathbf{s}_j^i = B\mathbf{s}_{j-1}^i, \quad 1 \le j \le d_i, \quad 1 \le i \le m, \tag{19}$$

DOI: 10.1002/nla

with $\mathbf{s}_0^i \equiv \mathbf{0}$. In particular, $\mathbf{s}_1^i$ is the eigenvector corresponding to the eigenvalue λ_i for all $1 \le i \le m$. From (19), we see that $A\mathcal{K} \subseteq B\mathcal{K}$. Therefore we choose $B\mathcal{K}$ as the test subspace. The Petrov-Galerkin condition then becomes:

$$(A\mathbf{x}_i - \lambda_i B\mathbf{x}_i) \perp B\mathcal{K}, \quad 1 \le i \le l, \tag{20}$$

with $\lambda_i \in \mathbb{C}$ and $\mathbf{x}_i \in \mathcal{K}$.

Now we are in the position to find a basis for $B\mathcal{K}$. From (2), we know that the rank of $BS_{(:,1:s)}$ is s. Hence by Lemma 1, $BU = BS_{(:,1:s)}(S^{-1})_{(1:s,:)}Y$ is full-rank, which implies that BU forms a basis for $B\mathcal{K}$. Recall that U forms a basis for $\mathcal{K}$, and we seek an $\mathbf{x}_i \in \mathcal{K}$ satisfying (20). Therefore (20) can be written in matrix form

$$(BU)^*(AU\mathbf{y}_i - \lambda_i BU\mathbf{y}_i) = 0, \tag{21}$$

where $\mathbf{y}_i \in \mathbb{C}^s$ satisfying $\mathbf{x}_i = U\mathbf{y}_i$. Accordingly, we get the projected eigenproblem

$$\tilde{A}\mathbf{y} = \tilde{\lambda}\tilde{B}\mathbf{y}, \tag{22}$$

with

$$\tilde{A} = (BU)^*AU \quad \text{and} \quad \tilde{B} = (BU)^*BU. \tag{23}$$

Our method is to compute the desired eigenpairs of (1) by solving the projected eigenproblem (22). The theory behind our method is given in the next theorem.

Theorem 3 (a) Let $\{(\tilde{\lambda}_i, \mathbf{y}_i)\}_{i=1}^s$ be s eigenpairs of the projected eigenproblem (22). Then $\{(\tilde{\lambda}_i, U\mathbf{y}_i)\}_{i=1}^s$ are the eigenpairs of (1) located inside Γ.

(b) If $\mathcal{Y}_{\tilde{\lambda}_i}$ is the eigenspace of (22) corresponding to the eigenvalue $\tilde{\lambda}_i$, then $U\mathcal{Y}_{\tilde{\lambda}_i}$ is the eigenspace of (1) corresponding to the eigenvalue $\tilde{\lambda}_i$.

Proof

(a): First, since $A\mathcal{K} \subseteq B\mathcal{K}$ and BU forms a basis for $B\mathcal{K}$, there exist vectors $\mathbf{q}_i \in \mathbb{C}^s, 1 \le i \le s$, such that

$$AU\mathbf{y}_i - \tilde{\lambda}_i BU\mathbf{y}_i = BU\mathbf{q}_i, \quad 1 \le i \le s.$$

By (21), we have $(BU)^*(BU)\mathbf{q}_i = 0$. Since BU is full-rank, $\mathbf{q}_i = 0$. Consequently, $AU\mathbf{y}_i = \tilde{\lambda}_i BU\mathbf{y}_i$. Thus $\{(\tilde{\lambda}_i, U\mathbf{y}_i)\}_{i=1}^s$ are the eigenpairs of (1).

Next we want to show that the eigenvalues $\{\tilde{\lambda}_i\}_{i=1}^s$ are exactly the s eigenvalues of (1) inside Γ. By (18), we can easily verify that

$$(zB - A)S_{(:,1:s)} = BS_{(:,1:s)}(zI_s - (J_d)_{(1:s,1:s)}), \quad \forall z \in \mathbb{C}.$$

Hence by the definitions in (23) and (17), we have

$$\begin{aligned} z\tilde{B} - \tilde{A} &= (BU)^*(zB - A)S_{(:,1:s)}(S^{-1})_{(1:s,:)}Y \\ &= (BU)^*BS_{(:,1:s)}(zI_s - (J_d)_{(1:s,1:s)})(S^{-1})_{(1:s,:)}Y. \end{aligned} \tag{24}$$

Therefore,

$$\det(z\tilde{B} - \tilde{A}) = \det((BU)^*BS_{(:,1:s)})\det(zI_s - (J_d)_{(1:s,1:s)})\det((S^{-1})_{(1:s,:)}Y).$$

Since $BS_{(:,1:s)}$ is of full rank s (see (2)), and $(S^{-1})_{(1:s,:)}Y$ is nonsingular by Lemma 1, the matrix $(BU)^*BS_{(:,1:s)} = ((S^{-1})_{(1:s,:)}Y)^*(BS_{(:,1:s)})^*BS_{(:,1:s)}$ is nonsingular. Hence $\det(z\tilde{B} - \tilde{A}) = 0$ if and only if $\det(zI_s - (J_d)_{(1:s,1:s)}) = 0$. By the special structure of $(J_d)_{(1:s,1:s)}$ (see (3)), the zeros of the determinant $\det(z\tilde{B} - \tilde{A})$ are precisely $\{\lambda_i\}_{i=1}^l$ with multiplicities $\{d_i\}_{i=1}^l$ respectively. Therefore, $\{\tilde{\lambda}_i\}_{i=1}^s$ are precisely all the eigenvalues of (1) inside Γ.

(b): Let $\mathcal{X}_{\tilde{\lambda}_i}$ be the eigenspace of (1.1) corresponding to the eigenvalue $\tilde{\lambda}_i$. Then $U\mathcal{Y}_{\tilde{\lambda}_i} \subseteq \mathcal{X}_{\tilde{\lambda}_i}$ by Part (a). From (3) and (24), it can be seen that $\dim(\mathcal{Y}_{\tilde{\lambda}_i})$ is equal to the number of Jordan blocks

DOI: 10.1002/nla

in $(J_d)_{(1:s,1:s)}$ corresponding to the eigenvalue $\tilde{\lambda}_i$. On the other hand, the later coincides with the number of Jordan blocks in J_d corresponding to $\tilde{\lambda}_i$, which is equal to $\dim(\mathcal{X}_{\tilde{\lambda}_i})$. Therefore, we have $\dim(\mathcal{Y}_{\tilde{\lambda}_i}) = \dim(\mathcal{X}_{\tilde{\lambda}_i})$. Since U has full column rank, it follows that

$$\dim(U\mathcal{Y}_{\tilde{\lambda}_i}) = \dim(\mathcal{Y}_{\tilde{\lambda}_i}) = \dim(\mathcal{X}_{\tilde{\lambda}_i}).$$

Therefore, we have $U\mathcal{Y}_{\tilde{\lambda}_i} = \mathcal{X}_{\tilde{\lambda}_i}$. □

Thus computing the eigenpairs of (1) inside Γ is transformed into computing the eigenpairs of the small s-by-s projected problem (22), which can be solved by standard solvers in LAPACK [4, 13], such as `xGGES` and `xGGEV` [2].

In order to construct the projected eigenproblem (22), the most important task is to compute the matrix U in (17). In practice, we have to compute U approximately by a quadrature rule:

$$U = QY = \frac{1}{2\pi\sqrt{-1}} \oint_{\Gamma} (zB - A)^{-1} B dz Y \approx \frac{1}{2\pi\sqrt{-1}} \sum_{j=1}^{q} \omega_j (z_j B - A)^{-1} BY, \tag{25}$$

where z_j are the quadrature nodes on Γ associated with weights ω_j.

We summarize our above derivation into the following algorithm.

Algorithm 3

Input $A, B \in \mathbb{C}^{n\times n}$, an i.i.d. random matrix $Y \in \mathbb{R}^{n\times t}$ where $t \geq s$, a closed curve Γ, a convergence tolerance ϵ, and "`max_iter`" to control the maximum number of iterations. The function "EIGENPAIRS" computes eigenpairs $(\tilde{\lambda}_i, \tilde{\mathbf{x}}_i)$ of (1) that satisfies

$$\tilde{\lambda}_i \text{ inside } \Gamma \quad \text{and} \quad \frac{\|A\tilde{\mathbf{x}}_i - \tilde{\lambda}_i B\tilde{\mathbf{x}}_i\|_2}{\|A\tilde{\mathbf{x}}_i\|_2 + \|B\tilde{\mathbf{x}}_i\|_2} < \epsilon. \tag{26}$$

The results are stored in the vector Λ and the matrix X.

Function $[\Lambda, X] =$ EIGENPAIRS$(A, B, Y, \Gamma, \epsilon,$ `max_iter`$)$
1. For $k = 1, \cdots,$ `max_iter`
2. Compute U approximately by the quadrature rule (25).
3. Compute QR decompositions: $U = U_1 R_1$ and $BU = U_2 R_2$.
4. Form $\tilde{A} = U_2^* A U_1$ and $\tilde{B} = U_2^* B U_1$.
5. Solve the projected eigenproblem $\tilde{A}\mathbf{y} = \tilde{\lambda}\tilde{B}\mathbf{y}$ of size t to obtain eigenpairs $\{(\tilde{\lambda}_i, \mathbf{y}_i)\}_{i=1}^{t}$. Set $\tilde{\mathbf{x}}_i = U_1\mathbf{y}_i, i = 1, 2, \ldots, t$.
6. Set $\Lambda = [\,]$ and $X = [\,]$.
7. For $i = 1 : t$
8. If $(\tilde{\lambda}_i, \tilde{\mathbf{x}}_i)$ satisfies (26), then $\Lambda = [\Lambda, \tilde{\lambda}_i]$ and $X = [X, \tilde{\mathbf{x}}_i]$.
9. End
10. If there are s eigenpairs satisfying (26), stop. Otherwise, set $Y = U_1$.
11. End.

Algorithm 3 faces the same issue as in CIRR and FEAST, that is we have to know s in advance in order to choose t for the starting matrix Y and to determine whether all desired eigenpairs are found. More precisely, the number of columns t of Y should satisfy $t \geq s$. This is because if $t < s$, then $\mathrm{rank}(U) < s$. Consequently, the columns of U cannot form a basis for $\mathcal{K}$. Moreover, since s is not known a priori, it is also hard to decide whether all desired eigenvalues are found and therefore it is hard to decide when to stop the algorithm. In the next section, we present strategies to address these two problems, which will make the resulting algorithm applicable to practical implementation.

DOI: 10.1002/nla

4. OUR ALGORITHM

In this section, we first introduce a method to find an estimator of an upper bound for s. Our method is similar to a technique proposed in [37]. Next we design stopping criteria to capture all eigenvalues. After that, we present the complete algorithm.

4.1. Estimating an Upper Bound for the Number of Eigenvalues inside Γ

In the following, by "$M \sim \mathsf{N}_{p\times q}(0,1)$", we mean M is a $p \times q$ matrix with i.i.d. entries drawn from the standard normal distribution $\mathsf{N}(0,1)$.

Recently, an approach based on stochastic estimator of $\mathrm{trace}(Q)$ was described in [14, 16] for finding an estimation of s. Below we study this kind of stochastic estimation method.

Theorem 4
Let $Y \sim \mathsf{N}_{n\times p}(0,1)$, then we have

$$\mathbb{E}[\frac{1}{p}\mathrm{trace}(Y^*U)] = \mathrm{trace}(Q) = s, \tag{27}$$

and

$$\mathrm{Var}[\frac{1}{p}\mathrm{trace}(Y^*U)] = \frac{\mathrm{trace}(Q^*Q)+s}{p}. \tag{28}$$

Proof
By (16) and (17), we establish

$$\begin{aligned}\mathbb{E}[\frac{1}{p}\mathrm{trace}(Y^*U)] &= \frac{1}{p}\mathbb{E}[\mathrm{trace}(Y^*QY)]\\ &= \mathrm{trace}(Q) = \mathrm{trace}(S_{(:,1:s)}(S^{-1})_{(1:s,:)})\\ &= \mathrm{trace}((S^{-1})_{(1:s,:)}S_{(:,1:s)}) = \mathrm{trace}(I_s) = s.\end{aligned} \tag{29}$$

Therefore, (27) holds.

Next we compute the variance using the approach in [20]. First, it can be shown that

$$\begin{aligned}\mathbb{E}[(\mathrm{trace}(Y^*U))^2] &= \mathbb{E}[(\sum_{k}^{p}\sum_{i,j}^{n} Y_{(k,i)}Q_{(i,j)}Y_{(k,j)})(\sum_{l}^{p}\sum_{r,q}^{n} Y_{(l,r)}Q_{(r,q)}Y_{(l,q)})]\\ &= \mathbb{E}[\sum_{k}^{p}\sum_{l}^{p}\sum_{i,j}^{n}\sum_{r,q}^{n} Q_{(i,j)}Q_{(r,q)}Y_{(k,i)}Y_{(k,j)}Y_{(l,r)}Y_{(l,q)}]\\ &= p(\sum_{i\neq j} Q_{(i,j)}^2 + \sum_{i\neq j} Q_{(i,j)}Q_{(j,i)} + p\sum_{i,j} Q_{(i,i)}Q_{(j,j)} + 2\sum_{i} Q_{(i,i)}^2).\end{aligned}$$

On the other hand, it can be verified that

$$(\mathbb{E}[\mathrm{trace}(Y^*U)])^2 = (\mathbb{E}[\sum_{k}^{p}\sum_{i,j}^{n} Y_{(k,i)}Q_{(i,j)}Y_{(k,j)}])^2 = p^2\sum_{i,j} Q_{(i,i)}Q_{(j,j)}.$$

As a result, the variance of the estimator $\frac{1}{p}\mathrm{trace}(Y^*U)$ is

$$\begin{aligned}\mathrm{Var}[\frac{1}{p}\mathrm{trace}(Y^*U)] &= \frac{1}{p^2}\{\mathbb{E}[(\mathrm{trace}(Y^*U))^2] - (\mathbb{E}[\mathrm{trace}(Y^*U)])^2\}\\ &= \frac{1}{p}(\sum_{i\neq j} Q_{(i,j)}^2 + \sum_{i\neq j} Q_{(i,j)}Q_{(j,i)} + 2\sum_{i} Q_{(i,i)}^2)\\ &= \frac{1}{p}(\mathrm{trace}(Q^*Q) + \mathrm{trace}(Q^2)) = \frac{1}{p}(\mathrm{trace}(Q^*Q) + s).\end{aligned}$$

□

DOI: 10.1002/nla

From (27) it is clear that $\frac{1}{p}\mathrm{trace}(Y^*QY)$ is an unbiased estimator of $\mathrm{trace}(Q)$, therefore we expect $s_0 := \lceil \frac{1}{p}\mathrm{trace}(Y^*U)\rceil$ to provide a good estimation of s if the value of $\frac{1}{p}(\mathrm{trace}(Q^*Q)+s)$ is small (see (28)). However, according to (25) we know that the larger the value p is, the more expensive computing U will be. Therefore, in the experiment, we take the size of the sample vectors $p = 50$, a moderate number.

In Algorithm 3, we need to choose a number t, satisfying $t \geq s$, for the starting matrix $Y \in \mathbb{R}^{n\times t}$. However, in practice, the estimation s_0 may be less than s, and we may not know this fact as we do not know s. Below we present a way to find an estimator of an upper bound t based on s_0.

Recall that in CIRR, for a given integer g, we need to select an h such that $t = hg \geq s$. In [37], a method was proposed for selecting a suitable h. The method works as follows. Assume an estimation s_0 of s is available. Let $Y \in \mathbb{C}^{n\times h}$, where $h = \lceil \frac{\kappa s_0}{g}\rceil$ and $\kappa > 1$. Compute $P_h = [F_0BY, F_1BY, \ldots, F_{g-1}BY]$ (see (5)) and the minimum singular value $\sigma_{\min}$ of P_h. If $\sigma_{\min}$ is not small, then h is increased until $\sigma_{\min}$ of the updated P_h is small enough. Similar to this idea, here we give a way to determine an estimator t of an upper bound of s for our Algorithm 3. The rationale behind our idea is as follows.

Let $s^\dagger$ be a positive integer and $Y_{s^\dagger} \sim \mathsf{N}_{n\times s^\dagger}(0,1)$. Consider

$$U_{s^\dagger} = QY_{s^\dagger} = \frac{1}{2\pi\sqrt{-1}}\oint_\Gamma (zB-A)^{-1}BdzY_{s^\dagger} = S_{(:,1:s)}(S^{-1})_{(1:s,:)}Y_{s^\dagger}.$$

Then $U_{s^\dagger} \in \mathbb{C}^{n\times s^\dagger}$ is the projection of $Y_{s^\dagger}$ onto $\mathcal{K}$, and consequently, $\mathrm{rank}(U_{s^\dagger}) \leq s$. With this in mind, we see that if $\mathrm{rank}(U_{s^\dagger}) = s^\dagger$, then we have $s^\dagger \leq s$; and we will increase $s^\dagger$ and repeat the process. Otherwise, if $\mathrm{rank}(U_{s^\dagger}) < s^\dagger$, we can conclude that $s = \mathrm{rank}(U_{s^\dagger})$ with the help of Lemma 1; and thereby $s < s^\dagger$. Below we give the algorithm for finding t.

Algorithm 4

Input an increasing factor $\alpha > 1$, the size p of sample vectors and a threshold δ. The function "SEARCH" outputs t (an estimator of an upper bound of s) and the projection matrix $U_1 \in \mathbb{C}^{n\times t}$ onto $\mathcal{K}$.

Function $[U_1, t] = \textsc{Search}(A, B, \Gamma, \alpha, p)$

1. Pick $Y_0 \sim \mathsf{N}_{n\times p}(0,1)$ and compute $U = \frac{1}{2\pi\sqrt{-1}}\oint_\Gamma (zB-A)^{-1}BY_0dz$ approximately by the Gauss-Legendre quadrature rule .
2. Set $s_0 = \lceil \frac{1}{p}\mathrm{trace}(Y_0^*U)\rceil$ and $s^\dagger = \min(\max(p, s_0), n)$.
3. If $s^\dagger > p$
4. Pick $\hat{Y} \sim \mathsf{N}_{n\times(s^\dagger-p)}(0,1)$ and compute $\hat{U} = \frac{1}{2\pi\sqrt{-1}}\oint_\Gamma (zB-A)^{-1}B\hat{Y}dz$ approximately the Gauss-Legendre quadrature rule.
5. Augment V to U to form $U = [U, \hat{U}] \in \mathbb{C}^{n\times s^\dagger}$.
6. Else
7. Set $s^\dagger = p$.
8. End
9. Compute the rank-revealing QR decomposition [10, 19] of U with column pivoting strategy: $U\Pi = [U_1, U_2]\begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22}\end{bmatrix}$, here $\|R_{22}\|_2 \leq \delta$.
10. Set $t = \mathrm{rank}(R_{11})$. If $t < s^\dagger$, stop. Otherwise, set $p = t, U = U_1$ and $s^\dagger = \lceil \alpha t\rceil$. Then go to Step 3.

In Line 9, we use the rank-revealing QR decomposition method [10, 19] to detect whether U is numerically rank deficient. If it is, then we have found an estimator t of an upper bound. We remark that Algorithm 4 can be treated as the first iteration of Algorithm 3 because U_1 from Algorithm 4 is an approximate projection onto $\mathcal{K}$, therefore it can be taken as the U in line 2 of Algorithm 3.

DOI: 10.1002/nla

4.2. The Stopping Criteria

Although by Algorithm 4 we can obtain an estimator of an upper bound t for s, the actual value of s is still unknown. Thus Algorithm 3 is still impractical because it is hard to determine whether all desired eigenpairs are captured. Below we present simple but efficient stopping criteria to detect this. It also gives the accuracy that our algorithm can achieve.

In Algorithm 3, there are t eigenvalues being solved in each iteration, and hence s of them are the eigenvalues we seek and $(t-s)$ of them are spurious eigenvalues. Those spurious eigenvalues outside Γ can easily be detected by checking the values of their coordinates. It is only the spurious eigenvalues that are inside Γ that we need to pay special attention. In Algorithm 3, the accuracy of computed eigenpairs $(\lambda_i, \mathbf{x}_i)$ are measured by the residual norms

$$r_i = \frac{\|A\mathbf{x}_i - \lambda_i B\mathbf{x}_i\|_2}{\|A\mathbf{x}_i\|_2 + \|B\mathbf{x}_i\|_2}, \quad 1 \le i \le t. \tag{30}$$

It was shown in [44, Theorem 5.2] that the residual norms r_i of the true eigenpairs decrease monotonically in each iteration in the FEAST algorithm. Since Algorithm 3 is essentially a generalization of the FEAST algorithm to the non-Hermitian cases, we also observed numericlly that the residual norms (30) of desired eigenpairs decrease monotonically too as iteration progresses in Algorithm 3.

For the spurious eigenvalues appearing inside Γ, they cannot achieve very small residual norms (otherwise, they should be counted as true eigenvalues). Consequently, starting from some iteration, there will be a gap in the accuracy between the desired eigenvalues and the spurious eigenvalues. Based on this fact, we use a filtering tolerance to detect the number of desired eigenvalues inside Γ. We refer to the eigenvalues inside Γ whose residual norms are smaller than the filtering tolerance as our filtered eigenvalues. We observe numerically that after some iterations, the numbers of filtered eigenvalues will remain unchanged. So in our method, we determine the number of desired eigenvalues s by monitoring the numbers of filtered eigenvalues in two consecutive iterations. If the two numbers are the same, then we set the number to be s. As for the filtering tolerance, we take it to be 10^{-3} in our experiments. We find that for such tolerance, the number of filtered eigenvalues can be determined after one or two iterations, see Experiment 5.2.

The algorithm will stop when all filtered eigenvalues meet the user-prescribed tolerance ϵ, see (26). To avoid the situation where ϵ is set too small for the given problem, we also stop the iterations when (i) the iteration number reaches a prescribed maximum `max_iter`, or (ii) the overall accuracy of the s eigenvalues is not improved from one iteration to the next.

More detailed demonstration about the idea behind the stopping criteria will be given in Section 5.

4.3. The Complete Algorithm

We now present the complete algorithm based on Algorithm 3, Algorithm 4, and the above stopping criteria. Then we discuss some implementation issues pertaining to the algorithm.

Algorithm 5
Input filtering tolerance η for detecting the spurious eigenvalues, and the tolerance ϵ for the accuracy of the eigenpairs. The function "GFEAST" computes all the eigenvalues λ of (1) inside Γ and their associated eigenvectors $\mathbf{x}$. The computed λ and $\mathbf{x}$ are stored in vector Λ and matrix X respectively. The `flag` is set to 1 if there are s eigenpairs $(\lambda, \mathbf{x})$ satisfying (30), 0 if the overall accuracy is not improved from one iteration to the next; -1 if the maximum number of iterations `max_iter` is reached.

Function $[\Lambda, X, \texttt{Res}, \texttt{flag}] = \text{GFEAST}(A, B, \Gamma, \alpha, p, \epsilon, \eta, \texttt{max\_iter})$
1. Call $[U_1, t] = \text{SEARCH}(A, B, \Gamma, \alpha, p)$ to obtain an estimator t to bound the exact number s of the eigenvalues inside Γ, and a projection U_1 onto $\mathcal{K}$.
2. Compute the QR decomposition: $BU_1 = \tilde{U}_1\tilde{R}_1$.
3. Set $e(0) = 0$ and $c(0) = 0$.

DOI: 10.1002/nla

4. For $k = 1, 2, \cdots,$ `max_iter`
5. Form $\tilde{A} = (\tilde{U}_k)^* A U_k$ and $\tilde{B} = (\tilde{U}_k)^* B U_k$.
6. Solve the projected eigenproblem $\tilde{A}\mathbf{y} = \lambda \tilde{B}\mathbf{y}$ of size t to obtain eigenpairs $\{(\lambda_i, \mathbf{y}_i)\}_{i=1}^{t}$. Set $\mathbf{x}_i = U_k \mathbf{y}_i, i = 1, 2, \ldots, t$.
7. Set $r = [\,]$, $\Lambda^{(k)} = [\,]$, $X^{(k)} = [\,]$ and $c(k) = 0$.
8. For $i = 1 : t$
9. Compute $r_i = \|A\mathbf{x}_i - \lambda_i B\mathbf{x}_i\|_2 / (\|A\mathbf{x}_i\|_2 + \|B\mathbf{x}_i\|_2)$.
10. If λ_i inside Γ and $r_i < \eta$, then $c(k) = c(k) + 1$, $r = [r, r_i]$, $X^{(k)} = [X^{(k)}, \mathbf{x}_i]$ and $\Lambda^{(k)} = [\Lambda^{(k)}, \lambda_i]$.
11. End
12. Set $e(k) = \max(r)$.
13. If $c(k) = c(k-1)$ and $e(k) < \epsilon$, output $\Lambda = \Lambda^{(k)}$ and $X = X^{(k)}$, `Res` $= e(k)$, `flag` $= 1$. Stop.
14. If $c(k) = c(k-1)$ and $e(k) > e(k-1)$, output $\Lambda = \Lambda^{(k-1)}$ and $X = X^{(k-1)}$, `Res` $= e(k-1)$, `flag` $= 0$. Stop.
15. If $k =$ `max_iter`, output $\Lambda = \Lambda^{(k)}$ and $X = X^{(k)}$, `Res` $= e(k)$, `flag` $= -1$. Stop.
16. Compute $U^{(k+1)} = \frac{1}{2\pi\sqrt{-1}} \oint_\Gamma (zB - A)^{-1} B dz U_k$ approximately.
17. Compute QR decompositions: $U^{(k+1)} = U_{k+1} R_{k+1}$ and $BU^{(k+1)} = \tilde{U}_{k+1} \tilde{R}_{k+1}$.
18. End

Below we give some remarks on Algorithm 5.

1. Since the columns of the matrix U_1 obtained from the function SEARCH are orthonormalized, we only need to compute the QR decomposition for BU_1 in line 2.
2. The orthonormal base U_k and $\tilde{U}_k$ for the search and the test subspaces, respectively, in the kth iteration are not B-orthonormal, that is they do not satisfy the relationship $\tilde{U}_k^* B U_k = I_t$.
3. In line 10, $c(k)$ is the number of filtered eigenvalues in the kth iteration. The filtered eigenvalues are the ones inside Γ whose accuracy is less than the filtering tolerance η. We consider the filtered eigenvalues to be the desired eigenpairs of (1).
4. Lines 13 to 15 are the three stopping criteria. In line 13, we stop when $e(k)$, the overall accuracy of all s desired eigenvalues, is less than ϵ. In line 14, we stop when $e(k)$ is not improved from the $(k-1)$th iteration to the kth iteration. In line 15, we stop when the maximum number of iterations is reached.

In each iteration, the dominant work is to compute the projection U. In the case when Γ is a circle with center γ and radius ρ, we compute the contour integral in line 16 by using the q-point Gauss-Legendre quadrature on Γ. More precisely,

$$U^{(k+1)} = \frac{1}{2\pi\sqrt{-1}} \oint_\Gamma (zB - A)^{-1} B dz U_k \approx \frac{1}{2} \sum_{j=1}^{q} \omega_j (z_j - \gamma)(z_j B - A)^{-1} B U_k, \tag{31}$$

where $z_j = \gamma + \rho e^{\sqrt{-1}\theta_j}$, $\theta_j = (1 + t_j)\pi$, and t_j is the jth Gaussian node with associated weight ω_j. Accordingly, it requires us to solve q generalized shifted linear systems of the form

$$(z_j B - A)\widehat{U}_j = BU_k \quad z_j \in \mathbb{C}, \quad 1 \le j \le q. \tag{32}$$

When Γ is an irregular closed curve, we can choose a circle $\widetilde{\Gamma}$ such that $\widetilde{\Gamma}$ encloses Γ. We compute all eigenpairs inside $\widetilde{\Gamma}$ and then determine the eigenpairs that are indeed inside Γ.

Like other contour-integral based methods, our algorithm replaces the difficulty of solving the eigenvalue problem (1) by the difficulty of solving the linear systems (32). One has considerable freedom in choosing the approach to solve (32) based on the properties of the matrices in (32), such as the Krylov subspace based methods [30, 40, 41]. Since (32) are generalized shifted systems with multiple right-hand sides, the direct methods, such as the sparse Gaussian LU factorization,

DOI: 10.1002/nla

Table I. Test problems from Matrix Market that are used in our experiments.

No.	Matrix	Size	nnz	Property	condest
1	A: BFW398A	398	3678	unsymmetric	7.58×10^3
	B: BFW398B	398	2910	symmetric indefinite	3.64×10^1
2	A: BFW782A	782	7514	unsymmetric	4.63×10^3
	B: BFW782B	782	5982	symmetric indefinite	3.05×10^1
3	A: PLAT1919	1919	17159	symmetric indefinite	1.40×10^{16}
	B: PLSK1919	1919	4831	skew symmetric	1.07×10^{18}
4	A: BCSSTK13	2003	42943	symmetric positive definite	4.57×10^{10}
	B: BCSSTM13	2003	11973	symmetric positive semi-definite	Inf
5	A: BCSSTK27	1224	28675	symmetric positive definite	7.71×10^4
	B: BCSSTM27	1224	28675	symmetric indefinite	1.14×10^{10}
6	A: MHD3200A	3200	68026	unsymmetric	2.02×10^{44}
	B: MHD3200B	3200	18316	symmetric indefinite	2.02×10^{13}
7	A: MHD4800A	4800	102252	unsymmetric	2.54×10^{57}
	B: MHD4800B	4800	27520	symmetric indefinite	1.03×10^{14}

are also highly recommended. Notice that once we obtain the LU factors, they can be reused when we solve (32) in the subsequent iterations. Moreover, since the quadrature nodes $z_j, j = 1, \ldots, q$, are independent, and the columns of the right-hand sides are also independent, our algorithm has a good potential to be parallelized.

5. NUMERICAL EXPERIMENTS

In this section, we give some numerical experiments to illustrate the efficiency of our method for computing the eigenpairs of (1) inside a given contour Γ. All computations are carried out in MATLAB version R2012b on a MacBook with an Intel Core i5 2.5 GHz processor and 8 GB RAM. The test matrices are from the Matrix Market collection [9]. They are the real-world problems from scientific and engineering applications. In fact, Problem 2 serves as a classic testbed for generalized non-Hermitian eigenproblem [4]. The eigenvalues of Problem 3 occur in pairs, and hence the problem is well-known as a difficult eigenproblem [9]. The matrix B of Problem 4 is singular and Problems 6 and 7 are ill-conditioned. The descriptions of these matrices are presented in TABLE I, where nnz denotes the number of non-zero entries and `cond` denotes their condition numbers which are computed by MATLAB function `condest`.

In the numerical comparisons, we assume that the eigenvalues and eigenvectors computed by the MATLAB function `eig` in dense format are the accurate ones. We use Gauss-Legendre quadrature rule [12] with $q = 16$ quadrature points on Γ to compute the contour integrals (31). As for solving the generalized shifted linear systems of the form (32), we first use the MATLAB function `lu` to compute the LU decomposition of $A - z_j B, j = 1, 2, \ldots, q$, and then perform the triangular substitutions to get the corresponding solutions.

Experiment 5.1 (Finding an estimator of an upper bound for s): As stressed in the introduction and in Section 3, the information about the number of eigenvalues s inside Γ is crucial to the success of contour-integral based methods—they all need a parameter t satisfying $t \geq s$ to start the program with. Our Algorithm 4 is devoted to finding an estimator t of an upper bound of s. In this experiment, we test how accurate the computed t are. In the algorithm, the size p of sample vectors is set as 50, the increasing factor α is chosen to be 1.5, and the threshold δ is set to be 10^{-6}. TABLE II presents the results for the test problems in TABLE I. In TABLE II, the parameters γ and ρ denote the center and the radius of the circle Γ of each test problem respectively, s denotes the true number of eigenvalues inside Γ as obtained from MATLAB (by computing all eigenvalues and selecting those inside Γ), s_0 is the initial estimate obtained by the trace formula (see (27) or line 2 in Algorithm 4), and t is the estimator of an upper bound that our Algorithm 4 gives.

From TABLE II, we see that for the first five well-conditioned problems, the estimates s_0 are good approximations of s though it can underestimate s as in Problem 1. However for the ill-conditioned Problems 6 and 7, s_0 are not good—it underestimates and overestimates s by large

DOI: 10.1002/nla

Table II. Results of Algorithm 4: s is the exact number of eigenvalues inside Γ as computed by MATLAB, s_0 is the estimate of s by using the trace formula (27) and t is the estimator of an upper bound computed by our Algorithm 4.

No.	γ	ρ	s	s_0	t
1	-5.0×10^5	2.0×10^5	123	122	137
2	-6.0×10^5	3.0×10^5	230	231	262
3	0	1.0×10^{-3}	270	277	328
4	0	6.0×10^5	172	173	183
5	5.0×10^3	2.0×10^3	107	107	118
6	-4.0×10^1	3.0×10^1	162	118	178
7	-6.0	3.0	169	3667	186

Table III. Number of iterations required to get the correct s for different η's.

η	No.1	No.2	No.3	No.4	No.5	No.6	No.7
10^{-3}	1	2	1	2	2	2	2
10^{-5}	2	2	2	2	3	3	3
10^{-7}	2	3	3	3	3	4	3

margins. However our Algorithm 4 gives a good estimator t of an upper bound of s in all seven problems.

Experiment 5.2 (Stopping criteria): Here we illustrate the convergence behavior of our algorithm and examine the stopping criteria we propose.

In each iteration, t eigenvalues are computed; consequently, there are $(t-s)$ spurious eigenvalues. Since the spurious eigenvalues cannot achieve very small residual norms, in our method we use a filtering tolerance η to distinguish them. The eigenvalues with residual norms r_i (see (30)) less than η are referred as filtered eigenvalues, and the remaining are considered spurious.

In our algorithm, if the number of filtered eigenvalues are the same in two consecutive iterations, then we consider that number to be the number of desired eigenvalues we seek. TABLE III gives the number of iterations needed to determine this number for a given η's. For example, in test problem No. 2, with $\eta = 10^{-3}$, the number of filtered eigenvalues is the same in the 2nd and 3rd iterations. So we use that number of filtered eigenvalues to be the number of desired eigenvalues. We remarked that in all the cases we tested in TABLE III, once the number stays the same for two consecutive iterations, it stays the same for all subsequent iterations. Moreover, the number of desired eigenvalues we found are exactly the same as the true number of eigenvalues s as computed by MATLAB. For efficiency, in the following experiments, we set $\eta = 10^{-3}$.

In FIG. 1, we show the convergence history of the s desired eigenvalues by plotting their maximum residual norms $\texttt{Res} := \max_{1\le i\le s} r_i$, starting from the iterations where s is first determined until the 10th iteration (the iteration numbers where s is first determined are given in the first row of TABLE III). We see from FIG. 1 that Res decreases monotonically and dramatically in the first few iterations for all test problems. Then it levels off for the first five problems while it rebounds for the ill-conditioned problems Nos. 6 and 7. Based on the convergence behavior, we stop our algorithm as follows:

(i) Res in the current iteration is less than a given tolerance ϵ (line 13 in Algorithm 5),
(ii) Res starts to increase again from one iteration to the next (line 14 in Algorithm 5), or
(iii) the maximum number of iterations (line 15 in Algorithm 5) is reached.

Experiment 5.3 (Comparisons with the dual-subspace FEAST algorithm): We have shown that the FEAST algorithm may fail when (1) is non-Hermitian (see Example 3.1). The main goal of our work is to make the FEAST algorithm also applicable for the non-Hemitian cases. To this end, we have used the oblique projection technique to extract desired eigenpairs. In this experiment, we compare our method with the recently developed dual-subspace FEAST algorithm (Dual FEAST) [27, 29], which also aims to extend FEAST to non-Hermitian problems.

The search subspaces in both our method and the dual-subspace FEAST algorithm are $\mathrm{span}\{U\}$ (see (17)). The key difference between the two non-Hermitian FEAST algorithms is the choice of

DOI: 10.1002/nla

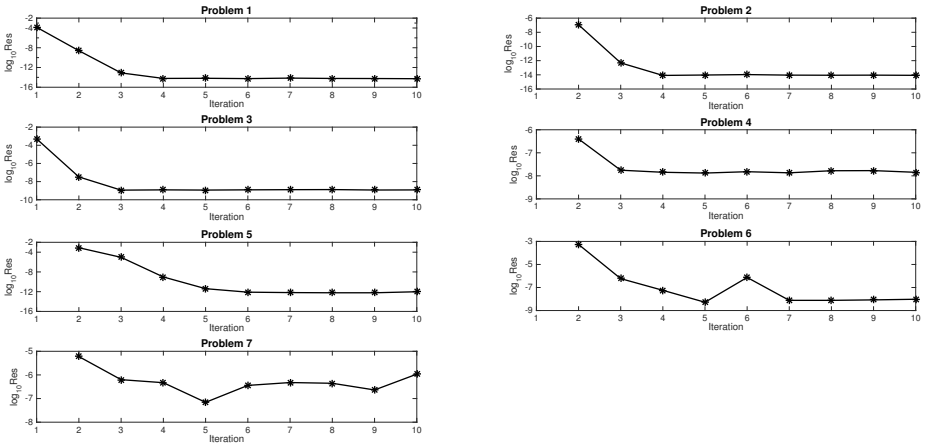

Figure 1. The maximum relative residual norms in different iterations.

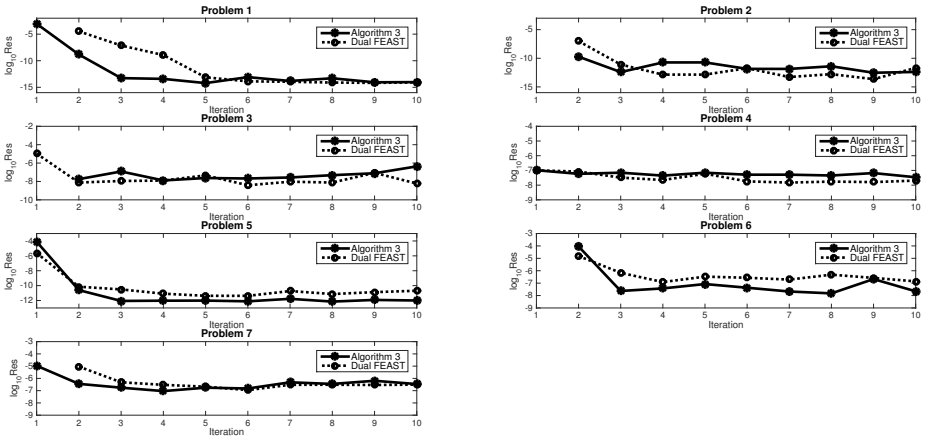

Figure 2. The convergence behavior of our algorithm and the dual-subspace FEAST algorithm.

the test subspace. In our method, the test subspace is taken to be $\mathrm{span}\{BU\}$, while in the dual-subspace FEAST algorithm the test subspace is constructed using the contour-integral technique so as to contain the left eigenspace corresponding to desired eigenvalues. As a result, the dual-subspace FEAST algorithm costs about twice as much as our method in each iteration, if only the eigenvalues or the eigenvalues along with their right eigenvectors are needed. In this experiment, we compare the numerical performance of the two non-Hermitian FEAST algorithm by their convergence behavior.

The dual-subspace FEAST algorithm also needs to know the number of target eigenvalues, s, in advance in order to determine a suitable size for the starting vectors Y. In the test we assume the value of s is known and set the number of column of Y to be $t = \lceil 1.3s \rceil$ for the dual-subspace FEAST algorithm, as well as for our method. As a result, the only difference between the two test methods is the test subspace.

FIG. 2 depicts the maximum residual norms $\texttt{Res} := \max_{1\le i\le s} r_i$ for the first 10 iterations for both methods. We can see that both methods have similar convergence behavior and can achieve almost the same accuracy. Thus the cost our method is about half of that of the dual-subspace

DOI: 10.1002/nla

Table IV. Comparison of `eig` and our method.

No.	`eig`		our method	
	`Res`	Time (sec.)	`Res`	Time (sec.)
1	6.75×10^{-14}	0.67	5.85×10^{-15}	1.18
2	3.83×10^{-14}	10.22	2.22×10^{-14}	4.77
3	4.77×10^{-9}	116.12	5.78×10^{-10}	17.74
4	2.52×10^{-6}	269.56	3.36×10^{-9}	33.94
5	1.25×10^{-11}	48.83	1.87×10^{-13}	6.25
6	5.76×10^{-7}	561.29	3.27×10^{-9}	30.70
7	1.44×10^{-7}	2725.43	2.23×10^{-8}	92.74

FEAST algorithm, if only the eigenvalues or the eigenvalues together with their right eigenvectors are needed.

Experiment 5.4 (Comparisons with other methods): Here we compare our method with three other methods both in terms of accuracy and timing.

We first compare our method with MATLAB function `eig`. The goal is to examine the accuracy that our method can achieve. We set $\epsilon = 10^{-16}$ and `max_iter` $= 10$ for our method. For all test problems, our method stops in Line 14 before reaching `max_iter`.

The comparison of `eig` and our method are listed in TABLE IV. It is clear that our algorithm can achieve higher accuracy when compared with its counterpart. In terms of CPU time, except for Problem 1 where the size is small, our algorithm runs significantly faster than `eig`—though we should stress that `eig` has to compute all eigenvalues while our method computes only those inside Γ.

Finally we compare our method with Beyn's method in [8] and the block version of the CIRR method (BLOCK_CIRR), i.e., Algorithm 1. Beyn's method, which is also a contour-integral based eigensolver, was originally proposed for nonlinear eigenproblems. So it can be used to solve (1). In the test we apply an iterative refinement scheme recently developed in [37] for BLOCK_CIRR. The three test methods are contour-integral based eigensolvers, hence they need to know the number of eigenvalues, s, before starting. In our method, we provide a way to find an estimator of an upper bound of s. As for Beyn's method and BLOCK_CIRR, here we assume that s is known and set $t = \lceil 1.5s \rceil$ for each test problem. Some other parameters in Beyn's method and BLOCK_CIRR need to be specified. The parameter tol_{rank} in Beyn's method used for rank test is taken to be 10^{-3}. Beyn's method is noniterative in nature, while the other two contour-integral based test methods improve the accuracy of computed eigenpairs iteratively. For the sake of fairness, we initially set the number of quadrature nodes $q = 60$ for Beyn's method. If the computed eigenpairs do not attain the prescribed accuracy, we increase the value of q by 20 and redo the test for Beyn's method. This process is terminated when either the prescribed accuracy is achieved or q reaches the value of 600. In BLOCK_CIRR, we take the parameters $q = 16$ and `max_iter` $= 10$, which are the same with those in our method. The parameter g in BLOCK_CIRR is taken to be 10.

The prescribed accuracy ϵ is 10^{-8} for all three methods. The numerical results are reported in TABLE V. Remarkably, we see that Beyn's method and BLOCK_CIRR fail to compute all desired eigenpairs in Problems 6 and 7, respectively, where the matrices involved are ill-conditioned. We recall that the matrix B in Problem 4 is singular, Beyn's method and BLOCK_CIRR are unable to achieve the prescribed accuracy for this test problem. Especially, the latter can only achieve about four digits accuracy. On the other hand, we see that our method can attain the desired accuracy for all test problems except for the last one. Therefore, our algorithm is more accurate and stable when compared to Beyn's method and BLOCK_CIRR.

When it comes to timing, which is measured in seconds, it is shown in TABLE V that Beyn's method is most costly, and BLOCK_CIRR outperforms our method. The dominant computational cost of three methods are the solution of a set of linear systems of the form in (32). The numbers of such linear systems needed to solve in each iteration for both BLOCK_CIRR and our method are $q = 16$. But the number of right-hand sides in the BLOCK_CIRR is $\lceil \frac{1.5s}{10} \rceil$ in the tests here, while in our method, it is t, which is an estimator of an upper bound of s. Consequently, the block-CIRR

DOI: 10.1002/nla

Table V. Comparison of Beyn's method, BLOCK_CIRR and our method.

No.	Beyn's method			BLOCK_CIRR		our method	
	Res	Time	q	Res	Time	Res	Time
1	8.67×10^{-10}	1.72	160	9.76×10^{-10}	0.46	7.32×10^{-14}	0.75
2	1.87×10^{-9}	7.17	160	2.09×10^{-9}	2.38	4.77×10^{-12}	3.53
3	9.32×10^{-9}	253.84	500	2.91×10^{-6}	13.22	5.74×10^{-9}	14.96
4	4.39×10^{-8}	355.27	600	2.96×10^{-4}	19.54	5.34×10^{-9}	29.42
5	6.66×10^{-10}	58.66	320	1.06×10^{-9}	1.42	8.77×10^{-12}	4.17
6	—	151.96	600	2.85×10^{-4}	13.55	4.85×10^{-9}	19.43
7	3.17×10^{-6}	200.66	600	—	16.14	2.23×10^{-8}	92.74

method always requires less CPU time than our method. As with our method, the number of right-hand sides in Beyn's method also must not be less than s. In our experiment, it is taken to be $\lceil 1.5s \rceil$. The number of quadrature nodes required by Beyn's method are reported in TABLE V, from which we see that Beyn's method needs a large number of quadrature nodes to achieve the prescribed accuracy. As a result, Beyn's method needs to solve more linear systems than our method. Hence Beyn's method is more expensive than our method, as is shown in TABLE V.

6. CONCLUSIONS

We have developed a contour-integral based method which extends the FEAST algorithm to non-Hermitian problems. It can compute the eigenvalues lying inside a given region in the complex plane and their associated eigenvectors. To extract the desired eigenpairs, we use the oblique projection technique with appropriately chosen test subspace rather than the Rayleigh-Ritz procedure. The numerical experiments illustrate that our algorithm is fast and can achieve high accuracy. We also provide a way to find an estimator of an upper bound of the number of eigenvalues inside the contour, and give stopping criteria to capture the desired eigenvalues. Our algorithm is easily parallelizable. How to further improve its numerical performance, and to extend it to computing the left and right eigenvectors simultaneously will be our future work.

ACKNOWLEDGEMENTS

We would like to thank Dr. Peter P. T. Tang who introduced the FEAST algorithm to us. We would also like to thank Professor Tetsuya Sakurai for many fruitful discussions and providing us the codes of the BLOCK_CIRR method.

This work is supported in part by National Basic Research Program of China (973 Program) Grant No. 2015CB352501, Shenzhen Project Grant No. JCYJ 20140901003939012, HKRGC GRF Grant No. CUHK400412, HKRGC CRF Grant No. CUHK2/CRF/11G, HKRGC AoE Grant AoE/M-05/12, CUHK DAG No. 4053007, and CUHK FIS Grant No. 1902036.

REFERENCES

1. Ahlfors L. *Complex Analysis* (3rd edn), McGraw-Hill Education: New York, 1979.
2. Anderson A, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, and Sorensen D. *LAPACK Users' Guide* (3rd edn). Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 1999.
3. Austin AP and Trefethen LN. *Computing eigenvalues of real real symmetric matrices with rational filters in real arithmetic*. SIAM Journal on Scientific Computing 2015; **37**: A1365–A1387.
4. Bai Z, Demmel J, Dongarra J, Ruhe A, and Van Der Vorst A. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 2000.
5. Bai Z, Demmel J, and Gu M. *An inverse free parallel spectral divide and conquer algorithm for nonsymmetric eigenproblem*. Numerische Mathematik 1997; **76**: 279–308.
6. Beckermann B, Golub GH, and Labahn G. *On the numerical condition of a generalized Hankel eigenvalue problem*. Numerische Mathematik 2007; **106**: 41–68.
7. Berns-Müller J and Spence A. *Inexact inverse iteration with variable shift for nonsymmetric generalized eigenvalue problems*. SIAM Journal on Matrix Analysis and Applications 2006; **28**: 1069–1082.
8. Beyn WJ. *An integral method for solving nonlinear eigenvalue problems*. Linear Algebra and its Applications 2012; **436**: 3839–3863.

9. Boisvert RF, Pozo R, Remington K, Barrett R, and Dongarra J. *The matrix market: A web repository for test matrix data*. The Quality of Numerical Software, Assessment and Enhancement, R. Boisvert, ed., Chapman & Hall, London, 1997.
10. Chan TT. *Rank revealing QR factorizations*. Linear Algebra and its Applications 1987; **88-89**: 67–82.
11. Chu D, Liao LZ, and Ng MK. *Sparse orthogonal linear discriminant Analysis*. SIAM Journal on Scientific Computing 2012; **34**: A2421–A2443.
12. Davis PJ and Rabinowitz P. *Methods of numerical integration* (2nd edn). Academic Press: Orlando, FL, 1984.
13. Demmel J. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 1997.
14. Di Napoli E, Polizzi E, and Saad Y. *Efficient estimation of eigenvalue counts in an interval*. http://arxiv.org/abs/1308.4275.
15. Ford B and Hall G. *The generalized eigenvalue problem in quantum chemistry*. Computer Physics Communications 1974; **8**: 337–348.
16. Futamura Y, Tadano H, and Sakurai T. *Parallel stochastic estimation method of eigenvalue distribution*. JSIAM Letters 2010; **2**:127–130.
17. Gallivan K, Grimme E, and Van Dooren P. *A rational Lanczos algorithm for model reduction*. Numerical Algorithms 1996; **12**: 33–64.
18. Gantmacher FR. *The Theory of Matrices*. Chelsea Publishing Company: New York, 1959.
19. GH and Van Loan CF. *Matrix Computations* (3rd edn). Johns Hopkins University Press: Baltimore, MD, 1996.
20. Hutchinson MF. *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*. Communications in Statistics – Simulation and Computation 1990; **19**: 433–450.
21. Grimes RG, Lewis JD, and Simon HD. *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*. SIAM Journal on Matrix Analysis and Applications 1994; **15**: 228–272.
22. Imakura A, Du L, and Sakurai T. *A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems*. Applied Mathematics Letters 2014; **32**: 22–27.
23. Imakura A, Du L, and Sakurai T. *Error bounds of Rayleigh-Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems*. Numerical Algorithms 2016; **71**: 103–120.
24. Ikegami T and Sakurai T. *Contour integral eigensolver for non-Hermitian systems: a Rayleigh-Ritz-type approach*. Taiwanese Journal of Mathematics 2010; **14**: 825–837.
25. Ikegami T, Sakurai T, and Nagashima U. *A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method*. Journal of Computational and Applied Mathematics 2010; **233**: 1927–1936.
26. Joubert WD. *Lanczos methods for the solution of nonsymmetric systems of linear equations*. SIAM Journal on Matrix Analysis and Applications 1992; **13**:926–943.
27. Kestyn J, Polizzi E, and Tang P. *FEAST eigensolver for non-Hermitian problems*. http://arxiv.org/abs/1506.04463.
28. Krämer L, Di Napoli E, Galgon M, Lang B, and Bientinesi P. *Dissecting the FEAST algorithm for generalized eigenproblems*. Journal of Computational and Applied Mathematics 2013; **244**: 1–9.
29. Laux SE. *Solving complex band structure problems with the FEAST eigenvalue algorithm*. Physical Review B 2012; **86**: 075103.
30. Meerbergen K and Bai Z. *The Lanczos method for parameterized symmetric linear systems with multiple right-hand sides*. SIAM Journal on Matrix Analysis and Applications 2010; **31**: 1642–1662.
31. Moler CB and Stewart GW. *An algorithm for generalized matrix eigenvalue problems*. SIAM Journal on Numerical Analysis 1973; **10**: 241–256.
32. Nakatsukasa Y and Higham NJ. *Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the SVD*. SIAM Journal on Scientific Computing 2013; **35**: A1325–A1349.
33. Polizzi E. *Density-matrix-based algorithm for solving eigenvalue problems*. Physical Review B 2009; **79**:115112.
34. Ruhe A. *Rational Krylov: A practical algorithm for large sparse nonsymmetric matrix pencils*. SIAM Journal on Scientific Computing 1998; **19**: 1535–1551.
35. Saad Y. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 2011.
36. Saad Y, Chelikowsky JR, and Shontz SM. *Numerical methods for electronic structure calculations of materials*. SIAM Review 2010; **52**: 3–54.
37. Sakurai T, Futamura Y, and Tadano H. *Efficient parameter estimation and implementation of a contour integral-based eigensolver*. Journal of Algorithms & Computational Technology 2013; **7**: 249–269.
38. Sakurai T and Sugiura H. *A projection method for generalized eigenvalue problems using numerical integration*. Journal of Computational and Applied Mathematics 2003; **159**: 119–128.
39. Sakurai T and Tadano H. *CIRR: A Rayleigh–Ritz type method with contour integral for generalized eigenvalue problems*. Hokkaido Mathematical Journal 2007; **36**: 745–757.
40. Simoncini V and Szyld DB. *Recent computational developments in Krylov subspace methods for linear systems*. Numerical Linear Algebra with Applications 2007; **14**: 1–59.
41. Sogabe T, Hoshi T, Zhang SL, and Fujiwara T. *Solution of generalized shifted linear systems with complex symmetric matrices*. Journal of Computational Physics 2012; **231**: 5669–5684.
42. Sriperumbudur BK, Torres DA, and Lanckriet GRG. *A majorization-minimization approach to the sparse generalized eigenvalue problem*. Machine Learning 2011; **85**: 3–39.
43. Stewart GW. *Matrix Algorithms, Vol. II, Eigensystems*. Society for Industrial and Applied Mathematics (SIAM): Philadelphia, PA, 2001.
44. Tang P and Polizzi E. *FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection*. SIAM Journal on Matrix Analysis and Applications 2014; **35**: 354–390.
45. Virnik E. *Stability analysis of positive descriptor systems*. Linear Algebra and its Applications 2008; **429**: 2640–2659.

DOI: 10.1002/nla