Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.2018.s11

Vol. **11**, No. 4, pp. 877-894
November 2018

Computing the Maximal Eigenpairs of Large Size Tridiagonal Matrices with $\mathcal{O}(1)$ Number of Iterations

Tao Tang[1,*] and Jiang Yang[1]

[1] *Department of Mathematics, Southern University of Science and Technology Shenzhen, China*

Received 15 September 2017; Accepted (in revised version) 19 December 2017

Dedicated to Professor Xiaoqing Jin on the occasion of his 60th birthday

Abstract. In a series of papers, Chen [4–6] developed some efficient algorithms for computing the maximal eigenpairs for tridiagonal matrices. The key idea is to explicitly construct effective initials for the maximal eigenpairs and also to employ a self-closed iterative algorithm. In this paper, we extend Chen's algorithm to deal with large scale tridiagonal matrices with super-/sub-diagonal elements. By using appropriate scalings and by optimizing numerical complexity, we make the computational cost for each iteration to be $\mathcal{O}(N)$. Moreover, to obtain accurate approximations for the maximal eigenpairs, the total number of iterations is found to be independent of the matrix size, i.e., $\mathcal{O}(1)$ number of iterations. Consequently, the total cost for computing the maximal eigenpairs is $\mathcal{O}(N)$. The effectiveness of the proposed algorithm is demonstrated by numerical experiments.

AMS subject classifications: 60J60, 34L15

Key words: Maximal eigenpair, large size tridiagonal matrix, scaling, complexity.

1. Introduction

This paper is concerned with computing the maximal eigenpairs of tridiagonal matrices, aiming at an $\mathcal{O}(N)$ complexity for a matrix of size $N \times N$. The eigenpair here means the twins consist of eigenvalue and its eigenvector, and the maximal eigenpair indicates the largest eigenvalue and the corresponding eigenvector. The problem of computing the maximal eigenpairs has been a classical subject treated in most books of numerical analysis. The methods for this problem that are discussed most commonly are the power method, the inverse method, the Rayleigh quotient method, and some hybrid method, see, e.g., [1, 16, 17]. Finding the largest eigenpairs has many applications in signal processing, control, and recent development of Google's PageRank algorithm. On the other hand,

*Corresponding author. *Email addresses:* tangt@sustc.edu.cn (T. Tang), yangj7@sustc.edu.cn (J. Yang)

numerous methods exist for the numerical computation of the eigenvalues of a real tridiagonal matrix to high accuracy. It is well known that a transformation that reduces a general matrix to Hessenberg form will reduce a Hermitian matrix to tridiagonal form. So, many eigenvalue algorithms, when applied to a Hermitian matrix, reduce the input Hermitian matrix to tridiagonal form as a first step. On the computational side, much efforts have been made to deal with symmetric tridiagonal cases, see, e.g., [2, 14, 15], typically requiring $\mathcal{O}(N^2)$ operations [11], although fast algorithms exist which require $\mathcal{O}(N \ln N)$ [7].

In [4], an efficient algorithm was introduced to compute the maximal eigenpair of the tridiagonal matrices with positive sub-diagonal elements. The key contribution in [4] is the explicit construction of the initial values which makes the relevant iterative algorithms unexpectedly efficient. In a following-up article [5], Chen proposed two global algorithms for computing the maximal eigenpair in a rather general setup, including a class of real (with some negative off-diagonal elements) or complex matrices.

The main idea of this work is from [5] which gives elegant formulas to approximate the largest eigenpairs in an iterative manner. It is found that for computing large scale matrices his work needs some computational polishing since possible overflows may occur when the matrix size becomes very large. The main contributions of this work is to introduce appropriate scalings to reduce the numerical instability. To make the algorithm more attractive, we also take care of the numerical complexity so that the best possible $\mathcal{O}(N)$ operations can be achieved. For nonsymmetric tridiagonal matrices, we introduce a diagonal similarity transformation to convert them into symmetric ones. Note that this symmetrization procedure can be implemented with $\mathcal{O}(N)$ operations.

One application of the fast algorithm developed in this work is to compute the largest eigenpairs of the tridiagonal random matrices. In probability theory and mathematical physics, a random matrix is a matrix-valued random variable, which in some cases share the same eigenvalues with certain tridiagonal matrices. We can take an example of the Gaussian Unitary Ensemble (GUE) which is defined as the $n \times n$ Hermitian matrices X, where the diagonal elements x_{jj} and the upper triangular elements $x_{jk} = u_{jk} + iv_{jk}$ are independent Gaussians with zero-mean. To compute the eigenpairs of X the main problem is due to the computational requirements and the memory requirements which grow fast with N. As pointed out in [9, 10] computing all the eigenvalues of a full Hermitian matrix requires a computing time proportional to N^3. This means that it will take many days to create a smooth eigenvalue histogram by simulation, even for relatively small values of N, say $N = 500$. To improve upon this situation, another matrix can be studied that has the same eigenvalue distribution as X above. In [8], it was shown that this is true for the following symmetric matrix when $\beta = 2$:

$$H_\beta \sim \frac{1}{2}\begin{bmatrix} N(0,2) & \chi_{(n-1)\beta} & & & & & \\ \chi_{(n-1)\beta} & N(0,2) & \chi_{(n-2)\beta} & & & & \\ & \chi_{(n-2)\beta} & N(0,2) & \chi_{(n-3)\beta} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \chi_{2\beta} & N(0,2) & \chi_\beta & \\ & & & & \chi_\beta & N(0,2) & \end{bmatrix}.$$

This matrix has a tridiagonal sparsity structure, and only $2N$ double-precision numbers are required to store an instance of it. The time for computing the largest eigenvalue is proportional to N, either using Krylov subspace based methods or the method of bisection. However, these methods are of first-order rate of convergence, and need long computational time if higher accuracy (say 10^{-8}) is required. The method proposed in Chen [4,5] and the present work can produce highly accurate largest eigenpair with much less computational time.

2. Chen's algorithm

Consider an $(N+1)\times(N+1)$ square matrix $A=(a_{ij})$ satisfying $a_{ij}=0$ for $|i-j|>1$. By using a shift $Q:=A-mI$, where I is the $(N+1)\times(N+1)$ identity matrix and

$$m=\max_{0\le i\le N}\sum_{j=1}^{N}a_{i,j}, \tag{2.1}$$

we may assume that

$$Q=\begin{pmatrix} -(b_0+c_0) & b_0 & 0 & 0 & \cdots \\ a_1 & -(a_1+b_1+c_1) & b_1 & 0 & \cdots \\ 0 & a_2 & -(a_2+b_2+c_2) & b_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & a_N & -(a_N+c_N) \end{pmatrix}, \tag{2.2}$$

where $a_i,b_i>0,c_i\ge 0$ but $c_j\not\equiv 0$.

Define, for $1\le n\le N$,

$$\mu_0=1, \qquad \mu_n=\mu_{n-1}\frac{b_{n-1}}{a_n}, \tag{2.3a}$$

$$r_0=1+\frac{c_0}{b_0}, \qquad r_n=1+\frac{a_n+c_n}{b_n}-\frac{a_n}{b_n r_{n-1}}, \tag{2.3b}$$

$$h_0=1, \qquad h_n=h_{n-1}r_{n-1}. \tag{2.3c}$$

Furthermore, define

$$\phi_n=\sum_{k=n}^{N}\frac{1}{h_k h_{k+1}\mu_k b_k}, \qquad 0\le n\le N \tag{2.4}$$

with

$$h_{N+1}=c_N h_N+a_N(h_N-h_{N-1}), \quad b_N=1.$$

Theorem 2.1. ([4])*For a given tridiagonal matrix A, define* $m,(a_i,b_i,c_i)$ *as in* (2.1)-(2.2). *Set*

$$\tilde{v}_0(i)=h_i\sqrt{\phi_i}, \quad 0\le i\le N; \qquad v_0=\frac{\tilde{v}_0}{\sqrt{\tilde{v}_0^*\tilde{v}_0}}, \tag{2.5}$$

where h_i, ϕ_i are defined by (2.3a)-(2.4). *Furthermore, let*

$$z_0 = \frac{1}{\delta_0}; \quad \delta_0 = \max_{0\le n\le N}\left[\sqrt{\phi_n}\sum_{k=0}^{n}\mu_k h_k^2\sqrt{\phi_k} + \frac{1}{\sqrt{\phi_n}}\sum_{j=n+1}^{N}\mu_j h_j^2\phi_j^{3/2}\right]. \tag{2.6}$$

With the initial values v_0 and shift δ_0, we perform the shifted inverse power method on matrix Q given in (2.2), *which produces a vector sequence of $\{v_k\}$ and associated Rayleigh quotient $\{z_k\}$. Then $m - z_k$ converges to the largest eigenvalue of A and v_k converges to the corresponding eigenvector.*

Although the above theorem gives a useful initial eigenpair approximation, however, using the inverse iteration in general requires many iterations. This in turn slows down the convergence of the computation. To improve this, a more powerful (robust and accurate) method was introduced in [5]. First, we take the similarity transformation on Q using the diagonal matrix $\mathrm{Diag}(h_i)$, whose main diagonal is by the vector h, i.e.,

$$\tilde{Q} = \mathrm{Diag}(h_i)^{-1} Q \mathrm{Diag}(h_i). \tag{2.7}$$

It is easy to check that $\tilde{Q}$ is of the following form

$$\tilde{Q} = \begin{pmatrix} -\tilde{b}_0 & \tilde{b}_0 & 0 & 0 & \cdots \\ \tilde{a}_1 & -(\tilde{a}_1+\tilde{b}_1) & \tilde{b}_1 & 0 & \cdots \\ 0 & \tilde{a}_2 & -(\tilde{a}_2+\tilde{b}_2) & \tilde{b}_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \tilde{a}_N & -(\tilde{a}_N+\tilde{b}_N) \end{pmatrix}, \tag{2.8}$$

where $\tilde{a}_i, \tilde{b}_i > 0$. We then define

$$v_0 = 1, \qquad v_n = v_{n-1}\frac{\tilde{b}_{n-1}}{\tilde{a}_n}, \qquad 1 \le n \le N, \tag{2.9a}$$

$$\varphi_n = \sum_{k=n}^{N}\frac{1}{v_k\tilde{b}_k}, \qquad 0 \le n \le N. \tag{2.9b}$$

Similar as before, we define

$$\tilde{\delta}_0 = \max_{0\le k\le N}\left[\sqrt{\varphi_k}\sum_{i=0}^{k} v_i\sqrt{\varphi_i} + \frac{1}{\sqrt{\varphi_k}}\sum_{j=k+1}^{N} v_j\varphi_j^{3/2}\right]. \tag{2.10}$$

With the above preparations, we are now ready to state Chen's algorithm:

Note the key contribution of the above algorithm is the use of the iteration (2.12), which was first introduced in [5].

Algorithm 2.1 Chen's Algorithm.

Step 1 Choose the initial eigenpair using (2.9b) and (2.10): $\omega^{(0)} = \sqrt{\varphi}$,

$$\nu^{(0)} = \frac{\omega^{(0)}}{\|\omega^{(0)}\|_{\nu,2}}, \qquad z^{(0)} = \frac{1}{\tilde{\delta}_0}.$$

Step 2 For $n = 1, 2, \cdots$, solve the linear equation

$$\left(-\tilde{Q} - z^{(n-1)} I \right) \omega = \nu^{(n-1)}, \tag{2.11}$$

and then define $\nu^{(n)} = \omega / \|\omega\|_{\nu,2}$.

Step 3 Update $z^{(n)} = 1/\tilde{\delta}_n$ with

$$\tilde{\delta}_n = \max_{0 \le k \le N} \frac{1}{\nu_k^{(n)}} \left[\varphi_k \sum_{i=0}^{k} \nu_i \nu_i^{(n)} + \sum_{j=k+1}^{N} \nu_j \varphi_j \nu_j^{(n)} \right], \tag{2.12}$$

until $\left| z^{(n)} - z^{(n-1)} \right|$ is smaller than some given tolerence.

Step 4 Output the largest eigenpair as $\rho(A) = m - z^{(n)}$, $g = \mathrm{Diag}(h_i)\nu^{(n)}$.

3. Practical issues relevant to large size

We begin by discussing the total number of operations used in Chen's Algorithm introduced in the last section. It is aimed to reduce the complexity of computing to $\mathcal{O}(N)$ in total.

First, with the recursive formula (2.3b) and (2.3c), it only requires $2N$ operations to get the sequences r and h. After that, for the similarity transformation (2.7), we know

$$\tilde{Q}_{ij} = Q_{ij} \frac{h_j}{h_i}.$$

Note that Q is a tridiagonal matrix. Then the transformation only needs to calculate, for $0 \le i < N$,

$$\tilde{Q}_{i,i+1} = Q_{i,i+1} \frac{h_{i+1}}{h_i} = Q_{i,i+1} r_i, \qquad \tilde{Q}_{i+1,i} = Q_{i+1,i} \frac{h_i}{h_{i+1}} = \frac{Q_{i+1,i}}{r_i}, \tag{3.1}$$

which also consumes $2N$ operations.

The second part is to prepare ν and φ given by (2.9a)-(2.9b). Obviously, it takes N operations to compute ν. For φ, using the backward formula

$$\varphi_n = \varphi_{n+1} + \frac{1}{\nu_n \tilde{b}_n} \tag{3.2}$$

yields N operations to obtain φ. We further employ Thomas algorithm for solving the tridiagonal system (2.11), then an additional $\mathcal{O}(N)$ operations is required.

3.1. Scaling

In this part, we will discuss a scaling issue relevant to the above algorithm. Consider a case that $\tilde{b}_j \geq 4$ and $0 < \tilde{a}_j \leq 2$ for all j. In this cae, it can be verified that $\nu_n \geq 4^n/2^n = 2^n$. On the other hand,

$$\varphi_n = \sum_{k=n}^{N} \frac{1}{\nu_k \tilde{b}_k} \leq \sum_{k=n}^{N} \frac{1}{2^k 4} \leq \frac{1}{2^{n+1}},$$

which is obviously too small for large values of n.

Since ν_n is too large and φ_n is too small, we need to avoid using them explicitly in the formulas. This can be done by introducing some proper scaling factors. To this end, we first observe that the growth rate of ν_n is almost the same as the decay rate of φ_n. Hence, we introduce a new variable

$$W_n = \nu_n \varphi_n, \quad 0 \leq n \leq N. \tag{3.3}$$

Observe

$$W_N = \mu_N \varphi_N = \frac{1}{\tilde{b}_N}. \tag{3.4}$$

For $0 \leq n < N-1$, using the backward formula (3.2) gives

$$\begin{aligned} W_n =& \nu_n \left(\varphi_{n+1} + \frac{1}{\nu_n \tilde{b}_n} \right) \\ =& \frac{1}{\tilde{b}_n} + \frac{\nu_n}{\nu_{n+1}} \left(\nu_{n+1} \varphi_{n+1} \right) \\ =& \frac{1}{\tilde{b}_n} + \frac{\tilde{a}_{n+1}}{\tilde{b}_n} W_{n+1}. \end{aligned} \tag{3.5}$$

The following ratio will be used in the computations in the following subsection:

$$T_n = \frac{\varphi_{n+1}}{\varphi_n}, \quad 0 \leq n \leq N-1. \tag{3.6}$$

It is easy to verify that when $0 \leq n < N-1$

$$T_n = \frac{W_{n+1}}{W_n} \frac{\nu_n}{\nu_{n+1}} = \frac{\tilde{a}_{n+1}}{\tilde{b}_n} \frac{W_{n+1}}{W_n}. \tag{3.7}$$

3.2. $\mathcal{O}(N)$ complexity

Note that the most expensive part in the computations is to obtain $\tilde{\delta}_k$ using (2.12). Technically, if it is computed in a straightforward manner, i.e., term by term using (2.10) and (2.12), then the resulting complexity will be $\mathcal{O}(N^2)$. Below we will demonstrate how to obtain δ_k using $\mathcal{O}(N)$ operations, which is the core of complexity reduction.

Observe that each term in computing δ_k ends up with evaluating the following two sequences:

$$\alpha_n = x_n \sum_{k=0}^{n} y_k, \qquad \beta_n = p_n \sum_{k=n+1}^{N} q_k.$$

The basic idea is to find recursive formula for the series $\{\alpha_n\}$ and $\{\beta_n\}$ recursively. For $\{\alpha_n\}$, we have

$$\begin{aligned}\alpha_{n+1} =& x_{n+1} \sum_{k=0}^{n+1} y_k \\ =& x_{n+1} y_{n+1} + \frac{x_{n+1}}{x_n} \left(x_n \sum_{k=0}^{n} y_k \right) \\ =& x_{n+1} y_{n+1} + \frac{x_{n+1}}{x_n} \alpha_n. \end{aligned} \tag{3.8}$$

Similarly, we have

$$\beta_n = p_n q_{n+1} + \frac{p_n}{p_{n+1}} \beta_{n+1}. \tag{3.9}$$

Using the forward recursive formula for $\{\alpha_n\}$ and backward recursive formula for $\{\beta_n\}$ needs $\{\alpha_n\}$ and $\{\beta_n\}$ with $O(N)$ operations. Then another N operations give $\alpha_n + \beta_n, 0 \le n \le N$. By choosing the maximum value from $\alpha_n + \beta_n$ yields δ_k.

If Chen's Algorithm converges within a constant number of iterations (which will be demonstrated numerically in the next section), then the overall complexity is $\mathcal{O}(N)$.

It is seen from Sect. 3.1 that the scaled variables W and T are now introduced. Consequently, we need to reformulate δ_k in (2.12) using W and T. We follow basic idea of deriving the recursive formulas for α in (3.8) and β in (3.9). In δ_1 (2.10), we denote the former part as $\eta_n^{(1)}$ and the latter part as $\eta_n^{(2)}$, i.e.,

$$\eta_n^{(1)} = \sqrt{\varphi_n} \sum_{i=0}^{n} \nu_i \sqrt{\varphi_i}, \qquad \eta_n^{(2)} = \frac{1}{\sqrt{\varphi_n}} \sum_{j=n+1}^{N} \nu_j \varphi_j^{3/2}. \tag{3.10}$$

First $\eta_0^{(1)} = W_0$, and for $n \ge 0$

$$\begin{aligned}\eta_{n+1}^{(1)} =& W_{n+1} + \sqrt{\varphi_{n+1}} \sum_{i=0}^{n} \nu_i \sqrt{\varphi_i} \\ =& W_{n+1} + \sqrt{\frac{\varphi_{n+1}}{\varphi_n}} \left(\sqrt{\varphi_n} \sum_{i=0}^{n} \nu_i \sqrt{\varphi_i} \right) \\ =& W_{n+1} + \sqrt{T_n} \eta_n^{(1)}. \end{aligned}$$

Similarly, for $\eta^{(2)}$ when $0 \le n < N-1$ we have

$$\eta_n^{(2)} = T_n \left(\eta_{n+1}^{(2)} + W_{n+1} \right),$$

with $\eta_N^{(2)} = 0$. Note that

$$\tilde{\delta}_1 = \max_{0\le k\le N}\left(\eta_k^{(1)} + \eta_k^{(2)}\right).$$

To evaluate $\tilde{\delta}_n$ $(n > 1)$, we define

$$\xi_k^{(1)} = \frac{\varphi_k}{v_k^{(n)}}\sum_{i=0}^{k} v_i v_i^{(n)}, \qquad \xi_k^{(2)} = \frac{1}{v_k^{(n)}}\sum_{j=k+1}^{N} v_j \varphi_j v_j^{(n)}. \tag{3.11}$$

It can be easily verified that for $k \ge 0$: Clearly, $\xi_0^1 = W_0$ and for $k \ge 0$

$$\xi_{k+1}^{(1)} = \frac{v_k^{(n)}}{v_{k+1}^{(n)}} T_k \xi_k^{(1)} + W_{k+1}, \qquad \xi_k^{(2)} = \frac{v_{k+1}^{(n)}}{v_k^{(n)}}(\xi_{k+1}^{(2)} + W_{k+1}) \tag{3.12}$$

with $\xi_0^{(1)} = W_0$ and $\xi_N^{(2)} = 0$. Thus, it follows from (2.12) that

$$\delta_n = \max_{0\le k\le N}\left(\xi_k^{(1)} + \xi_k^{(2)}\right).$$

Note that in the reformulation of δ_n another sequence $v_{k+1}^{(n)}/v_k^{(n)}$ is introduced. It is expected that for smooth variation of the eigen-approximation $v^{(n)}$ this ratio should be bounded uniformly.

3.3. Non-symmetric matrix

Our numerical experiments demonstrated that the algorithm described at the end of Section 2 together with above two subsections work well for a class of large size symmetric tridiagonal matrices. However, for non-symmetric case, the result is quite unsatisfactory. The main reason can be seen from the following simple example. Consider the following simple non-symmetric tridiagonal matrix:

$$\begin{pmatrix} c & b & 0 & 0 & \cdots \\ a & c & b & 0 & \cdots \\ 0 & a & c & b & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a & c \end{pmatrix}_{N\times N}.$$

It is known that the eigenvalue of the above matrix is given by, (see, e.g. [12]),

$$\lambda_j = c + 2\sqrt{ab}\cos\frac{\pi j}{N+1}, \quad 1 \le j \le N,$$

and the associated eigenvector is given as

$$\vec{v}_j = \left(\left(\frac{a}{b}\right)^{\frac{1}{2}}\sin\left(\frac{j\pi}{N+1}\right), \left(\frac{a}{b}\right)^{\frac{2}{2}}\sin\left(\frac{2j\pi}{N+1}\right), \cdots, \left(\frac{a}{b}\right)^{\frac{N}{2}}\sin\left(\frac{Nj\pi}{N+1}\right)\right)^T.$$

Note that each element of the eigenvector has an exponential factor. Consider the case that $a/b > 1$. Then most of the elements will blow up if N is large. Likewise, in the case $0 < a/b < 1$, most of the elements will be recognized by machine as 0, which also causes numerical difficulty.

To fix the above problem, we introduce a well-known diagonal scaling technique given in many textbook of Linear Algebra, which is a diagonal similarity transformation to convert the non-symmetric tridiagonal matrix into a symmetric one. More precisely, consider a general tridiagonal matrix with positive off-diagonal elements:

$$A = \begin{pmatrix} c_1 & b_2 & 0 & 0 & \cdots \\ a_2 & c_2 & b_3 & 0 & \cdots \\ 0 & a_3 & c_3 & b_4 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & a_{N-1} & c_{N-1} & b_N \\ 0 & 0 & \cdots & a_N & c_N \end{pmatrix}_{N\times N}. \tag{3.13}$$

Define the diagonal matrix P in form of

$$P = \operatorname{diag}\left(1, \sqrt{\frac{b_2}{a_2}}, \sqrt{\frac{b_2 b_3}{a_2 a_3}}, \cdots, \sqrt{\frac{b_2 b_3 \cdots b_N}{a_2 a_3 \cdots a_N}}\right). \tag{3.14}$$

Thus,

$$B = PAP^{-1} = \begin{pmatrix} c_1 & \sqrt{a_2 b_2} & 0 & 0 & \cdots \\ \sqrt{a_2 b_2} & c_2 & \sqrt{a_3 b_3} & 0 & \cdots \\ 0 & \sqrt{a_3 b_3} & c_3 & \sqrt{a_4 b_4} & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{a_{N-1} b_{N-1}} & c_{N-1} & \sqrt{a_N b_N} \\ 0 & 0 & \cdots & \sqrt{a_N b_N} & c_N \end{pmatrix}_{N\times N}. \tag{3.15}$$

This end up with a symmetric matrix. Obviously, the total computation of this similarity transformation is also $\mathcal{O}(N)$.

Note the above similarity transformation works for A with positive off-diagonal elements, or in a loose sense, by only requiring $a_j \cdot b_j > 0, 2 \le j \le N$.

4. Numerical experiments

In this section, we will take several numerical examples to demonstrate the performance of Chen's Algorithm with proper scalings introduced above.

Example 4.1. We start with a simple symmetric tridiagonal matrix:

$$S_{N-1} = \begin{pmatrix} 4 & 1 & 0 & 0 & \cdots \\ 1 & 4 & 1 & 0 & \cdots \\ 0 & 1 & 4 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 4 \end{pmatrix}_{N\times N}. \tag{4.1}$$

As shown previously the eigenvalues of the above matrix is

$$\lambda_j = 4 + 2\cos\frac{\pi j}{N+1}, \quad 1 \le j \le N.$$

The largest eigenvalue is

$$\lambda_N = 4 + 2\cos\frac{\pi}{N+1} \to 6 \ \text{ as } N \to \infty.$$

In this example, we set $m = \|A\|_\infty = 6$. Thus, define

$$Q_N = A - mI = \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix}. \tag{4.2}$$

Hence we have $a_i = b_i = 1$ and $c_i = 0$ except $c_0 = c_N = 1$. It can be easily verified that those numerical parameters given in Sect. 2 can be evaluated analytically. For example, $\mu_j \equiv 1$,

$$r_0 = 2, \ \ h_0 = 1; \quad r_n = \frac{n+2}{n+1}, \ \ h_n = n+1, \quad 1 \le n \le N.$$

This gives that

$$\frac{h_i}{h_j} = \frac{i+1}{j+1}.$$

The numerical result is shown in Table 1. It is observed that the initial guess is very close to the largest eigenvalue, which leads to the desired eigenvalue with only 2 *or* 3 *iterations*. It is also interesting to see that the larger the matrix size the faster the convergence for the maximal value.

Example 4.2. Consider the tridiagonal matrix in Gauss-Laguerre quadrature [13]

$$G_{N+1} = \begin{pmatrix} \alpha_0 & \beta_0 & 0 & 0 & \cdots \\ \beta_0 & \alpha_1 & \beta_1 & 0 & \cdots \\ 0 & \beta_1 & \alpha_2 & \beta_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \beta_{N-1} & \alpha_N \end{pmatrix}, \tag{4.3}$$

Table 1: (Example 4.1) Numerical performance of rescaled algorithm for Example 4.1.

N	10^2	10^3	10^4
$\lambda_{\max}$	5.999032564583976	5.999990150113323	5.999999901323693
z_0	5.999132539362944	5.999991169380220	5.999999911534810
z_1	5.999034149120376	5.999990166273074	5.999999901485584
z_2	5.999032564938140	5.999990150116939	5.999999901323729

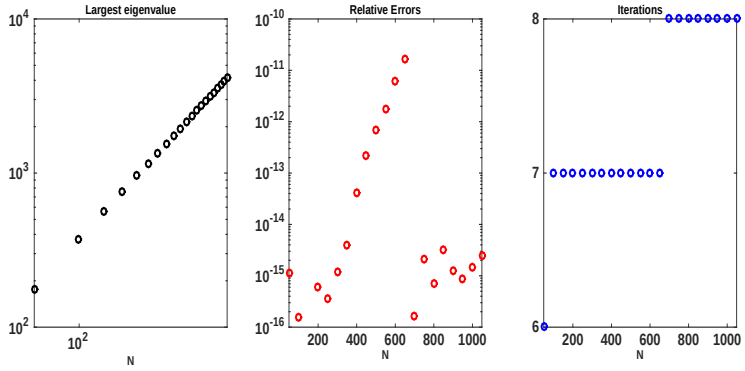

Figure 1: (Example 4.2) the largest eigenvalue in Gauss-Laguerre quadrature versus the matrix size N (left), the corresponding relative errors (middle), and the total number of iterations (right).

where

$$\alpha_i = 2i+1+\alpha, \quad \beta_i = \sqrt{(i+1)(i+1+\alpha)}, \quad i \geq 0,$$

and α is chosen as $\alpha = -0.75$.

In Fig. 1, we display the growth of the largest eigenvalue with respect to the matrix size N. It is noticed that the eigenvalue grows almost quadratically with respect to N. By use of the method given by Chen [5] together with scaling and recurrence techniques proposed in Section 3, the corresponding approximation errors can reach machine accuracy after 8 iterations for $N = 1000$.

We further increase the value of N to 10,000. The rescaled variables W and T and the original variables μ and φ (with logarithmic scale) are shown in Fig. 2. It is seen that the original variable μ grows exponentially and φ decays exponentially, and the rescaled variables are well behaved.

In Table 2, we present numerical results for Example 4.2 with $N = 10,000$ using Chen [5] together with scaling and recurrence techniques proposed in Section 3. It is observed that very accurate approximations are obtained after 10 iterations. In fact, as the largest eigenvalue is of order 10^4 it takes a couple of more iterations for this case.

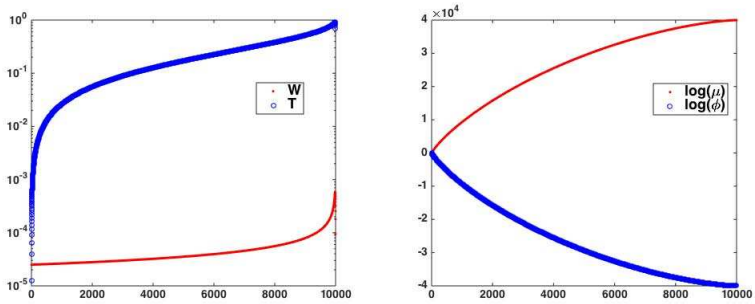

Figure 2: (Example 4.2) The rescaled variables W and T versus the matrix size N (left) and the original variables $\log(\mu)$ and $\log(\varphi)$ (right).

Table 2: (Example 4.2) Numerical approximation for the largest eigenvalue of the Gauss-Laguerre quadrature problem with $N = 10000$. The exact value of the largest eigenvalue is $\lambda_{\max} \approx 3.986965228013262 * 10^4$.

# iteration	Numerical	l^∞ Absolute Error	Relative Error
0	3.988534579649927e+04		
1	3.988534579649927e+04	15.69	3.936e-04
2	3.988349568611173e+04	13.84	3.472e-04
3	3.988099826354650e+04	11.35	2.846e-04
4	3.987815640126808e+04	8.504	2.133e-04
5	3.987522432114032e+04	5.572	1.398e-04
6	3.987253111806946e+04	2.879	7.221e-05
7	3.987056788545732e+04	9.1561e-01	2.296e-05
8	3.986975020621370e+04	9.793e-02	2.456e-06
9	3.986965323980745e+04	9.597e-04	2.407e-08
10	3.986965228020606e+04	7.344e-08	1.842e-12

Example 4.3. Consider a random matrix taken from [10]

$$H = \frac{1}{2\sqrt{n}} \begin{bmatrix} N(0,2) & \chi_{n-2} & & & & & \\ \chi_{n-2} & N(0,2) & \chi_{n-3} & & & & \\ & \chi_{n-3} & N(0,2) & \chi_{n-4} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \chi_{n-k} & N(0,2) & \chi_{n-k-1} & \\ & & & & \chi_{n-k-1} & N(0,2) & \end{bmatrix},$$

where $N(0,2)$ is a zeros-mean Gaussian with variance 2, and χ_r is the square-root of a χ^2-distributed number with r degrees of freedom.

In the first part of this example, a size of $10^3 \times 10^3$ matrix is considered, i.e., $n = 10^6$ and $k = 1000$. In Fig. 3, we first show the diagonal elements and the distribution of eigenvalues, where it is observed that all off-diagonal elements are positive and uniformly

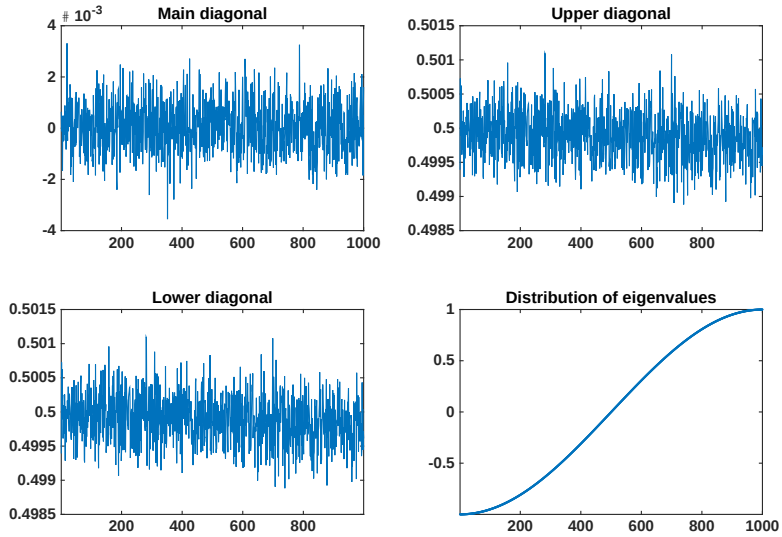

Figure 3: (Example 4.3) Elements and the distribution of eigenvalues.

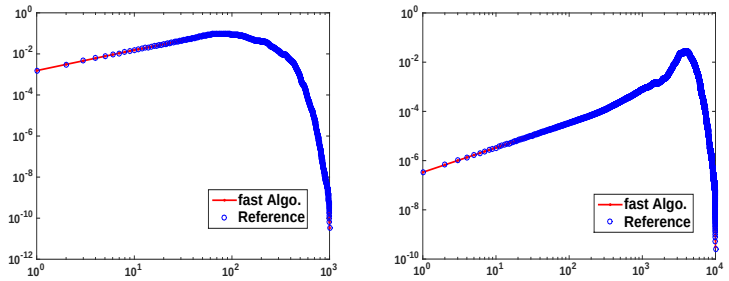

Figure 4: (Example 4.3) Comparison of computed eigenvector and the reference one for left: $N = 1000$ and right: $N = 10000$.

bounded. It is further seen that the main diagonal are close to zero, and the two off-diagonals are close to 0.5. The eigenvalues are continuously distributed over $(-1, 1)$.

Table 3 presents the numerical approximations and the errors for the largest eigenvalue approximation, with not only $N = 10^3$ but also $N = 10^4$ (in the latter case $n = 10^9$ and $k = 10^4$). In both cases, machine accuracy is achieved after 4 or 5 iterations.

Fig. 4 shows the comparison of computed eigenvector and the reference one (which was obtained by MATLAB software), for both cases $N = 10^3$ and 10^4. We list two observations below. First, the numerical accuracy is very satisfactory, and secondly, the magnitude of the smallest component of the eigenvector is very small.

All three previous examples test symmetric matrix. Next we consider a non-symmetric tridiagonal matrice.

Table 3: (Example 4.3) The approximation of the largest eigenvalue, top: $N = 10^3$ and the exact $\lambda_{\max} = 0.999978982089438$, and bottom: $N = 10000$ and $\lambda_{\max} = 0.999998519368622$.

Case $N = 1000$		
# iteration	Numerical	Error
0	1.000184076339609	2.050942501710118e-04
1	1.000014529642320	3.554755288270872e-05
2	0.999983838833936	4.856744498682453e-06
3	0.999979272929260	2.908398220036190e-07
4	0.999978983270231	1.180793351984732e-09
5	0.999978982089456	1.787459069646502e-14

Case $N = 10,000$		
# iteration	Numerical	Error
0	1.000002323023300	3.803654677891899e-06
1	0.999999120257260	6.008886387354195e-07
2	0.999998623640710	1.042720878530190e-07
3	0.999998524745736	5.377114309368380e-09
4	0.999998519385876	1.725475318181680e-11

Example 4.4. We consider a simple non-symmetric tridiagonal matrix:

$$S_{N-1} = \begin{pmatrix} 4 & 2 & 0 & 0 & \cdots \\ 1 & 4 & 2 & 0 & \cdots \\ 0 & 1 & 4 & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & 4 \end{pmatrix}_{N\times N}. \tag{4.4}$$

As shown previously, the eigenvalues of the above matrix is

$$\lambda_j = 4 + 2\sqrt{2}\cos\frac{\pi j}{N+1}, \quad 1 \le j \le N,$$

and the associated eigenvector is given as

$$\vec{v}_j = \left(\left(\frac{1}{2}\right)^{\frac{1}{2}} \sin\left(\frac{j\pi}{N+1}\right), \left(\frac{1}{2}\right)^{\frac{2}{2}} \sin\left(\frac{2j\pi}{N+1}\right), \cdots, \left(\frac{1}{2}\right)^{\frac{N}{2}} \sin\left(\frac{Nj\pi}{N+1}\right) \right)^T. \tag{4.5}$$

Thus, the largest eigenvalue is

$$\lambda_N = 4 + 2\sqrt{2}\cos\frac{\pi}{N+1} \to 4 + 2\sqrt{2} \ \text{ as } N \to \infty.$$

Table 4: (Example 4.4) Numerical performance of rescaled algorithm.

N	20	200	2000
$\lambda_{\max}$	6.796835930680340	6.828081652004034	6.828423638801624
z_0	6.817202941350335	6.828427124746190	6.828427124746190
z_1	6.798773435375877	6.828146565080354	6.828427124746191
z_2	6.796852823590889	6.828083591205758	6.828425005492695
z_3	6.796835931884297	6.828081653523198	6.828423852767121
z_4			6.828423644005617

In this example, we set $m = \|A\|_\infty = 7$. Thus, define

$$Q_N = A - mI = \begin{pmatrix} -3 & 2 & 0 & 0 & \cdots \\ 1 & -3 & 2 & 0 & \cdots \\ 0 & 1 & -3 & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -3 \end{pmatrix}, \tag{4.6}$$

which is still a non-symmetric matrix.

It is seen from (4.5) that the components of the eigenvector decay exponentially. The resulting difficult poses numerical challenges even for MATLAB. To see this, we take the MATLAB function *eig(full(A))* to get the maximal eigenpair, with $N = 100$ and $N = 1000$ respectively. The comparison for the first eigenvector (which corresponds to the largest eigenvalue) is presented in Fig. 5. It is seen that when N is small, both MATLAB and the proposed algorithm provide accurate approximations. However, for larger N, the one given by MATLAB is far away from the exact one, especially for large index of components. On the other hand, our algorithm still provides accurate approximation. This is further confirmed by plotting the relative errors in Fig. 6, in which even for the last component (which is $\sim 10^{-150}$) about 10^{-6} relative error is achieved.

In the second part, we still aim to show the performance for the approximation of the largest eigenvalue. We vary N as 20, 200, and 2000. The numerical result is shown in Table 4. It is seen that the initial guess is very close to the largest eigenvalue, which leads to the desired eigenvalue super efficiently, only 3 or 4 iterations.

Note the above computations for Example 4.4 have not used the similarity transformation. Numerical experiments show that for this non-symmetric case the matrix size has to be of order of $O(10^3)$. In other words, if we choose $N = 10,000$, then convergence will be failed. As expected, this situation can be improved if the similarity transformation (3.14)-(3.15) is used. In fact, as seen from Table 5, with $10^3 \le N \le 10^5$ fast convergence is obtained. It is even observed that the initial approximation can give 10 effective digits for the very large size case $N = 10^5$.

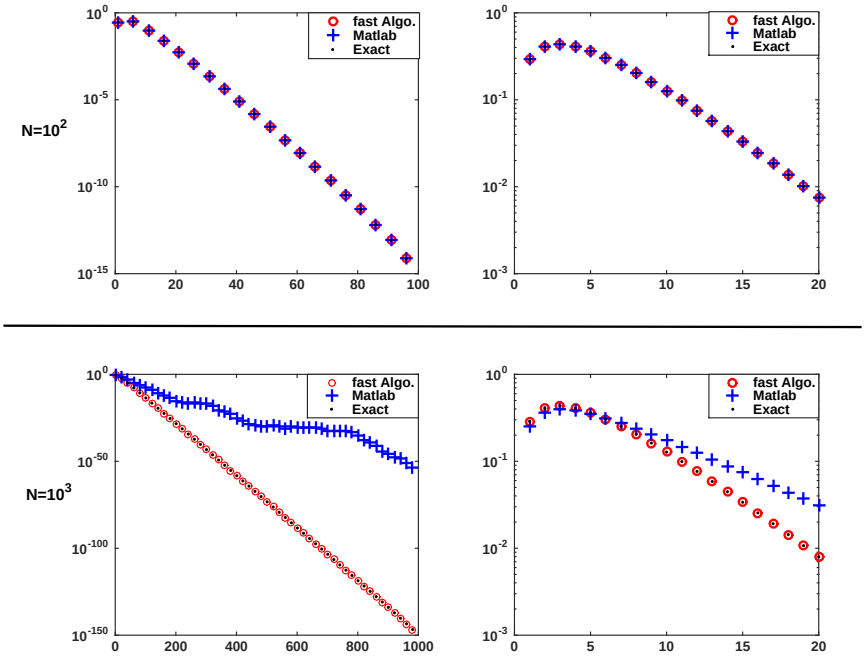

Figure 5: (Example 4.4) Comparisons of the maximal eigenvector obtained by using the revised algorithm and the MATLAB with the exact one given by (4.5), for $N = 100$ (upper) and $N = 1000$ (lower).

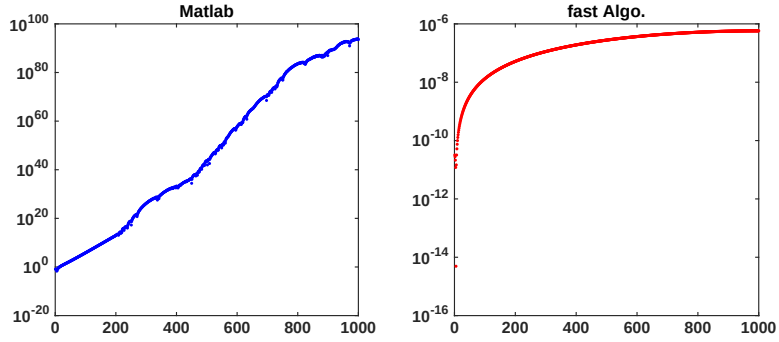

Figure 6: (Example 4.4) Component-wise relative-error for the approximated maximal eigenvector obtained by using sc MATLAB (left) and the revised method (right), for $N = 1000$.

5. Concluding remarks

In this work, we focus on numerical approximation of maximal eigenpair of tridiagonal matrices with positive super-/sub-diagonal elements. The starting algorithm is proposed by Chen in [4], which offers an accurate initial guess of the desired eigenpair. This efficient

Table 5: (Example 4.4) Numerical performance of rescaled algorithm with similarity transformation.

N	10^3	10^4	10^5
$\lambda_{\max}$	6.828413194902865	6.828426985196819	6.828427123350446
z_0	6.828414636367414	6.828426999637519	6.828427123494879
z_1	6.828413217756257	6.828426985425766	
z_2	6.828413194907978		

initial guess is equipped with a modified inverse iteration method. A more effective self-closed iterative algorithm without using the traditional inverse algorithms was proposed by Chen [5]. Note that both [4, 5] are proposed for medium size matrices, some efforts handling stability and complexity have to be made in order to solve large size matrix problems. In this work, we reduce the computational complexity to $\mathcal{O}(N)$ operations, and we also introduce two rescaling variables to overcome the overflow/underflow problem. Moreover, a diagonal similarity transformation is used to make the computation for non-symmetric matrices much more effective.

It is pointed out that due to the use of the self-closed iterative algorithm of Chen [5], the total number of iterations used in our computation is found less than 10. By testing 1000 randomly generated tridiagonal matrices, about 5–8 iterations are sufficient to make the relative errors below 10^{-10}.

We close this work by noting a very recent work of Chen [6] which applying [5] to block tridiagonal and Toeplitz matrix cases. It is natural to extend the present work to more general cases including these applications.

Acknowledgments The authors thank Prof. Mufa Chen of Beijing Normal University, Prof. Junfeng Yin of Tongji University, Prof. Michael Ng of Hong Kong Baptist University and Prof. Falai Chen of University of Science and Technology of China, for many helpful discussions. This work is partially supported by the Special Project on High-Performance Computing of the National Key R&D Program under No. 2016YFB0200604, the National Natural Science Foundation of China under No. 11731006 and the Science Challenge Project under No. TZ2018001.

References

[1] P. Arbenz, *Lecture notes on solving large scale eigenvalue problems*, Computer Science Department, ETH, 2016.

[2] W. Barth, R. S. Martin and J. H. Wilkinson, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, Numer. Math., 9 (1967), pp. 386–393.

[3] M. F. Chen, Eigenvalues, Inequalities, and Ergodic Theory, Springer, 2005.

[4] M. F. Chen, *Efficient initials for computing maximal eigenpair*, Front. Math. China, 11 (2016), pp. 1397–1418 .

[5] M. F. Chen, *Global algorithms for maximal eigenpairs*, Front. Math. China, 12(5) (2017), pp. 1023–1043.

[6] M. F. Chen, *Trilogy on computing maximal eigenpair*, International Conference on Queueing Theory and Network Applications, Springer, Cham, 2017, pp. 312–329.
[7] E. S. Coakley and V. Rokhlin, *A fast divide-and-conquer algorithm for computing the spectra of real symmetric tridiagonal matrices*, Appl. Comput. Harmon. A., 34 (2012), pp. 379–414.
[8] I. Dumitriu and A. Edelman, *Matrix models for beta ensembles*, J. Math. Phys., 43 (002), pp. 5830-5847.
[9] A. Edelman and P. O. Persson, *Numerical methods for eigenvalue distributions of random matrix*, https://arxiv.org/abs/math-ph/0501068, 2005.
[10] A. Edelman and N. R. Rao, *Random matrix theory*, Acta Numerica, 2005, pp. 1–65
[11] D. Gill and E. Tadmor, *An $\mathcal{O}(N^2)$ method for computing the eigensystem of $N \times N$ symmetric tridiagonal matrices by the divide and conquer approach*, SIAM J. Sci. Stat. Comp., 11 (1990), pp. 161–173.
[12] G. D. Smith, Numerical Solution of Partial Differential Equations, 2nd Ed., Clarendon Press, Oxford, 1978.
[13] G. H. Golub and J. H. Welsch, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–239.
[14] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.
[15] I. C. F. Ipsen and E. R. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, SIAM J. Sci. Stat. Comput., 11 (2) (1990), pp. 203–229.
[16] Y. Saad, *Numerical methods for large eigenvalue problems*, SIAM, Philadelphia, PA, 2011.
[17] D. B. Szyld, *Criteria for combining inverse and Rayleigh quotient iteration*, SIAM J. Numer. Anal., 25 (1988), pp. 1369–1375.