JSIAM Letters Vol.1 (2009) pp.40–43 ©2009 Japan Society for Industrial and Applied Mathematics JSIAM Letters

Eigendecomposition algorithms solving sequentially quadratic systems by Newton method

Koichi Kondo[1], Shinji Yasukouchi[1] and Masashi Iwasaki[2]

Faculty of Science and Engineering, Doshisha University, 1-3 Tatara Miyakodani, Kyotanabe City, 610-0394, Japan[1]
Faculty of Life and Environmental Sciences, Kyoto Prefectural University, 1-5 Nagaragi-cho Shimogamo, Sakyo-ku, Kyoto 606-8522, Japan[2]

E-mail *kokondo@mail.doshisha.ac.jp*

Received March 17, 2009, Accepted June 20, 2009

Abstract

In this paper, we design new algorithms for eigendecomposition. With the help of the Newton iterative method, we solve a nonlinear quadratic system whose solution is equal to an eigenvector on a hyperplane. By choosing normal vector of the hyperplane in the orthogonal complement of the space spanned by already obtained eigenvectors, all eigenpairs are sequentially obtained by solving the quadratic systems.

Keywords eigendecomposition, the Newton method, quadratic method

Research Activity Group Algorithms for Matrix / Eigenvalue Problems and their Applications

1. Introduction

The quadratic method is known as one of the methods for all eigenpairs [1]. In this method, the eigenvalue problem is replaced with the nonlinear quadratic systems. For an eigenpair, the solution of the quadratic system is computed by using the Newton iterative method. For all eigenpairs, the continuation method is proposed in [1]. The continuation method requires not only a quadratic system to be solved for original eigenvalue problem but also many perturbative ones. And furthermore it often fails in finding the desired eigenpairs. Even if it succeeds, the obtained eigenpairs are not always computed with high accuracy. In this paper, we design new eigendecomposition algorithms, which are different from the continuation method, through solving the quadratic systems with the help of the Newton method. Our algorithms are not also equivalent to the standard inverse iteration method. In some numerical experiments, we show that all eigenvectors are computable by our algorithms.

2. Quadratic method

In this paper, we consider the eigenvalue problem

$$A\boldsymbol{x} = \lambda\boldsymbol{x}, \quad A \in \mathbb{C}^{n\times n}, \tag{1}$$

where $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \in \mathbb{C}^n$ denote the eigenvalue and the corresponding eigenvector of A, respectively.

Let $\boldsymbol{z}$ be an n-dimensional vector. Let $(\boldsymbol{z}, \boldsymbol{x}) = \boldsymbol{z}^H\boldsymbol{x} = C$ for some nonzero constant C, where $(\cdot,\cdot)$ and the superscript H denote the inner product of two vectors and the complex conjugate of matrix, respectively. The case where $\boldsymbol{z} = \boldsymbol{e}_k$ is discussed in [1] where $\boldsymbol{e}_k$ is a unit vector whose kth entry is the unity. Noting that $\lambda = \lambda(\boldsymbol{x}) = (A^H\boldsymbol{z}, \boldsymbol{x})/C$ for suitable $\boldsymbol{z}$, then the eigenvector $\boldsymbol{x}$ is given by solving the nonlinear quadratic system

$$\boldsymbol{F}(\boldsymbol{x}) := A\boldsymbol{x} - \frac{(\boldsymbol{w}, \boldsymbol{x})}{C}\boldsymbol{x} = \boldsymbol{0}, \quad \boldsymbol{w} = A^H\boldsymbol{z}. \tag{2}$$

With the help of the Newton iterative method, the solution $\boldsymbol{x}$ is computable by the recurrence formula

$$\begin{cases} \boldsymbol{x}^{(\ell+1)} = \dfrac{C\hat{\boldsymbol{x}}^{(\ell+1)}}{(\boldsymbol{z}, \hat{\boldsymbol{x}}^{(\ell+1)})}, \quad \ell = 0, 1, \ldots, \ell_{\max}, \\ \hat{\boldsymbol{x}}^{(\ell+1)} = \boldsymbol{x}^{(\ell)} - J(\boldsymbol{x}^{(\ell)})^{-1}\boldsymbol{F}(\boldsymbol{x}^{(\ell)}), \\ J(\boldsymbol{x}^{(\ell)}) = A - \lambda(\boldsymbol{x}^{(\ell)})I - \dfrac{\boldsymbol{x}^{(\ell)}\boldsymbol{w}^H}{C}, \\ \lambda(\boldsymbol{x}^{(\ell)}) = \dfrac{(\boldsymbol{w}, \boldsymbol{x}^{(\ell)})}{C}, \end{cases} \tag{3}$$

where I is an n-dimensional unit matrix and $\boldsymbol{x}^{(0)}$ is an initial vector. See Section 3 for the setting of $\boldsymbol{x}^{(0)}$. Let ℓ^* be the number in (3) such that

$$\|A\boldsymbol{x}^{(\ell^*)} - \lambda(\boldsymbol{x}^{(\ell^*)})\boldsymbol{x}^{(\ell^*)}\|_\infty < \epsilon_{\mathrm{itr}}\|\boldsymbol{x}^{(\ell^*)}\|_2 \tag{4}$$

for small ϵ_{itr}. Then $\boldsymbol{x}^{(\ell^*)}$ becomes a good approximation of $\boldsymbol{x}$ in (2). By the normalization $\boldsymbol{x}^{(\ell)} \to \boldsymbol{x}^{(\ell)}/\|\boldsymbol{x}^{(\ell)}\|_2$ for each ℓ in (3), the inequality (4) becomes

$$\|A\boldsymbol{x}^{(\ell^*)} - \lambda(\boldsymbol{x}^{(\ell^*)})\boldsymbol{x}^{(\ell^*)}\|_\infty < \epsilon_{\mathrm{itr}}. \tag{5}$$

Note here that $\|\boldsymbol{x}^{(\ell)}\|_2 = 1$ for each ℓ. Moreover, by replacing C with $C^{(\ell)} = (\boldsymbol{z}, \boldsymbol{x}^{(\ell)})$ in (2) and (3), it follows that

$$\begin{cases} \boldsymbol{x}^{(\ell+1)} = \dfrac{\hat{\boldsymbol{x}}^{(\ell+1)}}{\|\hat{\boldsymbol{x}}^{(\ell+1)}\|_2}, \quad \ell = 0, 1, \ldots, \ell_{\max}, \\ \hat{\boldsymbol{x}}^{(\ell+1)} = \boldsymbol{x}^{(\ell)} - J(\boldsymbol{x}^{(\ell)})^{-1}\boldsymbol{F}(\boldsymbol{x}^{(\ell)}), \\ J(\boldsymbol{x}^{(\ell)}) = A - \lambda(\boldsymbol{x}^{(\ell)})I - \dfrac{\boldsymbol{x}^{(\ell)}\boldsymbol{w}^H}{(\boldsymbol{z}, \boldsymbol{x}^{(\ell)})}, \\ \lambda(\boldsymbol{x}^{(\ell)}) = \dfrac{(\boldsymbol{w}, \boldsymbol{x}^{(\ell)})}{(\boldsymbol{z}, \boldsymbol{x}^{(\ell)})}. \end{cases} \tag{6}$$

At each ℓ, the hyperplane $(\boldsymbol{z}, \boldsymbol{x}^{(\ell)}) = C^{(\ell)}$ is translated without changing its normal vector. We call the algorithm for an eigenpair based on (6) the `neig_J` algorithm. By applying the Sherman-Morrison formula

$$(M + \boldsymbol{u}\boldsymbol{v}^H)^{-1} = \left(I - \frac{M^{-1}\boldsymbol{u}\boldsymbol{v}^H}{1 + (\boldsymbol{v}, M^{-1}\boldsymbol{u})}\right) M^{-1} \tag{7}$$

to the inverse $J(\boldsymbol{x}^{(\ell)})^{-1}$ in (6), we have

$$\begin{cases} \hat{\boldsymbol{x}}^{(\ell+1)} = \dfrac{\lambda(\boldsymbol{x}^{(\ell)})}{(\boldsymbol{w}, \tilde{\boldsymbol{x}}^{(\ell)})/(\boldsymbol{z}, \boldsymbol{x}^{(\ell)}) - 1}\tilde{\boldsymbol{x}}^{(\ell)}, \\ \tilde{\boldsymbol{x}}^{(\ell)} = (A - \lambda(\boldsymbol{x}^{(\ell)})I)^{-1}\boldsymbol{x}^{(\ell)}. \end{cases} \tag{8}$$

Hence the following recurrence formula also generates the evolution from ℓ to $\ell + 1$ of $\boldsymbol{x}^{(\ell)}$.

$$\begin{cases} \boldsymbol{x}^{(\ell+1)} = \dfrac{\tilde{\boldsymbol{x}}^{(\ell+1)}}{\|\tilde{\boldsymbol{x}}^{(\ell+1)}\|_2}, \quad \ell = 0, 1, \ldots, \ell_{\max}, \\ \tilde{\boldsymbol{x}}^{(\ell+1)} = (A - \lambda(\boldsymbol{x}^{(\ell)})I)^{-1}\boldsymbol{x}^{(\ell)}, \\ \lambda(\boldsymbol{x}^{(\ell)}) = \dfrac{(\boldsymbol{w}, \boldsymbol{x}^{(\ell)})}{(\boldsymbol{z}, \boldsymbol{x}^{(\ell)})}. \end{cases} \tag{9}$$

In [2, p. 194], (9) is called as a generalized Rayleigh quotient iteration. If $\lambda(\boldsymbol{x}^{(\ell)}) = \lambda$ is given, then the iteration (9) becomes

$$\boldsymbol{x}^{(\ell+1)} = \frac{\tilde{\boldsymbol{x}}^{(\ell+1)}}{\|\tilde{\boldsymbol{x}}^{(\ell+1)}\|_2}, \quad \tilde{\boldsymbol{x}}^{(\ell+1)} = (A - \lambda I)^{-1}\boldsymbol{x}^{(\ell)}. \tag{10}$$

This is well-known as the inverse iteration for computing eigenvector. The iteration (9) may be regarded as one of inverse iterations with updating $\lambda(\boldsymbol{x}^{(\ell)})$ at each ℓ by a generalized Rayleigh quotient $\lambda(\boldsymbol{x}^{(\ell)}) = (\boldsymbol{z}, A\boldsymbol{x}^{(\ell)})/(\boldsymbol{z}, \boldsymbol{x}^{(\ell)})$. We call the algorithm based on (9) the `neig_I` algorithm. Though the computed eigenpair by the `neig_I` algorithm is theoretically the same as that by the `neig_J` algorithm, the `neig_I` algorithm is obviously different from the `neig_J` algorithm with respect to numerical accuracy. See Section 4 for numerical accuracy.

3. Eigendecomposition algorithm

An eigenpair $(\lambda, \boldsymbol{x})$ is computable if suitable initial vector $\boldsymbol{x}^{(0)}$ is given in (6), (9). The other eigenpairs are also computed by changing $\boldsymbol{x}^{(0)}$ in (6), (9). Namely, we can theoretically compute all eigenpairs by using the `neig_`$*$ algorithm. It is, however, not easy to compute all eigenpairs if $\boldsymbol{x}^{(0)}$ is randomly given. It is well-known that the fractal graph is given from the relationship between the initial vector $\boldsymbol{x}^{(0)}$ and the limit $\lim_{\ell\to\infty}\boldsymbol{x}^{(\ell)}$ in the Newton iteration method (cf. [3, pp. 237–242]). Namely, it is not expected to choose $\boldsymbol{x}^{(0)}$ for computing the desired eigenpair in the `neig_`$*$ algorithm.

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ be the already obtained eigenvectors where $k < n$. We here consider the subspace $W_k := \langle \boldsymbol{x}_1, \cdots, \boldsymbol{x}_k \rangle_{\mathbb{C}}$ and its orthogonal complement $W_k^{\perp}$. Since the normal vector $\boldsymbol{z}$ of the hyperplane $(\boldsymbol{z}, \boldsymbol{x}^{(\ell)}) = C^{(\ell)}$ is changeable, we may adopt the vector in $W_k^{\perp}$ as $\boldsymbol{z}$. It is remarkable that $W_k^{\perp}$ does not include $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$. Let us assume that $\boldsymbol{x}^{(\ell)}$ converges as $\ell \to \infty$. Then it is obvious that, for $\ell = 1, 2, \ldots$, $C^{(\ell)} \neq 0$ and $\lim_{\ell\to\infty} C^{(\ell)} \neq 0$. This implies that $\lim_{\ell\to\infty}\boldsymbol{x}^{(\ell)} \notin W_k$. Hence $\boldsymbol{x}^{(\ell)} \to \boldsymbol{x}_{k+1}$ and $\lambda(\boldsymbol{x}^{(\ell)}) \to \lambda_{k+1}$ as $\ell \to \infty$. Namely, the eigenpair $(\lambda_{k+1}, \boldsymbol{x}_{k+1})$ is computable by the `neig_`$*$ algorithm. Similarly, the others are obtained only if $\boldsymbol{x}^{(\ell)}$ converges as $\ell \to \infty$ for each k. Therefore, all eigenpairs are sequentially computed by the following algorithm.

Algorithm 1

```
01  function [X,D]=sneig_*(A)
02  t := 0
03  Q = (q_1 ··· q_n) := I
04  for k = 1, 2, ..., n
05    z := q_k ∈ W_{k-1}^⊥
06    f := 0
07    do
08      x^(0) := random_vec(n)
09      [x_k, λ_k, E_k] := neig_*(A, x^(0), z, ℓ_max)
10      θ := min_{j=1,...,k-1} angle(x_k, x_j)
11      t := t + 1; f := f + 1
12      if f ≥ f_max then stop   % failed
13    while(E_k ≥ ε_good or θ ≤ θ_same)
14    r̃_k := x_k
15    for j = 1, ..., k - 1
16      r̃_k := r̃_k - α_j(h_j, r̃_k)h_j
17    end
18    [h_k, α_k] := householder_vec(r̃_k)
19    Q := Q - α_k(Qh_k)h_k^H
20  end
21  X := (x_1 ··· x_n); D := diag(λ_1, ..., λ_n)
```

Here we call Algorithm 1 the `sneig_`$*$ algorithm. The `sneig_J`, the `sneig_I` algorithms employ the `neig_J`, the `neig_I` algorithms, respectively.

In the 8th line of Algorithm 1, we make choice of the initial complex vector $\boldsymbol{x}^{(0)}$ randomly. In the 9th line, by the `neig_`$*$ algorithm, we compute the kth eigenvalue λ_k, the corresponding eigenvector $\boldsymbol{x}_k$ and the residual norm $E_k := \|A\boldsymbol{x}_k - \lambda_k\boldsymbol{x}_k\|_\infty$. As discussed in the above, the `neig_`$*$ algorithm does not converge for unsuitable $\boldsymbol{x}^{(0)}$. We regard that the `neig_`$*$ algorithm does not converge if $E_k \geq \epsilon_{\text{good}}$ for small ϵ_{good}. And then we perform the `neig_`$*$ algorithm after the change of $\boldsymbol{x}^{(0)}$. The operations from the 7th line to the 13th line are repeated until $E_k < \epsilon_{\text{good}}$. Theoretically, $\boldsymbol{x}_k$ is not equal to one of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}$. This property is not always guaranteed in the double precision arithmetic. In the 10th line, we compute the minimal angle $\theta := \min_{j=1,\ldots,k-1}\text{angle}(\boldsymbol{x}_k, \boldsymbol{x}_j)$ where

$$\text{angle}(\boldsymbol{x}_k, \boldsymbol{x}_j) := \frac{180}{\pi}\cos^{-1}\left(\frac{|(\boldsymbol{x}_k, \boldsymbol{x}_j)|}{\|\boldsymbol{x}_k\|_2\|\boldsymbol{x}_j\|_2}\right). \tag{11}$$

We regard that $\boldsymbol{x}_k$ is equal to one of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}$ if $\theta \leq \theta_{\text{same}}$ for small θ_{same}, and then we perform the `neig_`$*$ algorithm after the change of $\boldsymbol{x}^{(0)}$. Let f be the iteration number of the `neig_`$*$ algorithm for an eigenpair. Then we regard that only a part of eigenpairs is computed by the `sneig_`$*$ algorithm if $f \geq f_{\max}$ for the maximal iteration number $f_{\max}$. In this case, the `sneig_`$*$ algorithm is coercively stopped in the 12th line.

In the 5th line of Algorithm 1, we choose $\boldsymbol{z}$ in the orthogonal complement $W_{k-1}^{\perp}$. In this paper, for the

choice of $\boldsymbol{z}$ we use the QR decomposition based on the Householder transformation. Let $X_{k-1} = Q_{k-1}R_{k-1}$ be the QR decomposition of $X_{k-1} = (\boldsymbol{x}_1 \cdots \boldsymbol{x}_{k-1})$, where $Q_{k-1} = (\boldsymbol{q}_1 \cdots \boldsymbol{q}_n) \in \mathbb{C}^{n\times n}$, $R_{k-1} = (\boldsymbol{r}_1 \cdots \boldsymbol{r}_{k-1}) \in \mathbb{C}^{n\times(k-1)}$ are the unitary, the upper triangle matrices, respectively. Let $W_{k-1} = \langle \boldsymbol{q}_1, \cdots, \boldsymbol{q}_{k-1}\rangle_{\mathbb{C}}$. Then it is obvious that $W_{k-1}^{\perp} = \langle \boldsymbol{q}_k, \cdots, \boldsymbol{q}_n\rangle_{\mathbb{C}}$. This implies that $\boldsymbol{z}$ should be the linear combination of the basis $\boldsymbol{q}_k, \ldots, \boldsymbol{q}_n$. In Algorithm 1, we set $\boldsymbol{z} = \boldsymbol{q}_k$. From the viewpoint of the running time, it is not desirable that we compute the QR decomposition of X_k for each k. It is of significance to note here that the columns from the 1st to the $(k-1)$th of R_k, Q_k are equal to those of R_{k-1}, Q_{k-1}, respectively. Hence, in the kth Householder transformation, we compute only the kth column of R_k. In the lines from the 14th to the 17th, we compute the kth column $\tilde{\boldsymbol{r}}_k = (r_{1,k} \cdots r_{k-1,k}\, \tilde{r}_{k,k} \cdots \tilde{r}_{n,k})^T$ of $Q_{k-1}^H X_k = (\boldsymbol{r}_1 \cdots \boldsymbol{r}_{k-1}\, \tilde{\boldsymbol{r}}_k)$ from $\boldsymbol{x}_k$. In the 18th line, we derive $\boldsymbol{h}_k$ and α_k from $\tilde{\boldsymbol{r}}_k$ for computing the Householder matrix $H_k := I - \alpha_k \boldsymbol{h}_k \boldsymbol{h}_k^H$ as follows:

$$\boldsymbol{h}_k = (0 \ \cdots \ 0 \ \ -\zeta\xi \ \ \tilde{r}_{k+1,k} \ \cdots \ \tilde{r}_{n,k})^T, \tag{12}$$

$$\zeta := \frac{\tilde{r}_{k,k}}{|\tilde{r}_{k,k}|}, \quad \eta := \sqrt{|\tilde{r}_{k,k}|^2 + \cdots + |\tilde{r}_{n,k}|^2}, \tag{13}$$

$$\xi := \frac{|\tilde{r}_{k+1,k}|^2 + \cdots + |\tilde{r}_{n,k}|^2}{|\tilde{r}_{k,k}| + \eta}, \quad \alpha_k = \frac{1}{\xi\eta}, \tag{14}$$

where $H_k : \tilde{\boldsymbol{r}}_k \mapsto \boldsymbol{r}_k = (r_{1,k} \cdots r_{k-1,k} \ \ \zeta\eta \ \ 0 \cdots 0)^T$ and $H_k^H Q_{k-1}^H X_k = R_k = (\boldsymbol{r}_1 \cdots \boldsymbol{r}_{k-1}\, \boldsymbol{r}_k)$. In the 19th line, we compute Q_k as $Q_k = Q_{k-1}H_k = Q_{k-1} - \alpha_k(Q_{k-1}\boldsymbol{h}_k)\boldsymbol{h}_k^H$. It is remarkable that $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k-1}$ are not changed in the 19th line since $\boldsymbol{h}_k$ has 0 from the 1st entry to the $(k-1)$th entry. As a result, the `sneig_`$*$ algorithm requires only the operations for a QR decomposition. The Lanczos method is also shown in [4] as the vector orthogonormalization method by using the Householder transformation without saving the upper-triangle matrix.

4. Numerical experiments

In this section, we show some numerical experiments with respect to the `sneig_`$*$ algorithm and the inverse iteration based on (10). Let us call the inverse iteration based on (10) the `sii` algorithm for simplicity. Numerical experiments have been carried out on our computer with OS: Linux 2.6.26, CPU: Intel Core i7, RAM: 2GB. We also use GNU C Compiler 4.3.2 and LAPACK 3.1.1 [5]. As test matrix, we adopt the Toeplitz matrix

$$A = \begin{pmatrix} 2 & 1 & & & & \\ 0 & 2 & 1 & & & \\ \gamma & 0 & 2 & \ddots & & \\ & \gamma & \ddots & \ddots & 1 & \\ & & \ddots & 0 & 2 & 1 \\ & & & \gamma & 0 & 2 \end{pmatrix}. \tag{15}$$

In [6], the Toeplitz matrix (15) appears in numerical test for the solvers of the linear equations. In the `sneig_`$*$ algorithm, we set $\epsilon_{\mathrm{itr}} = 10^{-13}$, $\ell_{\max} = 50$, $\epsilon_{\mathrm{good}} = 5 \times 10^{-13}$, $\theta_{\mathrm{same}} = 0.3°$, $f_{\max} = 2n$. The inverse

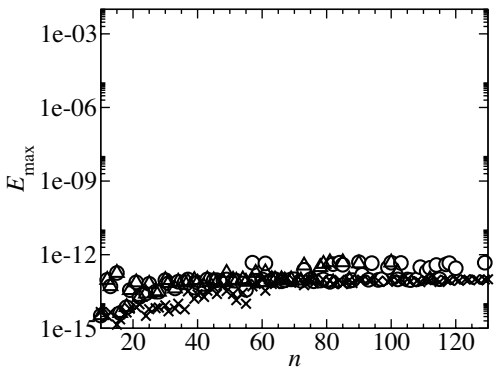

Fig. 1. Maximal residual norm $E_{\max}$ in the case of test matrix (15) with $\gamma = 1.6$. ◯ : `sneig_J`, △ : `sneig_I`, × : `sii`.

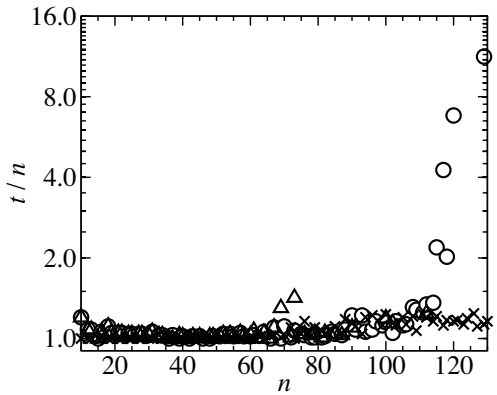

Fig. 2. Ratio of the iteration number t to the matrix size n in the case of test matrix (15) with $\gamma = 1.6$. ◯ : `sneig_J`, △ : `sneig_I`, × : `sii`.

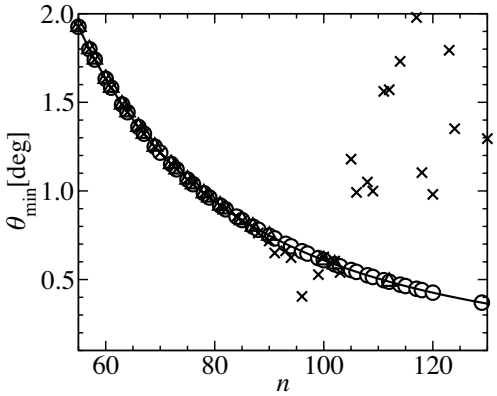

Fig. 3. Minimal angle $\theta_{\min}$ among the eigenvectors in the case of test matrix (15) with $\gamma = 1.6$. ◯ : `sneig_J`, △ : `sneig_I`, × : `sii`, – : Maple.

matrices appeared in (6), (9), (10) are computed by using the solver of the linear equations with the help of the LAPACK routine `zgesv`. In the `sii` algorithm, an eigenvalue and its corresponding eigenvector are computed by the LAPACK routine `zgeev` and the inverse iteration based on (10), respectively. The initial vector $\boldsymbol{x}^{(0)}$ in (10) is changed if $E_k \geq \epsilon_{\mathrm{good}}$ or $\theta \leq \theta_{\mathrm{same}}$. Let t be the iteration number of (6), (9), (10) for computing all eigenpairs.

Figs. 1–3 describe the numerical properties in the case where $\gamma = 1.6$. No plotted points exist for the case where the `sneig_`$*$ algorithms stop without computing all eigenpairs. Fig. 1 shows the maximal residual norm

$$E_{\max} = \max_{k=1,\ldots,n} E_k = \max_{k=1,\ldots,n} \|A\boldsymbol{x}_k - \lambda_k \boldsymbol{x}_k\|_\infty. \tag{16}$$

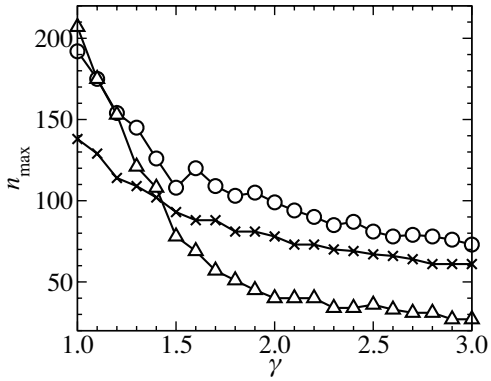

Fig. 4. Computable matrix size $n_{\max}$. ◯ : `sneig_J`, △ : `sneig_I`, × : `sii`.

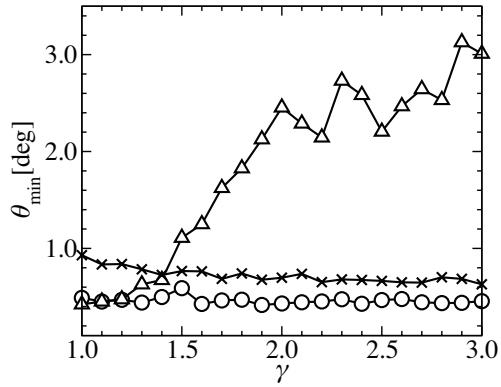

Fig. 5. Minimal angle $\theta_{\min}$ in the case of $n = n_{\max}$. ◯ : `sneig_J`, △ : `sneig_I`, × : `sii`.

By using the `sneig_`$*$, the `sii` algorithms, $E_{\max}$ becomes $O(10^{-13})$. Though, in the `sneig_`$*$ and the `sii` algorithms, the eigenvectors seem to be computed with high accuracy, it is necessary to investigate the angles among the computed eigenvectors. This is shown in the later discussion. Fig. 2 shows the ratio of t to the matrix size n for several n. For $n \leq 40$, t slightly increases in the `sneig_`$*$ algorithm. For $n \geq 60$, there is the observation that by both the `neig_`$*$ algorithm and the inverse iteration for an eigenpair, the computed eigenvector is not with high accuracy, or, is almost equal to the already obtained ones. And then the `sneig_`$*$, the `sii` algorithms require the change of the initial vector $\boldsymbol{x}^{(0)}$. This flow is surely dependent on the angles among the eigenvectors. Let $\theta_{\min} := \min_{1 \leq i < j \leq n} \mathrm{angle}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ be the minimal angle among the eigenvectors. Fig. 3 shows the relationship between n and $\theta_{\min}$. Fig. 3 also includes the numerical results by Maple, where 100 digits arithmetic is performed in Maple. For $n \geq 60$, $\theta_{\min}$ is about 1°. A part of eignevectors are nearly parallel. As the matrix size n increases, $\theta_{\min}$ by Maple becomes smaller. All eigenvectors computed by the `sneig_`$*$ algorithm are near to those by Maple. The minimal angle $\theta_{\min}$ by the `sii` algorithm are different from that by Maple. Let $\theta^*_{\min}$ be the minimal angle among the eigenvectors by Maple. In the `sii` algorithm, for $n \geq 90$, $\theta_{\min}$ does not satisfy $|\theta_{\min} - \theta^*_{\min}| < 0.03^\circ$.

Next we investigate the computable maximal matrix size $n_{\max}$ as the entry γ in (15) becomes larger. We regard that the algorithms fail if $|\theta_{\min} - \theta^*_{\min}| \geq 0.03^\circ$ as the matrix size n grows larger. Fig. 4 shows the relationship between γ and $n_{\max}$. For $\gamma > 1.2$, $n_{\max}$ in the `sneig_J` algorithm is much larger than that in the `sneig_I` algorithm. And $n_{\max}$ in the `sii` algorithm is about 0.79 times as that in the `sneig_J` algorithm. Fig. 5 shows the relationship between γ and $\theta_{\min}$ in the case where the matrix size is equal to $n_{\max}$. For all γ, $\theta_{\min}$ in the `sneig_J` algorithm are almost 0.46°. For $\gamma > 1.2$, $\theta_{\min}$ in the `sneig_I` algorithm is larger than that in the `sneig_J` algorithm. And $\theta_{\min}$ in the `sii` algorithm is slightly larger than that in the `sneig_J` algorithm. Compared with the results by Maple, it is obvious that the `sii` algorithm is not with high accuracy. Consequently, the `sneig_J` algorithm generates the most accurate eigendecomposition among three algorithms.

5. Conclusion

In this paper, we design new eigendecomposition algorithms based on solving the nonlinear quadratic systems. In our algorithms, the existence space of eigenvectors is restricted to the suitable hyperplane. The eigenvalue problem is replaced with solving the quadratic systems. An eigenpair is computed through solving the quadratic systems with the help of the Newton iterative method. The normal vector of the hyperplane is given from the orthogonal complement of the space spanned by the already obtained eigenvectors. The solutions of the quadratic systems are not equal to the already obtained eigenvectors. Of course, for any initial vector, the computed vector by the Newton iterative method does not become the already obtained eigenvectors. Consequently, all eigenpairs are sequentially computable. Our algorithms are two types named the `sneig_J` algorithm with the Newton iterative method and the `sneig_I` algorithm with a modified inverse iteration. Our algorithms are compared with the standard inverse iteration from the viewpoint of numerical accuracy. It is shown that the `sneig_J` algorithm is the best algorithm for computing all eigenvectors with high accuracy in the case where the minimal angle among the eigenvectors is small.

Acknowledgments

The authors thank the reviewer for his carefully reading and helpful suggestions.

References

[1] M. B. Elgindi and A. Kharab, The quadratic method for computing the eigenpairs of a matrix, Int. J. Comput. Math., **73** (2000), 517–530.
[2] F. Chatelin, Eigenvalues of Matrices (in Japanese), Springer-Verlag, Tokyo, 2003.
[3] K. Falconer, Fractal Geometry, Mathematical Foundations and Applications, Second Edition, John Wiley & Sons, England, 2003.
[4] G. H. Golub and C. F. Van Loan, Matrix Computations, Third Edition, The Johns Hopkins Univ. Press, Baltimore and London, 1996.
[5] LAPACK, http://www.netlib.org/lapack/.
[6] M. H. Gutknecht, Variants of BiCGSTAB for matrix with complex spectrum, SIAM J. Sci. Comput., **14** (1993), 1020–1033.