CSIAM Trans. Appl. Math.
doi: 10.4208/csiam-am.2021-0005

Vol. **3**, No. 1, pp. 1-25
March 2022

Top Eigenpairs of Large Scale Matrices

Mu-Fa Chen[1,2,*] and Rong-Rong Chen[3]

[1] *RIMS, Jiangsu Normal University, Xuzhou, 221116, China.*
[2] *School of Mathematical Sciences and Key Laboratory of Mathematics, Beijing Normal University, Beijing 100875, China.*
[3] *Department of Electrical and Computer Engineering, University of Utah, UT 84112, USA.*

Received 28 December 2021; Accepted 16 January 2022

Abstract. This paper is devoted to the study of an extended global algorithm on computing the top eigenpairs of a large class of matrices. Three versions of the algorithm are presented that includes a preliminary version for real matrices, one for complex matrices, and one for large scale sparse real matrix. Some examples are illustrated as powerful applications of the algorithms. The main contributions of the paper are two localized estimation techniques, plus the use of a machine learning inspired approach in terms of a modified power iteration. Based on these new tools, the proposed algorithm successfully employs the inverse iteration with varying shifts (a very fast "cubic algorithm") to achieve a superior estimation accuracy and computation efficiency to existing approaches under the general setup considered in this work.

AMS subject classifications: 15A18, 65F15

Key words: Matrix eigenpair, extended global algorithm, localized estimation technique, top eigenpair, large sparse matrix.

1 Introduction: Extended global algorithm

The top eigenpairs of matrix play an important role in many fields. In particular, for the maximal eigenpair for instance, there are well-known algorithms in several different fields. For web-search, it is called PageRank. For economic optimization, there is so called left-positive eigenvector method (cf. [1; Chapter 10]). For statistics, there is principal component analysis (abbrev. PCA) which is also used in quantum mechanics computation (quantum chemistry in particular) and AI. In the last case, one needs not only the maximal one, but also a couple of the subsequent eigenpairs. Certainly, for such a well-developed field, there are some powerful algorithms in common use, the "singular value decomposition" (abbrev. SVD) for PCA for example. However, as mentioned at

*Corresponding author. *Email addresses:* mfchen@bnu.edu.cn (M.-F. Chen), rchen@utah.edu (R.-R. Chen)

the beginning of [9; p.65, §2.6]: "In some cases, SVD will not only diagnose the problem, it will also solve it, in the sense of giving you a useful numerical answer, although, as we shall see, *not necessarily 'the' answer that you thought you should get*." This happens for a number of known algorithms (see [7; Example 1] for instance) and so more careful study is valuable.

This paper is motivated by the study on the global algorithms given in [3, 7], where some effective algorithms were presented for computing the maximal eigenpair of a rather larger class of matrices. Roughly speaking, two approaches are adopted there: the power iteration (abbrev. PI) and the inverse power iteration with varying/fixed shifts (abbrev. $\mathrm{IPI}_v/\mathrm{IPI}_f$). The PI has only a little restriction on the initial vector and so has a wide range of applications. It is also economical (having lower computational complexity), but has a quite slow convergence speed, especially near the target eigenvalue. The fast convergence speed of the algorithms given in [3, 7] is mainly due to the use of IPI_v (having higher computational complexity). It is however quite dangerous if the initial is not close enough (from above) to the target eigenvalue. The last problem was avoided in [3, 7] mainly due to the assumption: the off-diagonal elements of the matrix are all nonnegative. This is essential: it implies the existence of the maximal eigenpair (as an application of the Perron–Frobenius theorem, by a shift if necessary). Then we have some important variational formulas for the upper/lower bounds of the maximal eigenvalue, i.e., the Collatz–Wielandt (abbrev. C-W) formula (cf. [2; §1 and Corollary 12]). For nonnegative matrix, the formula takes the following form:

$$\sup_{x>0}\min_k \frac{(Ax)(k)}{x(k)} = \lambda = \inf_{x>0}\max_k \frac{(Ax)(k)}{x(k)},$$

where λ is the maximal eigenvalue of the matrix A and $x(k)$ is the kth component of the vector x. The upper bound in the formula is very important in using IPI_v for avoiding the pitfalls (cf. [4; §4]). Now, a challenge appears:

Question: What can we do without the assumption of the nonnegative property of the off-diagonal elements?

A typical model led to the question is PCA, for which some of the off-diagonal elements can be negative. The question is quite serious since almost each advantage introduced in the previous paragraph is lost. We do not have the Perron–Frobenius theorem; more seriously, we do not have the C-W formula; and furthermore, the IPI_v is not practical.

Certainly, the answer to the above question is not obvious. If you have luckily produced enough courage, you may look for a way to find a substitute of the C-W formula. Assume that the given matrix A is real. Assume also for a moment that the maximal eigenvalue λ we are working is positive. Of course, at the present case, the corresponding eigenvector g is not necessarily positive, and it may have negative or zero components. Because we are now bare-handed, to find an exit from the darkness, we have to go back

to the original position: all we know is the eigenequation:

$$Ag = \lambda g. \tag{1}$$

That is, g is an eigenvector corresponding to the eigenvalue λ of A. It follows that once $g(k) \neq 0$, we must have $(Ag)(k)/g(k) > 0$, here we have preassumed that $\lambda > 0$. If a vector x produced by our iterative method (either PI or IPI) is close enough to g, then in one iteration, we have

$$x \approx g \Longrightarrow Ax \approx Ag = \lambda g.$$

We now arrive at the first localized estimation technique: *check sign and locally bilateral estimates* (abbrev. CS-LBE). Due to the property given above, on the set

$$\mathscr{N}_x := \{k\colon\ |x(k)| > 0\}, \tag{2}$$

we should have

$$\frac{Ax}{x}(k) := \frac{Ax(k)}{x(k)} > 0, \quad k \in \mathscr{N}_x, \tag{3}$$

since $\lambda > 0$ by assumption. As usual, here "$k \in \mathscr{N}_x$" means "for each $k \in \mathscr{N}_x$". The procedure checking (3) is called "*check sign*" (abbrev. CS). Once this holds for a few of iterations, then we do not need to check it again, just continue the PI until the *relative difference* (abbrev. RD)

$$1 - \min_{k \in \mathscr{N}_x} \frac{Ax}{x}(k) \Big/ \max_{k \in \mathscr{N}_x} \frac{Ax}{x}(k) < \varepsilon \tag{4}$$

for some sufficiently small ε. Under condition $\lambda > 0$, assertions (3) and (4) are actually due to the convergence of PI, assuming for a moment that the maximal eigenvalue coincides with the maximal one in modulus. In practice, one has to take care for the initial vector in using PI to guarantee its convergence. Next, using condition $\lambda > 0$ again, by (3), we have

$$\text{either} \quad (0 <) \frac{Ax}{x}(k) \leqslant \lambda \quad \text{or} \quad \frac{Ax}{x}(k) \geqslant \lambda \quad \text{for each} \quad k \in \mathscr{N}_x.$$

Hence under condition (4) with $\varepsilon \ll 1$, we obtain the following *locally bilateral estimates* (abbrev. LBE):

$$(0 \leqslant) \min_{k \in \mathscr{N}_x} \frac{Ax}{x}(k) \leqslant \lambda \leqslant \max_{k \in \mathscr{N}_x} \frac{Ax}{x}(k), \tag{5}$$

the equalities in (5) hold once x is taken to be an eigenvector of the corresponding eigenvalue λ. We now regard (5) as a substitute of the C-W upper/lower estimates, and adopt

$$z := \max_{k \in \mathscr{N}_x} \frac{Ax}{x}(k) \tag{6}$$

as an upper bound of λ for the use in IPI_v or IPI_f. Condition (4) guarantees the validity of LBE (5) and then (6). Thus, in (3)-(6), we use only those x in a small neighborhood of

the eigenvector of λ in the corresponding vector space. That is the meaning of "locally" used above.

In the above paragraph, we preassume that the maximal eigenvalue coincides with the one in modulus and is positive. This is important not only in computing the ratios above but also an essential point in the use of PI, since for which, the leading term in the algorithm is determined by the maximal eigenvalue in modulus, one cannot ignore the point "in modulus" here. Certainly, if one has known in advance that the spectrum (at least the top six eigenvalues) of A has satisfied the assumption, then the step we are working can be ignored. Otherwise, to remove the assumption, we simply use a *shift operator* : replacing A by

$$A_1 := A + \bar{\theta} I, \tag{7}$$
$$\bar{\theta} = \begin{cases} \theta & \text{if the order of } A \text{ is bigger than 6 and } \theta \text{ is an integer,} \\ \lceil \theta \rceil & \text{otherwise,} \end{cases}$$

where $\lceil x \rceil$ denotes the minimal integer that is greater or equal to x, and the constant θ is an upper bound of the spectral radius. Here, the use of $\bar{\theta}$ instead of θ is to simplify the computation. Clearly, the spectrum of A_1 is nonnegative. Therefore, working on A_1, the assumption just mentioned holds automatically. The reason that we choose the top six eigenpairs is to compare with the "eigs" package of MatLab, which is designed for the same aim (See section 4 below). Certainly, one can continue the algorithm for additional subsequent eigenpairs.

There are two ways to obtain an upper bound of the spectral radius of general complex matrix A without additional restriction. The first one is a theoretic result, deduced by the Gershgorin Circle Theorem (cf. [10]):

$$\theta = \min\{\|A\|_\infty, \|A\|_1\}, \quad \|A\|_\infty := \sup_i \sum_j |a_{ij}|, \quad \|A\|_1 = \|A^*\|_\infty,$$

where A^* denotes the transpose of A. In the symmetric case, the two terms in $\{\cdots\}$ are the same. The disadvantage of this method is that the result is usually quite rough. We now introduce the second numerical method which is similar to the technique deducing (1)-(6) above. Since we are now interested only in the modulus of the eigenvalue λ, instead of (1), we should start at

$$|Ag| = |\lambda||g|.$$

Next, we follow the analysis between (1) and (6). The output x produced by PI, with suitable initial and after enough iterations, should have the following property. With the same $\mathcal{N}_x$ defined by (2), replacing

$$\frac{Ax}{x} \quad \text{by} \quad \left|\frac{Ax}{x}\right|, \quad \lambda \quad \text{by} \quad |\lambda|, \quad \text{and} \quad z \quad \text{by} \quad \theta,$$

we obtain the analogs of (4)-(6) as follows:

$$\begin{aligned}
&1-\min_{k\in\mathscr{N}_x}\left|\frac{Ax}{x}\right|(k)\Big/\max_{k\in\mathscr{N}_x}\left|\frac{Ax}{x}\right|(k)<\varepsilon,\\
&\min_{k\in\mathscr{N}_x}\left|\frac{Ax}{x}\right|(k)\leqslant|\lambda|\leqslant\max_{k\in\mathscr{N}_x}\left|\frac{Ax}{x}\right|(k),\\
&\theta:=\max_{k\in\mathscr{N}_x}\left|\frac{Ax}{x}\right|(k).
\end{aligned}\tag{8}$$

Starting from $\mathbb{1}/\|\mathbb{1}\|$, where $\mathbb{1}$ is the constant column vector having its component 1 everywhere and $\|x\|$ is the L_2-norm of x. The resulting θ defined by (8) is what we need for (7). This method is especially good for PI, it converges economically to λ^*, the maximal eigenvalue in modulus, but not the real maximum, effective enough unless it is too close to λ^*. Hence this method is good enough for our purpose. The value of θ is noticeable since a larger θ makes the lower convergence speed of PI:

$$\lambda_1>\lambda_2>0\Longrightarrow(0,1)\ni\frac{\lambda_2+\alpha}{\lambda_1+\alpha}\uparrow\quad\text{as}\quad\alpha(>0)\uparrow.$$

We emphasize that the constant θ defined by (8) is used only in (7) for producing a matrix with nonnegative spectrum having positive six top eigenvalues. In the subsequent estimation of the eigenpairs, one does not use it again. In the special case that the given matrix already has the required property just mentioned above, one can simply ignore this shift procedure.

Usually, one needs to run the IPI$_v$ only for a few of iterations since its convergence speed is very fast. Otherwise, the calculation will overflow quickly. The computation can be finished once the output arrives at the required precision level:

$$\max_{k\in\mathscr{N}_x}\frac{Ax}{x}(k)-\min_{k\in\mathscr{N}_x}\frac{Ax}{x}(k)<\varepsilon,\tag{9}$$

the left-hand part above is called the *amplitude of LBE*. If we do not want to compute the next eigenpair, then we can stop the computations here. If otherwise, one has to improve the precise level of the output of the eigenvector. For this, one should continue the work, using IPI$_f$ instead of IPI$_v$. This is important since for computing the next eigenpairs, we will go to the subspace which is orthogonal to this eigenvector. The computation of orthogonalization often requires a higher level of precision. Failure to achieve such a precision often leads to error propagation and thus incorrect final results.

We now discuss the construction of the initial vector used by PI. First, for the maximal eigenpair, simply choose the *initial vector*

$$x_0=\mathbb{1}/\|\mathbb{1}\|.\tag{10}$$

Once the computation of the maximal eigenpair is done, we obtain the first (maximal) eigenvector, say g_1. After $k-1$ steps, we have $k-1$ eigenvectors $\{g_1,\cdots,g_{k-1}\}$ (normalized

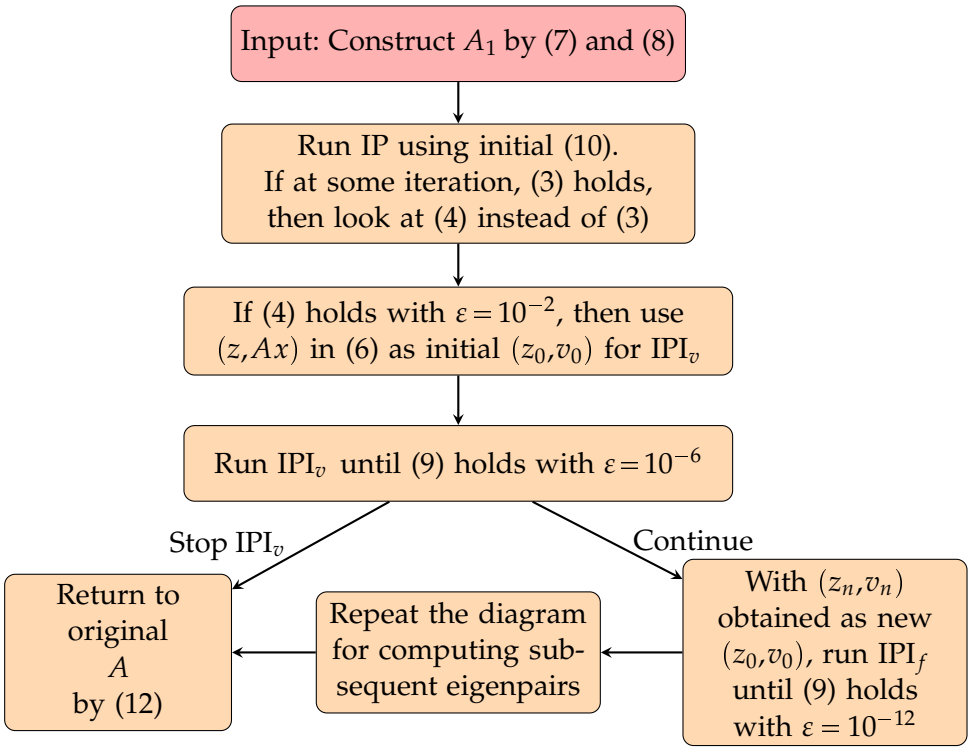


Figure 1: Flowchart of the preliminary version of the extended global algorithm.

with respect to their L_2-norm, respectively). Then the *initial vector* for computing the kth eigenpair can be chosen to be the projection vector of x_0 defined by (10) on the space which is orthogonal to Span $\{g_1, \cdots, g_{k-1}\}$. In general, for a given linear space $\mathscr{L}$, let $\mathscr{L}^\perp$ denote its orthogonal space. Then, the projection Proj (x,k) of a vector x on the space Span $\{v_1, \cdots, v_k\}^\perp$ is defined by

$$\text{Proj}(x,k) = x - \sum_{j=1}^{k} (v_j^* x) v_j \tag{11}$$

for normalized orthogonal family $\{v_1, \cdots, v_k\}$, where v^* (row vector) is the transpose of v (column vector).

To study several eigenpairs, one may assume that the matrix A has real spectrum. Otherwise, for a complex eigenpair, one may have a conjugate one. This poses some difficulty.

At the last step, return to the original matrix:

$$\text{Eigenpair } (\lambda, g) \text{ of } A_1 \rightarrow \text{Eigenpair } (\lambda - \bar{\theta}, g) \text{ of } A. \tag{12}$$

We now make some additional analysis on the preliminary version of the extended global algorithm in Fig. 1, as well as on the three algorithms used there: PI, IPI$_v$ and IPI$_f$.

While the localized estimation technique "check sign and locally bilateral estimates"(CS-LBE) mentioned above looks rather simple, the simplicity is precisely its biggest advantage — it can be applied to a rather wide range of applications, as we will see soon in the subsequent sections. The CS-LBE presents new opportunities to use techniques from a variety of fields such as optimization theory, machine learning, etc., since almost no theoretical results are available in this general setup. What we propose here is the (modified) PI. One may see a concrete example in the next section. Note that the choice of ε used for (4) or (9) in Fig. 1 may be changed according to different types of matrices used in various applications. Roughly speaking, one may use $\varepsilon \in [0.01, 0.1]$ instead of $\varepsilon = 0.01$ in (4) for medium size matrices. At this beginning step, we have used the main advantages of PI: it is safe and allows quite general initial vector, it has a good enough convergence and computing speed, except close too much to the target eigenvector.

Having the initial vector v_0 produced by the CS-LBE technique and the initial shift given by (7) at hand, we are ready to apply IPI$_v$ to accelerate the computing speed. Under the conditions (3) and (4), instead of (5), we have

$$\min_{k\in\mathcal{N}_x} \frac{Ax}{x}(k) \leqslant \frac{x^*Ax}{x^*x} \leqslant \max_{k\in\mathcal{N}_x} \frac{Ax}{x}(k).$$

Replacing the term z given on the right-hand side by the middle one $\frac{x^*Ax}{\|x\|^2}$, the IPI$_v$ becomes the so-called Rayleigh Quotient Iteration (abbrev. RQI), which is well-known a cubic algorithm (i.e., the iterative solutions generated by the algorithm converge cubically). Note that RQI is practical only if x is close enough to the target eigenvector, and hence is also a local algorithm. In particular, it is actually in a dangerous region once

$$\frac{x^*Ax}{x^*x} \in \left[\min_{k\in\mathcal{N}_x} \frac{Ax}{x}(k), \lambda\right).$$

However, since the precise local region over which the RQI is effective is not known, practical use of RQI often runs into the issue of converging to other eigenvectors that are close to the target ones. The last point is the main difference between our IPI$_v$ and RQI. The proposed IPI$_v$ ensures the algorithm robustness and allows convergence to the target eigenvector by adapting the shifts automatically. As verified by the practice in [4,7] and the subsequent sections, the difference given in (4) goes to zero very fast. then so is the difference

$$\max_{k\in\mathcal{N}_x} \frac{Ax}{x}(k) - \frac{x^*Ax}{\|x\|^2}.$$

Hence, it is believable that IPI$_v$ and RQI should have the same order of convergence speed, once RQI works.

For IPI$_f$, the initial vector v is similar to those of PI; the initial shift z of IPI$_f$ should be bigger than the target λ. Otherwise, the algorithm becomes dangerous. Certainly, IPI$_f$ is more effective if the initial pair (z,v) is closer to the target one. In Fig. 1, IPI$_f$ is used in the last step to improve the target eigenvector. For which, IPI$_v$ may no longer be practical

since the inverse matrix would be degenerated too fast. The convergence by IPI_f can be faster than PI whenever the shift is close enough to the target λ from above.

We now summarize roughly the comparison the three algorithms: PI, IPI_v and IPI_f. Let

$$\begin{aligned}
&\mathscr{D}(U) = \text{Domain of suitable initial (vector, shift) of algorithm } U,\\
&s(U) = \text{Convergence speed of algorithm } U,\\
&t(U) = \text{Computational complexity of algorithm } U.\\
&\text{From low to high is ordered by "}\prec\text{".}
\end{aligned}$$

Certainly, for PI, the shift variable is free in $\mathscr{D}(U)$. Then, roughly speaking, we have the following comparison

$$\begin{aligned}
&\mathscr{D}(\mathrm{PI}) \supset \mathscr{D}(\mathrm{IPI}_f) \supset \mathscr{D}(\mathrm{IPI}_v),\\
&s(\mathrm{PI}) \leqslant s(\mathrm{IPI}_f) \leqslant s(\mathrm{IPI}_v),\\
&t(\mathrm{PI}) \prec t(\mathrm{IPI}_f) \prec t(\mathrm{IPI}_v).
\end{aligned}$$

A mixed algorithm of PI and IPI_v was used in [4, 7]. In the present paper, we introduce some extended algorithms which have more mixture of the above three algorithms, making best use of the advantage and bypassing the disadvantage of each of these three algorithms.

The next section is an exception where the algorithm is applied to the so-called Hermitizable complex matrix, not the real one treated in most part of the paper, to illustrate the wide use of the algorithm. Certainly, from the preliminary version to more general situation, additional work is required, as shown in Section 3 by the algorithm for large scale sparse matrix. The powerful algorithm is then illustrated by two examples in Section 4. If a reader is eager to take a look at the power of the proposed algorithm introduced in the paper, he or she can skip Sections 2, 3, and go directly to Section 4.

2 Application to Hermitizable matrix

Consider the following complex matrix (cf. [5; Example 7])

$$A_0 = \begin{pmatrix}
-6 & \frac{8}{5}-\frac{6i}{5} & \frac{8}{13}+\frac{14i}{13} & \frac{18}{17}+\frac{4i}{17}\\
3+\frac{9i}{4} & -\frac{55}{4} & -\frac{5}{13}+\frac{40i}{13} & \frac{30}{17}+\frac{35i}{17}\\
\frac{12}{5}-\frac{21i}{5} & -\frac{4}{5}-\frac{32i}{5} & -13 & \frac{60}{17}-\frac{66i}{17}\\
\frac{63}{10}-\frac{7i}{5} & \frac{28}{5}-\frac{98i}{15} & \frac{70}{13}+\frac{77i}{13} & -16
\end{pmatrix}.$$

A complex matrix $A=(a_{ij})$ is called Hermitizable if there exists a positive measure $\mu=(\mu_k)$ such that $\mu_i a_{ij} = \mu_j \bar{a}_{ji}$ for each pair (i,j) (due to [5]). It is called symmetrizable in the real context. It is easy to check that A_0 is Hermitizable with respect to μ:

$$\mu_0 = 1, \quad \mu_1 = \frac{8}{15}, \quad \mu_2 = \frac{10}{39}, \quad \mu_3 = \frac{20}{119}.$$

In general, from the proof of [5; Theorem 20], it is known that a complex matrix $A=(a_{ij})$ is Hermitizable w.r.t. measure $\mu=(\mu_k)$ iff

$$A=\mathrm{Diag}(\mu)^{-1}A^H\mathrm{Diag}(\mu)\quad[A^H:=\bar{A}^*]. \tag{13}$$

Equivalently,

$$\hat{A}:=\mathrm{Diag}(\mu)^{1/2}A\,\mathrm{Diag}(\mu)^{-1/2} \tag{14}$$

is Hermitian. Clearly, the transformation of the eigenpair (λ,g) of A to the one $(\lambda,\hat{g})$ of $\hat{A}$ goes as follows.

$$(\lambda,g)\to\big(\lambda,\hat{g}=\mathrm{Diag}(\mu)^{1/2}g\big). \tag{15}$$

At the moment,

$$\hat{A}_0=\begin{pmatrix} -6 & (4-3i)\sqrt{\frac{3}{10}} & (4+7i)\sqrt{\frac{6}{65}} & (9+2i)\sqrt{\frac{7}{85}}\\ (4+3i)\sqrt{\frac{3}{10}} & -\frac{55}{4} & -\frac{2-16i}{\sqrt{13}} & (6+7i)\sqrt{\frac{14}{51}}\\ (4-7i)\sqrt{\frac{6}{65}} & -\frac{2+16i}{\sqrt{13}} & -13 & (10-11i)\sqrt{\frac{42}{221}}\\ (9-2i)\sqrt{\frac{7}{85}} & (6-7i)\sqrt{\frac{14}{51}} & (10+11i)\sqrt{\frac{42}{221}} & -16 \end{pmatrix}.$$

Due to (15), for computing the eigenpair of A_0, it suffices to study the one for $\hat{A}_0$. Hence, from now on, we need only to consider the matrix $\hat{A}_0$.

The maximal eigenpair

We now start the algorithm given in Fig. 1. The computation in this section is done by using Mathematica (version 11.3) on PC.

Step 1. *Construct* A_1. The upper bound produced by the first method given in Section 1 is $\theta\big(\hat{A}_0\big)=29.957$. We now consider the second method.

Starting at $w_0=\mathbb{1}/\|\mathbb{1}\|$ (cf. (10)) and use the following PI:

$$w_n=\hat{A}_0v_{n-1},\quad n\geqslant 1,\qquad v_n:=w_n/\|w_n\|,\quad n\geqslant 0.$$

Let

$$\begin{cases}\mathscr{N}(w)=\{k:\ |w(k)|>0\},\\ x_n=\left\{\left|\dfrac{\hat{A}_0w_n}{w_n}\right|(k),\ k\in\mathscr{N}(w_n)\right\},\\ y_n=\min\limits_{k\in\mathscr{N}(w_n)}x_n(k),\quad z_n=\max\limits_{k\in\mathscr{N}(w_n)}x_n(k).\end{cases}$$

Then in 5 iterations, the outputs are as follows.

$$\begin{aligned}&\{z_n,y_n\}_{n=1}^5\colon\ (21.2379,5.26626),(27.0853,17.7591),(27.2742,17.6156),\\ &\qquad\qquad(21.9304,17.52740),(21.6953,17.4757);\\ &\{1-y_n/z_n\}_{n=1}^5\colon\ .752035,.344325,.354132,.200772,.194493.\end{aligned}$$

Clearly, PI converges very well. Since z_4 and z_5 are closed each other, for them we have the same $\bar{\theta}=22$ which is an upper bound of the spectral radius of $\hat{A}_0$ and is obviously smaller than the one obtained by the first method. Actually, if we continue PI for more iterations,

$$z_5=21.6953,\quad z_{10}=21.5148,\quad z_{20}=21.7481,\quad z_{30}=21.4567,\quad z_{40}=21.3927,$$

then we get the same $\bar{\theta}$, since the convergence becomes rather slow when z_n is close to the modulus of the maximal eigenvalue $\lambda^*=-21.3806$. Thus by (7), we have

$$\begin{aligned}A_1&=\hat{A}_0+\bar{\theta}I\\&=\begin{pmatrix}16 & (4-3i)\sqrt{\frac{3}{10}} & (4+7i)\sqrt{\frac{6}{65}} & (9+2i)\sqrt{\frac{7}{85}}\\ (4+3i)\sqrt{\frac{3}{10}} & \frac{33}{4} & -\frac{2-16i}{\sqrt{13}} & (6+7i)\sqrt{\frac{14}{51}}\\ (4-7i)\sqrt{\frac{6}{65}} & -\frac{2+16i}{\sqrt{13}} & 9 & (10-11i)\sqrt{\frac{42}{221}}\\ (9-2i)\sqrt{\frac{7}{85}} & (6-7i)\sqrt{\frac{14}{51}} & (10+11i)\sqrt{\frac{42}{221}} & 6\end{pmatrix}.\end{aligned}$$

To justify the effectiveness of the shift used here, let us compute the eigenvalues of A_1:

$$21.8344, 12.5542, 4.24189, 0.619429.$$

It follows that there is only a little room (about 0.6) for the improvement of the shift $\bar{\theta}=22$ to keep the positivity of the spectrum of A_1. The transformation of the maximal eigenpair $(\lambda_1(A_1),g_1(A_1))$ of A_1 to the one $(\lambda_1,g_1):=(\lambda_1(A_0),g_1(A_0))$ of the original A_0 is as follows.

$$\lambda_1=\lambda_1(A_1)-\bar{\theta},\qquad g_1=\mathrm{Diag}(\mu)^{-1/2}g_1(A_1). \tag{16}$$

Step 2. *Run PI.* As in Step 1, we use the following PI:

$$w_n=A_1v_{n-1},\quad n\geqslant 1,\qquad v_n:=w_n/\|w_n\|,\quad n\geqslant 0.$$

However, the original initial $\mathbb{1}/\|\mathbb{1}\|$ is replaced by $w_0=(1+i)\mathbb{1}/\left(\sqrt{2}\|\mathbb{1}\|\right)$. The reason is that for non-real A_1, since the eigenvalues are all real, the eigenvectors should be non-real and so as a mimic, it is better to choose w_0 to be non-real. However, this is useless in Step 1, since a nonzero constant factor α can be ignored in the equation

$$|A(\alpha v)|=|\lambda|\,|\alpha v|.$$

We now come to the essential different point from the real case. Actually, for non-real A_1, instead of the single equation (1), we have two:

$$\mathrm{Re}(A_1g)=\lambda\,\mathrm{Re}(g),\quad \mathrm{Im}(A_1g)=\lambda\,\mathrm{Im}(g).$$

Thus, it is naturally to split the original vector x (corresponding to g in the eigenequation) into two: x^R and x^I (corresponding to $\mathrm{Re}g$ and $\mathrm{Im}g$, respectively). Similarly we have $\mathscr{N}_R$ and $\mathscr{N}_I$ defined as follows.

$$\begin{cases} \mathscr{N}_R(w)=\{k\colon\ |\mathrm{Re}w(k)|>0\}, \quad \mathscr{N}_I(w)=\{k\colon\ |\mathrm{Im}w(k)|>0\}; \\ p_n=A_1w_n; \\ x_n^R=\left\{\dfrac{\mathrm{Re}p_n}{\mathrm{Re}w_n}(k),\ k\in\mathscr{N}_R(w_n)\right\}, \quad x_n^I=\left\{\dfrac{\mathrm{Im}p_n}{\mathrm{Im}w_n}(k),\ k\in\mathscr{N}_I(w_n)\right\}; \\ \text{weak}\begin{cases} y_n=\left(\bigwedge_{k\in\mathscr{N}_R(w_n)}x_n^R(k)\right)\bigwedge\left(\bigwedge_{k\in\mathscr{N}_I(w_n)}x_n^I(k)\right), \\ z_n=\left(\bigvee_{k\in\mathscr{N}_R(w_n)}x_n^R(k)\right)\bigvee\left(\bigvee_{k\in\mathscr{N}_I(w_n)}x_n^I(k)\right); \end{cases} \\ \text{strong}\begin{cases} y_n=\left(\bigwedge_{k\in\mathscr{N}_R(w_n)}x_n^R(k)\right)\bigvee\left(\bigwedge_{k\in\mathscr{N}_I(w_n)}x_n^I(k)\right), \\ z_n=\left(\bigvee_{k\in\mathscr{N}_R(w_n)}x_n^R(k)\right)\bigwedge\left(\bigvee_{k\in\mathscr{N}_I(w_n)}x_n^I(k)\right); \end{cases} \end{cases} \tag{17}$$

where $\alpha\wedge\beta=\min\{\alpha,\beta\}$ and $\alpha\vee\beta=\max\{\alpha,\beta\}$ for real α and β. The last two parts "weak" and "strong" need some explanation. First, the only difference is exchanging the "$\wedge$" and "$\vee$" in the middle of definition of (y_n,z_n). To understand its essential difference, recall that condition (5) is now split into two:

$$\begin{aligned} &(\mathrm{Re}_x)\colon \qquad \min_{k\in\mathscr{N}_R(x)}\frac{\mathrm{Re}(A_1x)}{\mathrm{Re}x}(k)\leqslant\lambda\leqslant\max_{k\in\mathscr{N}_R(x)}\frac{\mathrm{Re}(A_1x)}{\mathrm{Re}x}(k), \\ &(\mathrm{Im}_x)\colon \qquad \min_{k\in\mathscr{N}_I(x)}\frac{\mathrm{Im}(A_1x)}{\mathrm{Im}x}(k)\leqslant\lambda\leqslant\max_{k\in\mathscr{N}_I(x)}\frac{\mathrm{Im}(A_1x)}{\mathrm{Im}x}(k). \end{aligned}$$

Now, for the "weak" case in (17) we simply adopt a weaker estimate of (y_n,z_n) from (Re_{x_n}) and (Im_{x_n}). And then the "strong" case should be clear. The weaker version of (y_n,z_n) plays the main role for the safety of converging to the required eigenpair, but makes a little slower convergence. While the stronger version makes a faster convergence but it requires that we are at the position close enough to the target eigenpair. Keeping these ideas in mind, one may adopt a mixture of these choices in designing the algorithms.

To fix the idea, throughout this section, the weak version of (y_n,z_n) is adopted at the first use of PI only in the computation of each eigenpair. For the other steps, we adopt the strong version.

It is the position to start the PI. In 6 iterations, the outputs are as follows.

$$\begin{aligned} &\{z_n,y_n\}_{n=1}^6\colon (22.6771,-8.15858),\ (92.2205,21.1287),\ (25.9135,20.2681), \\ &\qquad\qquad (23.4485,18.5274),\ (22.6331,19.0585),\ (22.2652,19.8867); \\ &\{z_n-y_n\}_{n=1}^6\colon 30.8357,\ 71.0918,\ 5.64541,\ 4.92104,\ 3.57468,\ 2.37847; \\ &\{1-y_n/z_n\}_{n=1}^6\colon 1.35977,\ .770889,\ .217856,\ .209866,\ .15794,\ .106825. \end{aligned}$$

Note that here $y_1 < 0$. The outputs show that not only the components of $\mathrm{Re}w_n$ and $\mathrm{Re}w_{n-1}$ have the same sign once $n \geqslant 2$, but also the sequence of relative difference decreases quite quickly. We choose $n = 6$ ($\varepsilon \sim .1$) as the final iteration. Then, we have

$$v_6 = (.363237 + .491209i,\ -.00786884 + .44046i,\ .488441 - .0516616i,\\ .326973 + .290776i)^*.$$

Step 3. *Run IPI$_v$.* Starting at $(z_0, v_0) = (z_6, v_6)$ obtained in the last step, run IPI$_v$. Here we adopt a little different notation. Let w_n solve the equation

$$(z_{n-1}I - A_1)w_n = v_{n-1}, \quad n \geqslant 1.$$

and set $v_n = w_n/\|w_n\|$ again. Next, define $\mathscr{N}_R(w)$, $\mathscr{N}_I(w)$ and $\{x_n^R, x_n^I, y_n, z_n\}$ by (17) with the strong version of (y_n, z_n).

Note that z_n and $1 - y_n/z_n$ are analogs of (6) and (4) in the complex context, respectively. Then, in 2 iterations, we obtain

$$\begin{aligned}
&(z_n, y_n)_{n=1}^2\colon\ (21.8358, 21.8324),\ (21.8344, 21.8344),\\
&(z_n - y_n)_{n=1}^2\colon\ .00346973,\ 5.22045\cdot 10^{-7};\\
&\{1 - y_n/z_n\}_{n=1}^2\colon\ .000158901,\ 2.39093\cdot 10^{-8},\\
&v_2 = (.359825 + .494092i,\ -.0061931 + .44037i,\ .488017 - .054044i,\\
&\qquad .328093 + .289324i)^*.
\end{aligned}$$

Moreover, $1 - y_2/z_2 \sim 10^{-8}$. This is not too small for the use of IPI$_f$ in the next step.

Step 4. *Run IPI$_f$.* In the case we want to improve the above result furthermore, we adopt the IPI$_f$. Now, we take (z_2, v_2) from the last step as our new initial (z_0, v_0). The only change to the last IPI$_v$ is using the fixed $z_n \equiv z_0$. In 3 iterations, if we adopt the same precise digits as the last step, then we get the same outputs of (z_n, y_n) as the last one:

$$\begin{aligned}
&\{(z_n, y_n)\}_{n=1}^3\colon \text{ the same pair } (21.8344,\ 21.8344);\\
&\{y_n - z_n\}_{n=1}^3\colon\ \{10.6581,\ 0,\ 3.55271\}\cdot 10^{-15};\\
&\{1 - y_n/z_n\}_{n=1}^3\colon\ \{5.55112,\ 1.11022,\ 2.22045\}\cdot 10^{-16},\\
&v_3 = (.359825 + .494092i,\ -.00619309 + .44037i,\ .488017 - .054044i,\\
&\qquad .328093 + .289324i)^*.
\end{aligned}$$

In what follows, we rewrite (z_3, v_3) as $(\lambda_1(A_1), g_1(A_1))$ which is regarded as the maximal eigenpair of A_1.

The submaximal eigenpair

From the last part, we have obtained the maximal eigenpair $(\lambda_1(A_1), g_1(A_1))$, at the machine level of precision, as follows.

$$\begin{aligned}\lambda_1(A_1) &= 21.834441785286337,\\ g_1(A_1) &= (.35982503686976175+.49409186313969483i,\\ &\quad -.006193088194633169+.44037016603620777i,\\ &\quad .48801737987976945-.054043998846425696i,\\ &\quad .3280927162424674+.28932402046371486i)^*.\end{aligned}$$

Step 1. *Run modified PI.* As an analog (11), the projection of the vector w on the space Span $(g_1(A_1))^\perp$ is as follows.

$$w-\frac{g_1(A_1)^H w}{(g_1(A_1))^H g_1(A_1)}g_1(A_1) \quad [g^H := \bar{g}^*].$$

The modified PI means the use of the usual PI with the modification by the projection above at each step. That is

$$\begin{aligned}w_0 &= \frac{(1+i)\mathbb{1}}{\sqrt{2}\|\mathbb{1}\|},\\ u_n &= w_n-\frac{g_1(A_1)^H w_n}{(g_1(A_1))^H g_1(A_1)}g_1(A_1), \quad n\geqslant 0,\\ w_n &= A_1\frac{u_{n-1}}{\sqrt{u_{n-1}^H u_{n-1}}}, \quad n\geqslant 1.\end{aligned}$$

Next, similar to (17), replacing w and w_n by u and u_n respectively, we can define $\mathscr{N}_R$, $\mathscr{N}_I$, p_n, x_n^R, x_n^I, and the weak version of (y_n,z_n). Starting at w_0 and running the modified PI, in 5 iterations, we obtain

$$\begin{aligned}&\{(z_n,y_n)\}_{n=1}^5\colon (13.3067,1.07981),\ (12.7854,-32.9231),\ (18.4212,10.2147),\\ &\qquad (13.9055,11.5768),\ (12.9665,12.1958);\\ &\{z_n-y_n\}_{n=1}^5\colon 12.2269,\ 45.7085,\ 8.20655,\ 2.32871,\ .770683;\\ &\{1-y_n/z_n\}_{n=1}^5\colon .918852,\ 3.57505,\ .445495,\ .167467,\ .0594367.\end{aligned}$$

Note that here $y_2<0$. We stop at $n=5$ since $1-y_5/z_5$ is small enough, even though it is bigger than 10^{-2}. Then, we have

$$\begin{aligned}z_5 &= 12.9665,\\ v_5 &= (.0677311-.786181i,\ -.190409+.21529i,\ .356539+.247925i,\\ &\quad .0428153+.322965i)^*.\end{aligned}$$

Step 2. *Run IPI$_v$.* Taking (z_5,v_5) from the last step as new (z_0,v_0), run IPI$_v$. Let w_n solve the equation

$$(z_{n-1}I-A_1)w_n=v_{n-1},\quad n\geqslant 1,$$

and define first $v_n=w_n/\|w_n\|$, and then $\mathscr{N}_R(w)$, $\mathscr{N}_I(w)$ and $\{x_n^R,x_n^I,y_n,z_n\}$ by (17) with the strong version of (y_n,z_n). Now, in 3 iterations, we obtain

$$\begin{aligned}
&\{(z_n,y_n)\}_{n=1}^3\colon (12.5546,12.5507),\ (12.5542,12.5542),\ (12.5542,12.5542);\\
&\{z_n-y_n\}_{n=1}^3\colon .00385406,\ 1.50454\cdot 10^{-7},\ 3.55271\cdot 10^{-15};\\
&\{1-y_n/z_n\}_{n=1}^3\colon .000306985,\ 1.19843\cdot 10^{-8},\ 3.33067\cdot 10^{-16},\\
&v_2=(.604525-.508517i,\ -.301145+.0168374i,\ .0761289+.417867i,\\
&\qquad -.196423+.2569i)^*.\\
&v_3=(.604525-.508517i,\ -.301145+.0168374i,\ .0761289+.417867i,\\
&\qquad -.196423+.2569i)^*.
\end{aligned}$$

In the case we do not want to go further, we can stop here at $n=3$ since $1-y_3/z_3\sim 10^{-16}$ is sufficiently small. It is actually too smaller to go to the next step, otherwise it would cost some computational error.

Step 3. *Run IPI$_f$.* To have a test, setting (z_0,v_0) to be (z_2,v_2) obtained in the last step, run IPI$_f$ also in 3 iterations, we obtain the same output $z_n=y_n=12.5542$ for $n=1,2,3$, and

$$\begin{aligned}
&\{z_n-y_n\}_{n=1}^3\colon \{3.55271,\ 3.55271,\ 8.88178\}\cdot 10^{-15};\\
&\{1-y_n/z_n\}_{n=1}^3\colon \{3.33067,\ 3.33067,\ 6.66134\}\cdot 10^{-16}.
\end{aligned}$$

Moreover

$$\begin{aligned}
v_3=(&.6045251632662887-.5085174051419706i,\\
&-.3011448284487476+.016837350902488956i,\\
&.07612884589652998+.4178669662768421i,\\
&-.19642273529356236+.2568995483366027i)^*.
\end{aligned}$$

The present v_3 has a much higher precise level than v_2 obtained in Step 2. We now regard (z_3,v_3) as the submaximal eigenpair $(\lambda_2(A_1),g_2(A_1))$ of A_1.

Similarly, one can compute the other eigenpairs of A_1 but we are not going to the details here.

Finally, we return to the original eigenpairs of A_0 by (16):

$$\begin{aligned}
&\lambda_1=\lambda_1(A_1)-22=-.165558,\\
&g_1=\mathrm{Diag}(\mu)^{-1/2}g_1(A_1)=(.359825+.494092i,\ -.00848024+.603002i,\\
&\qquad\qquad .963757-.106728i,\ .800304+.705737i)^*,\\
&\lambda_2=\lambda_2(A_1)-22=-9.44576,\\
&g_2=\mathrm{Diag}(\mu)^{-1/2}g_2(A_1)=(.604525-.508517i,\ -.41236+.0230555i,\\
&\qquad\qquad .150342+.825221i,\ -.479127+.626645i)^*.
\end{aligned}$$

It is nice chance to learn some thing from the above computation.

1) In the earlier papers [3] and [7], the sequence $\{z_n\}$ should control the maximal eigenvalue from above, due to the Perron-Frobenius theorem and the C-W formula mentioned in Section 1. However, this may not be true in the present general setup, as can be seen from Step 1 of computing the maximal eigenpair,

$$z_1 < |\lambda^*(\hat{A}_0)| = 21.3806 < z_2 < z_3 > z_4 > z_5 > |\lambda^*(\hat{A}_0)|,$$

the sequence $\{z_n\}$ arrives its maximum at z_3. In Step 2, we have similarly,

$$\lambda_1(A_1) = 21.8344 < z_1 < z_2 > z_3 > \cdots > z_6 > \lambda_1(A_1).$$

In this case, it follows that the sequence $\{z_n\}$ arrives its maximum at z_2, and then it goes down. In both cases, the sequence $\{1-y_n/z_n\}$ is decreasing in n quickly.

Step 1 in computing the submaximal eigenpair is much more interesting. It illustrates the unstable property of $\{z_n\}$ at the beginning. Here we adopt the modified PI. We have

$$z_1 > z_2 > \lambda_2(A_1) = 12.5542 < z_3 > z_4 > z_5 > \lambda_2(A_1).$$

Correspondingly, for $\xi_n := 1-y_n/z_n$, we have

$$\{\xi_n\}_{n=1}^5\colon\ .918852,\ 3.57505,\ .445495,\ .167467,\ .0594367.$$

A big jump happens at z_3 since as mentioned earlier, $y_2 < 0$ and so the check sign (CS) is necessary. At $n = 5$, even though $\xi_5 \sim .059 > .01$, but $z_5 = 13.0168 > \lambda_2(A_1)$, and so the use of IPI$_v$ in the subsequent step is safe. Roughly speaking, one can stop PI at the mth iteration, if starting from z_m, the sequence $\{z_n\}_{n\geqslant m}$ converges decreasingly. It is the case if the matrix has nonnegative off-diagonals, as studied in [3, 7], or the examples given in Section 4. In view of this point, one may reduce the number of iterations in using PI at the beginning of the computation for the maximal/submaximal eigenpair. More precisely, the PI (*Step* 2) for computing the maximal eigenpair needs only $6-2$ iterations and for submaximal one, it requires only $5-1$ iterations. For subsequent IPI$_v$ or IPI$_f$, the number of iterations remains the same as the original in the both cases.

2) All the computations above show that the sequence $\{1-y_n/z_n\}_n$, may be except a few of terms at the beginning, is monotone decreasing and converges, much stable than the other sequences, $\{z_n\}$ or $\{|1-z_n/z_{n-1}|\}$, in the present general setup. Among the computations above, the exceptional part of the sequence $\{\xi_n = 1-y_n/z_n\}$ appears mainly in the last case just discussed above. For which, the first 2 terms are unstable, especially the second one is bigger than 1 since $y_2 < 0$ as mentioned before. The stability starts at the third term. It follows that the use of the sequence $\{1-y_n/z_n\}$ is more practical and is actually adopted in the preliminary version of the algorithm given in Fig. 1. For this reason, it seems more precise to rename the "CS-LBE" technique by adding the relative difference (RD): CS-RD-LBE technique in the general situation.

It is hoped that the algorithm given here could be used in the quantum mechanics computation (cf. [6]).

For the remainder of the paper, we return to real matrices for which Hermitizable becomes symmetrizable. By (15), we can reduce a symmetrizable matrix to a symmetric one. Then, by using (16), we can assume that the given symmetric matrix has a nonnegative spectrum.

3 A version of the global algorithm for large scale matrices

As remarked at the end of the last section, we need only to study the symmetric matrix having nonnegative eigenvalues. In this section, we describe the extended global (or global for short) algorithm for computing the top eigenpairs of a large sparse matrix. This algorithm computes the eigenpairs sequentially, starting from the top eigenpair and then uses the previously computed $(i-1)$ eigenpairs to compute the next ith eigenpair. The flowchart of the algorithm for computing the ith eigenpair is shown in Fig. 2.

We first summarize the key points of the proposed algorithm as follows.

- The inputs to the algorithm are the first $i-1$ eigenpairs $\{(\lambda_j,v_j),\ j=1,\cdots,i-1\}$ that have already been computed using the same algorithm. Here, λ_j denotes the jth largest eigenvalue and v_j denotes the jth eigenvector.
- At the initial iteration $n=0$, we initialize with y_0 given by (10).
- Starting from the initial vector y_0, run a procedure called "Check sign with locally bilateral estimates (CS-LBE)" to determine initial shift z_0 and the corresponding eigenvector estimate x_0. This procedure involves running multiple power iterations with projection and check sign, and estimating z_0 based on the locally bilateral estimates (an analog of (5)). Details of the CS-LBE procedure will be described later.
- Given x_n and z_n, determined by the CS-LBE procedure, we then perform one iteration of the IPI$_v$: $(z_nI-A)y_n=x_n$ to solve for the updated eigenvector estimate y_n.
- Given y_n, we will run the CS-LBE procedure to determine the next shift z_{n+1} and the corresponding eigenvector estimate x_{n+1}.
- Given x_{n+1}, we will check whether the accuracy of x_{n+1} has improved compared to that of earlier iterations. Detailed criterion used to evaluate the accuracy of the eigenvector (which corresponds to the amplitude of LBE given in Section 1) will be described later.
- If the accuracy of x_{n+1} has not improved compared to earlier iterations, then the algorithm has converged. It then outputs the ith eigenpair $\lambda_i=z_{n+1}$, $v_i=x_{n+1}$ and proceeds with the computation of the $(i+1)$th eigenpair. On the other hand, if the accuracy of x_{n+1} has improved compared to earlier iterations, then the algorithm proceeds with the next iteration of IPI$_v$.

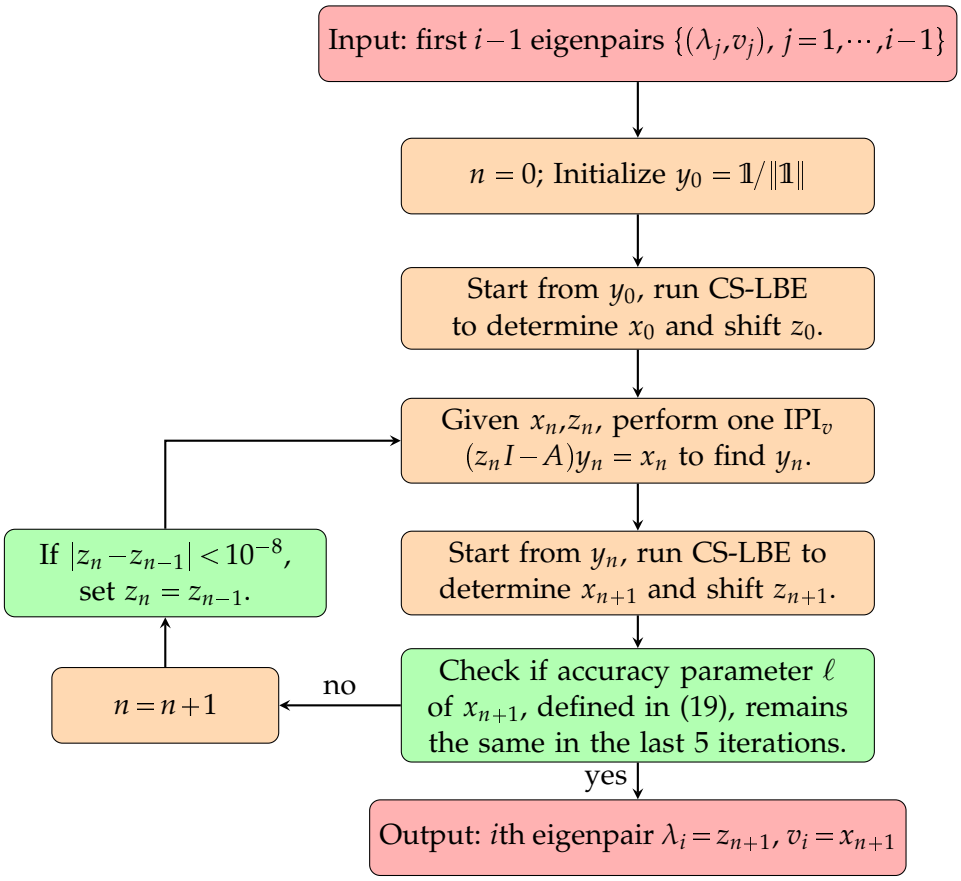


Figure 2: Flowchart of the main algorithm for computing the ith eigenpair. Assume that the previous $i-1$ eigenpairs have been computed.

- Note that for the nth iteration of the IPI$_v$, if the condition $|z_n - z_{n-1}| < 10^{-8}$ is met, then we stop updating the shift and set $z_n = z_{n-1}$ instead. That is, we turn to IPI$_f$.

Next, we provide more details of the global algorithm. We will first define the CS-LBE procedure. This procedure requires the following two basic operations.

Projection operator. This is defined by (11). It ensures that after projection, the vector Proj(x,k) is orthogonal to the linear space Span $(\{v_j, j=1,\cdots,k\})$.

Shift evaluation. Given a current estimate of the eigenvector x, we aim to determine a proper shift based on the locally bilateral estimates. For large sparse matrices, the components of x may decay to zero very quickly. Thus, estimation of the shift using all components of x can be unreliable, and sensitive to the estimation errors of those components of x with very small amplitudes. In our algorithm, we propose to calculate the shift based on only the principal components of x such that $|x(i)| \geqslant t(x)$, where $t(x)$ is a threshold value to be determined. Estimating the shift based on only principal components with

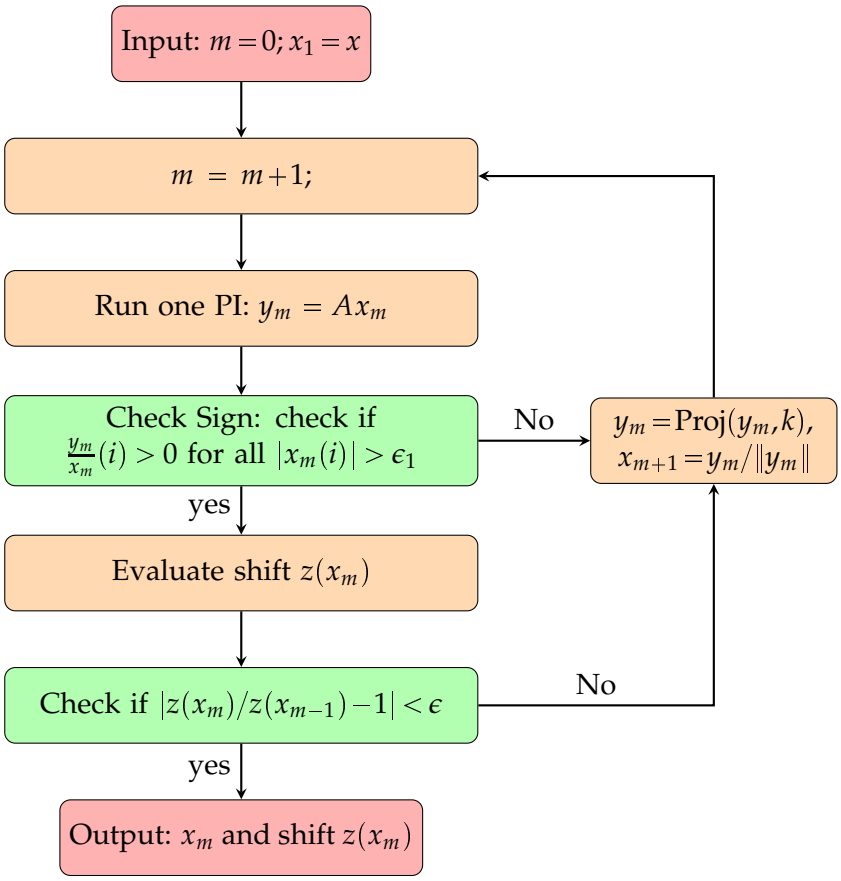


Figure 3: Flowchart of the compute-shift with locally bilateral estimates (CS-LBE) procedure.

larger amplitude improves the estimate of the shift. Let x be a unit vector in the L_2-space of dimension N. Given x, we define the shift evaluation function, denoted by $z(x)$, as follows. Let x_a denote the sorted vector of $|x|$ in the descending order. Let n' be the smallest integer such that $\sum_{i=1}^{n'} x_a(i)^2 \geqslant \epsilon_0$. Typically, we set $\epsilon_0 = 0.9$. This means that the first n' components of vector x_a captures 90% of the energy of vector x. Let $t(x) = |x_a(n')|$. Given x and $y = Ax$, we define the shift evaluation function $z(x)$ by considering only the major components of x such that $|x(i)| \geqslant t(x)$:

$$z(x) = \max_{\{i:|x(i)|\geqslant t(x)\}} \frac{y}{x}(i) \quad \left[\frac{y}{x}(i) := \frac{y(i)}{x(i)}\right]. \tag{18}$$

This is a modification of (6) for the large scale matrix. Note that in (18), we adaptively determine the principal components of the estimated eigenvector x over iterations. This is important to obtain good estimates of the shift $z(x)$.

In Fig. 3, we show the flow-diagram of the CS-LBE procedure in using the *modified PI* (cf. Step 1 of computing the submaximal eigenpair given in Section 2). The input to this

procedure is an initial estimate of the eigenvector x. The subscript m is the index of the PI. At the mth PI, we calculate $y_m = Ax_m$. This is followed by a check sign step in which we check whether the condition that

$$\frac{y_m}{x_m}(i) > 0 \quad \text{is satisfied for all} \quad |x_m(i)| > \epsilon_1 \qquad \text{(analog of (3))}.$$

If check sign fails, then we conduct a projection step on y_m to make sure that the resulting vector is orthogonal to the linear space generated by the first $k-1$ eigenvectors. Then we set $x_{m+1} = y_m/\|y_m\|$ and then proceeds to the next PI. If the check sign is successful, then we compute the shift $z(x_m)$ in the next step. The shift evaluation function z is defined as in (18). We will compare the newly computed shift $z(x_m)$ with the previous shift $z(x_{m-1})$ to see whether the shift values have converged. If so, we will finish the procedure and output the updated estimate of the eigenvector x_m and the shift $z(x_m)$. Otherwise, the algorithm will proceed with the next PI.

Check eigenvector accuracy. Most works in the literature use L_2 norm of the error vector between the true eigenvector and the estimated eigenvector to evaluate the accuracy of the eigenvector estimation. However, since L_2 norm is obtained by summing over all components of the error vector, it can not accurately describe the accuracy of the individual components. In this work, we adopt a different metric by examining the accuracy of component-wise ratios of $y = Ax$ and x. By the definition of the eigenvector, for each component $x(k) \neq 0$, then the ratio $y(k)x(k)^{-1}$ should equal the eigenvalue λ. This is a challenging task for the setting of large matrices due to the high matrix dimension and the rapid decay of the eigenvectors. Typically, when the amplitude of a component $x(k)$ is large, the estimation tends to be more accurate, and thus the ratio $y(k)x(k)^{-1}$ will be closer to the eigenvalue λ. For small $x(k)$, the ratio $y(k)x(k)^{-1}$ tends to deviate away from λ due to estimation inaccuracy. Hence, it is meaningful to consider the amplitude range of $x(k)$ over which all $y(k)x(k)^{-1}$ are close to λ.

We now arrive at the second localized estimation technique: *Accuracy of the principal components of the approximating eigenvector.*

Consider an estimated eigenvector x of dimension N. Let I denote a permutation of $\{1,2,\cdots,N\}$ obtained by sorting the components of $|x|$ in the descending order. Given I, we define $\tilde{x}$ as $\tilde{x}(i) = x(I(i))$, $i = 1,\cdots,N$. Given the same I, we let $y = Ax$ and define $\tilde{y}$ as $\tilde{y}(i) = y(I(i))$, $i = 1,\cdots,N$.

- Let $m' = \max_i \left\{ i\colon\ |\tilde{x}(i)| > 0 \right\}$.
- The accuracy parameter ℓ of the estimated eigenvector specifies the number of reliable components of $\tilde{x}$. It is defined as

$$\ell = \max_{1\leqslant i\leqslant m'} \left\{ i\colon\ \max_{1\leqslant j\leqslant i} \frac{\tilde{y}}{\tilde{x}}(j) - \min_{1\leqslant j\leqslant i} \frac{\tilde{y}}{\tilde{x}}(j) < 10^{-6} \right\}. \tag{19}$$

- Based on the definition of ℓ in (19), we see that the estimated eigenvector x achieves a high accuracy for the largest ℓ components (in absolute value). In other words, the components of x have high accuracy for all components $x(i)$ such that $|x(i)| \geqslant |\tilde{x}(\ell)|$.

Note that in the proposed algorithm shown in Fig. 2, we calculate the accuracy parameter ℓ for the estimated eigenvector x_{n+1} according to (19). As the algorithm proceeds, ℓ will increase over iterations. We terminate the algorithm if ℓ no longer increases over five consecutive iterations.

4 Application to large scale sparse matrices

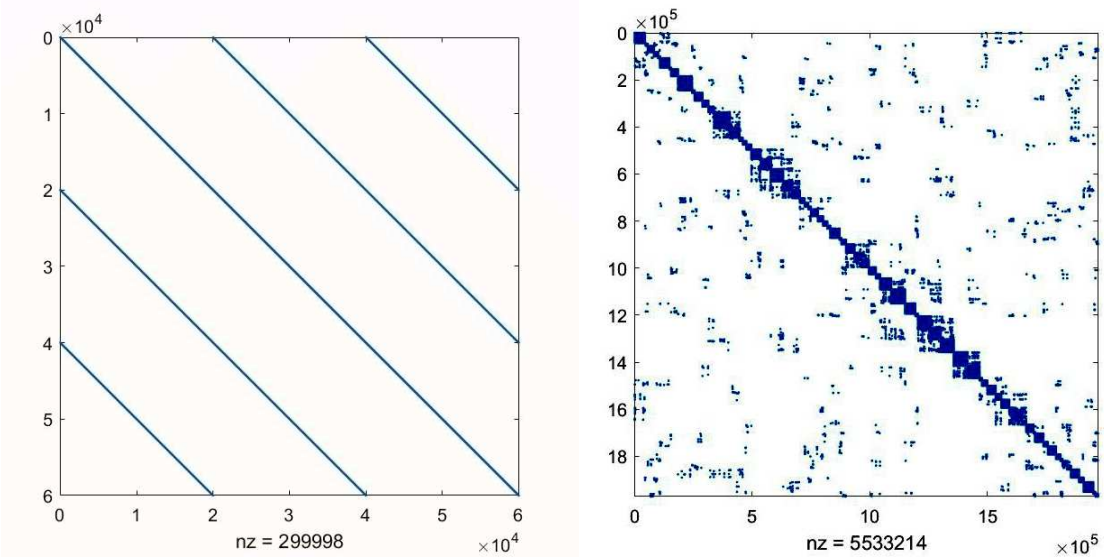
In this section, we provide two examples of using the global algorithm to compute the top 6 eigenpairs. The two large matrices come from the SuiteSparse Matrix Collection, publicly available at `https://sparse.tamu.edu`. We will compare the proposed algorithm with two other methods. One is the Matlab Eigs function, which computes the top six eigenpairs of large, sparse matrices. The other is the modified power iteration method, where we perform the standard power iteration together with the projection step to compute the top six eigenpairs. All the experiments presented in this section are executed on an AMD Ryzen 5 2600 Six-Core Processor with single core CPU speed 3.85 GHz, Memory 32 GB. Matlab version is R2015b Windows 10. Related work on computing the top eigenpair for large sparse matrices include [8, 11, 12]. In particular, [11, 12] consider the use of inverse iterations using fixed shifts. This work differs from [11, 12] in the use of the proposed (CS-LBE) procedure to adaptively compute the shifts. Furthermore, the estimated eigenvector accuracy considered in [8, 11, 12] (for the largest eigenpair only) is similar to that of the Matlab Eigs function, which only guarantees the accuracy of a small number of large principal components. In comparison, the proposed global algorithm achieves a high accuracy for even eigenvector components with an exceedingly small magnitude.

dixmaanl dataset

This matrix has a dimension of $N = 60000$. *The number of nonzero elements* (abbrev. nz) is 299998, This matrix is nonnegative, symmetric, and the range of the elements is between 0 and 154.8089. The sparsity pattern of this matrix is shown in Fig. 4(a).

In Table 1, we show the estimated top 6 eigenvalues obtained by each method. We see that all three methods provide similar eigenvalue estimates that agree with each other up to 10 decimal points.

In Table 2, we provide detailed comparisons of the three methods in terms of the accuracy of the eigenvector, the complexity, and the running time. Each row corresponds to results associated with the ith eigenpair. For instance, the row corresponds to λ_1 reads as follows. The global algorithm estimates the largest (in magnitude) $\ell = 56515$ components of the eigenvector v_1 accurately (see (19)). This represents accurate estimation of all components of v_1 with a magnitude that is greater or equal to $|v_1(\ell)| = 8.1 \cdot 10^{-316}$. The



(a) (b)

Figure 4: Sparsity of the two datasets. (a) dixmaanl (b) roadNet-CA.

Table 1: dixmaanl dataset. Computed top 6 eigenvalues using the Global algorithm, eigs, and modified PI.

	global	eigs	PI
λ_1	317.0152899359881	317.0152899359666	317.0152899359881
λ_2	317.0058090659085	317.0058090659162	317.0058090659074
λ_3	316.9980633932568	316.9980633932683	316.9980633932562
λ_4	316.9912300516546	316.9912300516576	316.9912300516548
λ_5	316.9849936226963	316.9849936226929	316.9849936226971
λ_6	316.9791911040992	316.9791911040974	316.9791911040990

triple (288,40,5) means that in order to achieve this accuracy, the global algorithm took a total of 288 power iterations, including 40 iterations for inverse power iterations (35 of IPI_f and 5 of IPI_v). The global algorithm took 5.1 seconds to compute the first eigenpair while achieving this high level of accuracy. In comparison, the Matlab eigs function, which computes all 6 top eigenpairs all at once, has a much inferior eigenvector accuracy. Only the largest $\ell = 3311$ components of estimated v_1 achieve the desired accuracy of (19) and these components are at least $|v_1(\ell)| = 3.7 \cdot 10^{-8}$ in magnitude. The total computation time of the eigs function for all 6 eigenpairs is 30 seconds. This is comparable with the total computation time of the global algorithm, however, with a significantly lower level of eigenvector accuracy. For the modified PI, we see that it can achieve an accuracy that is comparable to that of the global algorithm. However, the computation time is significantly longer. Due to its slow convergence, it takes 894 seconds and a total

Table 2: dixmaanl dataset. Results and complexity using the Global algorithm, eigs, and modified PI.

	Global				eigs		
	ℓ	$\lvert\tilde{x}(\ell)\rvert$	# iteration	time	ℓ	$\lvert\tilde{x}(\ell)\rvert$	time
1st	56515	8.1e-316	(288, 40, 5)	5.1	3311	3.7e-08	30
2nd	57294	8.7e-316	(319, 45, 6)	5.8	3883	3.2e-08	
3rd	57936	9.2e-316	(244, 40, 5)	4.6	4306	4.1e-08	
4th	58515	8.7e-316	(274, 45, 7)	5.2	4599	8.6e-08	
5th	59020	1.2e-315	(276, 45, 5)	5.4	5138	2.9e-08	
6th	59536	9.1e-316	(312, 45, 6)	6.3	5401	5.2e-08	

	Modified PI			
	ℓ	$\lvert\tilde{x}(\ell)\rvert$	time	# PI
1st	56460	1.9e-315	894	1.5e+06
2nd	57246	1.8e-315	1173	1.5e+06
3rd	57896	1.7e-315	1442	1.5e+06
4th	58472	1.7e-315	1718	1.5e+06
5th	58988	2.0e-315	1998	1.5e+06
6th	59480	2.0e-315	2292	1.5e+06

of $1.5\cdot10^6$ PIs in order to attain a similar accuracy as that of the global algorithm. Similar observations are made for the estimations of the other 5 eigenpairs. The proposed global algorithm achieves the best accuracy with the shortest computation time. We note that the main difference between the Global algorithm and the modified PI is that the former uses inverse power iteration with adaptive shifts, whereas the latter uses standard power iterations. Our results shown that the proposed CS-LBE procedure for computing the variable shifts is crucial in accelerating the convergence speed of the algorithm.

In Table 3, for each eigenpair, we show the value of the shifts used in the Global algorithm. The shifts are generated using the CS-LBE procedure. For instance, the column labeled as "1st" lists 5 values of the shifts z_i, $i=1,\cdots,5$, used in the estimation of the 1st eigenpair. We see that z_i approaches the true λ value (shown in the last row) quickly. For the first eigenpair, only 5 different shifts are needed. In comparison, for the 4th and the 6th eigenpair, more shifts 7, and 6, respectively, are needed.

roadNet-CA dataset

For this dataset, the dimension of the matrix is $N=1971281$. This matrix corresponds to a graph of the road network of California. Each element is either 0 or 1. The sparsity pattern of this matrix is shown in Fig. 4(b). The number of nonzero elements in the matrix is $\mathrm{nz}=5533214$, see Fig. 4(b).

In Table 4, we show detailed comparisons of the three methods in terms of the accuracy of the eigenvector, the complexity, and the running time. We see that the Global

Table 3: dixmaanl dataset. Shifts used by the Global algorithm.

	1st	2nd	3rd
z_1	317.2018759831095	317.0149029206981	317.0054220600994
z_2	317.0220587013249	317.0412999365110	317.0056140237707
z_3	317.0165531440067	317.0183499796456	316.9974443732343
z_4	317.0152788610227	317.0044840756761	316.9980602821128
z_5	317.0152899359775	317.0057627487807	316.9980633932562
z_6		317.0058090643057	
λ	317.0152899359881	317.0058090659085	316.9980633932568

	4th	5th	6th
z_1	316.9976763951934	316.9908430604246	316.9846066377027
z_2	317.0174070334262	316.9924907259373	317.0002253316557
z_3	316.9879937432648	316.9843539028472	316.9767242599030
z_4	316.9896903234464	316.9849885385036	316.9784124921462
z_5	316.9910317369073	316.9849936226933	316.9791679223474
z_6	316.9912298325970		316.9791911036812
z_7	316.9912300516546		
λ	316.9912300516546	316.9849936226963	316.9791911040992

Table 4: roadNet-CA dataset. Results and complexity using the Global algorithm, eigs, and modified PI.

	Global				eigs		
	ℓ	$\lvert\tilde{x}(\ell)\rvert$	# iterations	time	ℓ	$\lvert\tilde{x}(\ell)\rvert$	time
1st	1933344	2.8e-317	(244, 35, 3)	309	1543	8.7e-10	38
2nd	1926704	3.0e-317	(245, 35, 3)	322	1413	1.1e-09	
3rd	1957027	2.8e-295	(226, 30, 3)	276	2004	1.0e-09	
4th	1948213	2.2e-317	(243, 30, 3)	285	2190	5.3e-10	
5th	1956156	2.3e-317	(242, 30, 3)	293	2409	2.9e-10	
6th	1923583	2.7e-317	(282, 30, 2)	296	1648	7.8e-10	

	Modified PI			
	ℓ	$\lvert\tilde{x}(\ell)\rvert$	# PI	time
1st	1933452	2.0e-317	1.0e+04	381
2nd	1926900	2.0e-317	2.5e+04	1432
3rd	1957027	2.8e-295	2.7e+04	2062
4th	49653	1.6e-33	5e+04	4700
5th	62901	1.8e-33	7e+03	776
6th	1923767	1.9e-317	5.0e+04	6671

Table 5: roadNet-CA dataset. Computed top 6 eigenvalues using the Global algorithm, eigs, and modified PI.

	global	eigs	PI
λ_1	4.638361867351406	4.638361867351387	4.638361867351406
λ_2	4.527027931848926	4.527027931848909	4.527027931848924
λ_3	4.451588326941737	4.451588326941750	4.451588326941737
λ_4	4.390275021532836	4.390275021532792	4.390275021532837
λ_5	4.383736144475813	4.383736144475774	4.383736144475815
λ_6	4.325729176980614	4.325729176980572	4.325729176980615

Table 6: roadNet-CA dataset. Shifts used by the Global algorithm.

	1st	2nd	3rd
z_1	4.651095152690492	4.541091827266276	4.457490006778257
z_2	4.638369301398686	4.527034278350056	4.451618990253862
z_3	4.638361867350882	4.527027931841913	4.451588326915770
λ	4.638361867351406	4.527027931848926	4.451588326941737

	4th	5th	6th
z_1	4.390768119815626	4.384210430746412	4.325729209088518
z_2	4.390275047542154	4.383736186751131	4.325729176980588
z_3	4.390275021532815	4.383736144475802	
λ	4.390275021532836	4.383736144475813	4.325729176980614

algorithm reaches very good accuracy in terms of ℓ and $|\tilde{x}(\ell)|$ for all 6 eigenpairs. Due to the increased matrix dimension, the computation time increases compared to that of the dixmaanl dataset. The eigs function can compute the top 6 eigenpairs quickly, using only a total of 38 seconds, but with a much inferior accuracy in ℓ and $|\tilde{x}(\ell)|$. The modified PI algorithm can achieve a very good accuracy for the top 3 eigenpairs, despite a longer computation time for using a high number of PI. The accuracy of the remaining 3 eigenpairs is much worse for the given number of PI.

In Table 5, we show the estimated top 6 eigenvalues using the three algorithms. They all find similar eigenvalues.

In Table 6, we show the shifts produced by the CS-LBE procedure. We observe that, despite the higher dimension of this dataset, the shift values converge to the eigenvalues quickly. Up to 3 shift values are sufficient to approach the eigenvalues.

Acknowledgments

The first author thanks Professors Jia, Z.G. and Pang, H.K. and their teams for the fruitful discussions, verifying and suggestions which improve the quality of the paper. The

teams are now continuously working on applications of the proposed method, such as computing the sparse principal components of medical images and others. Special thanks to Professor Xie, Y.C. for his support and great help during the period the author worked at the university. Research supported in part by National Natural Science Foundation of China (Grant Nos. 12090011, 11771046), National Key R & D Program of China (No. 2020YFA0712900), the project from the Ministry of Education in China, and the Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions.

References

[1] Chen, M.F. (2005). *Eigenvalues, Inequalities, and Ergodic Theory*. London: Springer.
[2] Chen, M.F. (2016). *Efficient initials for computing maximal eigenpair*. Front. Math. China 11(6): 1379–1418.
[3] Chen, M.F. (2017a). *Global algorithms for maximal eigenpair*. Front. Math. China 12(5): 1023–1043.
[4] Chen, M.F. (2017b). *Trilogy on computing maximal eigenpair*. In: Yue, W., Li, Q. L., Jin, S., Ma, Z., eds. "Queueing Theory and Network Applications". QTNA 2017. Lecture Notes in Comput. Sci., Vol. 10591. Cham: Springer, 312–329.
[5] Chen, M.F. (2018). *Hermitizable, isospectral complex matrices or differential operators*. Front. Math. China 13(6): 1267–1311.
[6] Chen, M.F., Jia, Z.G. and Pang, H.K. (2021). *Computing top eigenpairs of Hermitizable matrix*. Front. Math. China 16(2): 345–379.
[7] Chen, M.F., Li, Y.S. (2019). *Improved global algorithms for maximal eigenpair*. Front. Math. China 14(6): 1077–1116.
[8] Lei, Q., Zhong, K., Dhillon, I.S. (2016). *Coordinate-wise power method.* Proc. Advances in Neural Information Processing Systems, 2056–2064.
[9] Press, W.H. et al. (2007). *Numerical Recipes—The Art of Scientific Computing*, 3rd ed. Cambridge Univ. Press.
[10] Varga, R.S. (2004). *Geršgorin and His Circles*. Springer.
[11] Wang, J.L., Wang, W.R., Garber, D., Srebro, N. (2018). *Efficient coordinate-wise leading eigenvector computation*. Proc. Algorithmic Learning Theory, PMLR, 806–820.
[12] Xu, Z.Q. (2018). *Gradient descent meets shift-and-invert preconditioning for eigenvector computation*. Proc. Advances in Neural Information Processing Systems, 31: 2825–2834.