mathematics

MDPI

Article

A Novel Divisional Bisection Method for the Symmetric Tridiagonal Eigenvalue Problem

Wei Chu [1], Yao Zhao [1,2] and Hua Yuan [1,2,*]

[1] School of Naval Architecture and Ocean Engineering, Huazhong University of Sciences and Technology, Wuhan 430074, China
[2] Hubei Key Laboratory of Naval Architecture and Ocean Engineering Hydrodynamics (HUST), Wuhan 430074, China
* Correspondence: yuanhua@hust.edu.cn; Tel.: +86-027-8754-3258

Abstract: The embarrassingly parallel nature of the Bisection Algorithm makes it easy and efficient to program on a parallel computer, but with an expensive time cost when all symmetric tridiagonal eigenvalues are wanted. In addition, few methods can calculate a single eigenvalue in parallel for now, especially in a specific order. This paper solves the issue with a new approach that can parallelize the Bisection iteration. Some pseudocodes and numerical results are presented. It shows our algorithm reduces the time cost by more than 35–70% compared to the Bisection algorithm while maintaining its accuracy and flexibility.

Keywords: symmetric tridiagonal matrix; eigenvalue solver; matrix division; parallel algorithm

MSC: 65F15

Citation: Chu, W.; Zhao, Y.; Yuan, W. A Novel Divisional Bisection Method for the Symmetric Tridiagonal Eigenvalue Problem. *Mathematics* **2022**, *10*, 2782. https://doi.org/10.3390/math10152782

Academic Editors: Fajie Wang and Ji Lin

Received: 10 July 2022
Accepted: 4 August 2022
Published: 5 August 2022

1. Introduction

The symmetric tridiagonal matrices often arise as primary data in many computational quantum physical [1,2], mathematical [3–5], dynamical [6,7], computational quantum chemical [8,9], signal processing [10], or even medical [11] problems and hence are important. The current software reduces the generalized and the standard symmetric eigenproblems to a symmetric tridiagonal eigenproblem as a common practice [10,12,13]. What is more interesting is that the opposite path is also productive. Marques [14] computes the SVD of a bidiagonal matrix through the eigenpairs of an associated symmetric tridiagonal matrix. In this paper, we focus on symmetric eigenvalue solving.

People desire a parallel algorithm of good performance and flexibility, especially today as CPU cores and massively parallel technology have skyrocketed. We noticed that in many application scenes of eigenvalue computation, for example, in dynamics, it is often necessary to solve only the first few orders of eigenvalues of a large matrix. The desire for the largest eigenvalue is also common in practice [15–17]. However, the current QR, MRRR (Multiple Relatively Robust Representations), DC (Divided and Conquer), and Bisection algorithms do not seem to perform sufficient parallel operations if the number of CPU cores (say, 40) is significantly larger than the number of eigenvalues (say, 1) to be solved.

The most popular algorithm at present for a symmetric eigenproblem is the QR algorithm because of its stability and computational efficiency [18–20]. When only eigenvalues are desired, all square roots can be eliminated in the QR transformation. This was first observed by Ortega and Kaiser in 1963 [21] and a fast, stable algorithm was developed by Pal, Walker, and Kahan (PWK) in 1969 [22]. However, the parallelization of the QR algorithm is a problem, in this case, requiring more than a straightforward transcription of serial code to parallel code. Many researchers have made attempts, such as blocking the given matrix [23], look-ahead strategies [24], load-balancing schemes [25], pipelining of iterations [20,26], or dimensional analysis [27]. However, few seem adequate for the

symmetric tridiagonal matrices because most of those attempts are for dense matrices. One more essential trouble is that the QR algorithm is unsuitable for computing one or several selected eigenvalues. The MRRR algorithm [28] has a similar disadvantage as it is based on the DQDS algorithm [29,30] to compute the eigenvalues. In detail, both QR and DQDS algorithms use a designed shift, for example, Wilkinson's shift, to obtain a high-order asymptotic convergence rate. As a consequence, the order of eigenvalue convergence is not manageable.

The DC algorithm [31] is easily parallelizable and has developed well in recent years [32,33]. However, efficient parallel implementations are not straightforward to program, and the decision to switch from task to data parallelism depends on the characteristics of the underlying machine. Its space complexity is also an obvious shortcoming. In fact, even the "dstedc" routine corresponding to the DC algorithm in LAPACK calls "dsterf" when only eigenvalues are computed, i.e., the PWK version of the QR algorithm. The DC algorithm also does not support the computation of eigenvalues of a specific order or within a particular interval, let alone parallelization.

The Bisection method [34] calculates eigenvalues in any order or interval with a variable precision, which is suitable and handy for the mixed precision calculation [35]. Its embarrassingly parallel nature and high accuracy make it implemented in current software libraries for distributed memory computers. In addition, the Bisection method has a parallelizing efficiency of 1 (unless the number of computational cores is larger than the matrix dimension, which is rare) and little communication cost, which makes it highly advantageous in massively parallel computations. However, parallel Bisection can only be implemented if the number of unsolved eigenvalues is no less than the number of CPU cores. In addition, the computational efficiency of the Bisection method disconcerts.

We briefly summarize here: QR, DC, and MRRR algorithms are only available for obtaining all the eigenvalues. The Bisection method has excellent accuracy and flexibility but with limited efficiency when computing all the eigenvalues. All existing methods fail to calculate a single eigenvalue in parallel. Therefore, this paper has two goals: (1) to give a new Bisection method that can perform parallel operations with any number of threads when computing one specific eigenvalue; (2) to improve the efficiency of the Bisection method when calculating a major set of or all eigenvalues.

Section 2 presents some theorems, lemmas, corollaries, and equations. They are demonstrated for the design of Algorithms 4 and 5 and the accuracy analyses in Section 5. The big view of our method for one specific eigenvalue is dividing the matrix for parallel computing and merging them for the final result, with an insignificant time cost in the merging process. For the Bisection method to retain its ability to compute eigenvalues of any order, our strategy is to make the underlying iteration loop parallelizable. Instead of counting Sturm sequences iteratively, Algorithm 4 (provided in Section 3) distributes the task into the submatrices, which can be fulfilled independently. To merge these submatrices, in Section 2, we give a special determinant Formula (2) (with our new proof inspired by Maxwell's reciprocity theorem), Corollary 1, and Theorem 3.

We give Algorithm 5 in Section 4 as a modified Bisection method for all the eigenvalues. To reduce the number of iterations, the key is called a faster root-finder, which has less than 20% time cost of the traditional Bisection iteration process. However, it can only work when an isolating interval, i.e., an interval within only one eigenvalue, is obtained. Theorem 3 provides an excellent approach to such an interval, and the calculation is executed by dividing and merging. To accelerate convergence, we prove Theorem 4 in Section 4 and utilize the deflation property in Algorithm 5.

In Section 5, we analyze the accuracy and present the numerical experiments. Section 5.2 shows the accuracy results and Section 5.3 shows the efficiency result. In Section 5.3, diversified computing tasks are discussed and the feasibility is analyzed. The results show that the new Divisional Bisection method can substantially improve the efficiency of the Bisection algorithm while maintaining its accuracy and flexibility.

2. Dividing the Matrix

The sequential principal minors of an ST (Symmetric Tridiagonal) matrix form a Sturm Chain, which is the key to the Bisection algorithm. We denote the ith sequential principal minor of a matrix A by $A_{1:i}$, which is similar to the conventions in Matlab. The submatrix of A in rows i through j will be denoted by $A_{i:j}$; A is determinant by $det(A)$. We denote the characteristic polynomial $det(A - uI)$ by $C_{1:n}$, $C_{1:n}(u)$, or $C^A_{1:n}(u)$ if necessary.

Let A be an $n \times n$ unreduced ST matrix (all ST matrices discussed in this paper are unreduced), λ_i be its ith eigenvalue, v_i be its ith eigenvector and v_{ij} be the jth component of v_i. Then, we have the iterative formulae of the ST determinants from [34] as

$$\begin{array}{c} q_0 = 1, q_1 = a_1 - u, q_i = a_i - u - b_{i-1}^2/q_{i-1}, \\ p_0 = 1, p_1 = a_n - u, p_i = a_{n+1-i} - u - b_{n+1-i}^2/p_{i-1}, \end{array} \tag{1}$$

where $q_i = C_i/C_{i-1}$ and $p_i = C_{n-i+1}/C_{n-i+2}$.

The Bisection method counts Sturm sequences by q or p. The number of eigenvalues that are less than u is equal to the number of negative q values, while the number of $\lambda_i > u$ is equal to the non-negative q's. The neighboring C_i and q_i have the following theorem from [12].

Theorem 1 (Root Separation Theorem)**.**
C_i has only simple roots, which are separated strictly by the roots of C_{i-1}, for $i = 2, \ldots, n$.

From Theorem 1, we have the following corollary.

Corollary 1. *The signs of C_{i-1} and C_i in the intervals separated by their roots can be expressed as*

$$\begin{array}{c} +s_1 - s_2 + s_3 - \ldots \\ +\lambda_1 - \lambda_2 + \lambda_3 - \lambda_4 + \ldots \end{array}$$

where $s_k (k = 1, \ldots, i-1)$ denotes the kth root of $C_i - 1$ and $\lambda_k (k = 1, \ldots, i)$ denotes the kth root of C_i.

Proof. As $C(u) = \prod_{i=1}^{n}(\lambda_i - u)$, we have

$$Sign(C(u)) = \begin{cases} 1, & u \to -\infty \\ (-1)^n. & u \to +\infty \end{cases}$$

Considering that C_i has only simple roots (Theorem 1), the result shows. □

We stress Theorem 1 and Corollary 1 here because they are not only the basis for the following Theorems 2 and 3 but also support our subsequent algorithms and analyses. When merging the submatrices, we use Corollary 1 and the signs of C_i values to decide the global ζ in Algorithm 4. The accuracy of original iterations in Algorithm 5 is analyzed through Theorem 1 and Corollary 1, which guarantee that the original results can be checked and fixed with an acceptable iteration number (this process is carried by Algorithm 7). See more details in Sections 3 and 5.

Recall that our task is to count Sturm sequences in submatrices; then, it is convenient to calculate q values and p values from both ends of A. A specific determinant formula shows the connection between $det(A)$ and $det(A_{1:k})$ and $det(A_{k+1:n})$ or q_i and p_i, which is from [36]. Here, we present a new proof inspired by Maxwell's reciprocity theorem.

According to Maxwell's reciprocity theorem, the output at j caused by input at any point i in a linear system is equal to the output at i caused by equal input at j. If we consider the ST matrix A to be a dynamical system, the following lemma holds.

Lemma 1. *For an invertible symmetry matrix A, if $Ax = e_i$ and $Ay = e_j$ then $x_j = y_i$, where x and y are both column vectors.*

Proof. It can be easily established by symmetry. □

Theorem 2 (Determinant Formula)**.**
Let a be the diagonal of an unreduced ST matrix A and b be the sub-diagonal, we have

$$\begin{aligned} C_{1:n} &= det(A - uI) \\ &= -b_{k-1}^2 C_{1:k-2} C_{k+1:n} + (a_k - u) C_{1:k-1} C_{k+1:n} - b_k^2 C_{1:k-1} C_{k+2:n} \\ &= C_{1:k-1} C_{k+1:n} (C_{1:k}/C_{1:k-1} - b_k^2 C_{k+2:n}/C_{k+1:n}) \\ &= C_{1:k-1} C_{k+1:n} (C_{k:n}/C_{k+1:n} - b_{k-1}^2 C_{1:k-2}/C_{1:k-1}). \end{aligned} \tag{2}$$

Proof. Let:

$$\begin{aligned} x &= [1, C_{1:1}/-b_1, \ldots, C_{1:n-1}/(\prod_{t=1}^{n-1} -b_t)]^T; \\ y &= [C_{2:n}/(\prod_{t=1}^{n-1} -b_t), \ldots, C_{n:n}/-b_{n-1}, 1]^T, \end{aligned} \tag{3}$$

substitute them into (1), then we have

$$\begin{aligned} (A - uI)x &= [0, \ldots, 0, F_1]^T; \\ (A - uI)y &= [F_1, 0, \ldots, 0]^T; \\ F_1 &= C_{1:n}/\textstyle\prod_{i=1}^{n-1}(-b_i) \end{aligned} \tag{4}$$

when uniting (1) and (3).

Construct a vector z so that

$$\begin{aligned} z_{1:k} &= x_{1:k}; \\ z_{k:n} &= \eta \times y_{k:n}; \\ (A - uI)z &= [0, \ldots, F_2, \ldots, 0]^T, \end{aligned} \tag{5}$$

where η is a nonzero scalar.

As $z_k = x_k$, we have

$$\eta = \frac{C_{1:k-1}/(\prod_{t=1}^{k-1} -b_t)}{C_{k+1:n}/(\prod_{t=k}^{n-1} -b_t)}.$$

According to Lemma 1,

$$\frac{x_k}{F_1} = \frac{z_n}{F_2}. \tag{6}$$

Unite (4)–(6); then, the result shows. □

Remark 1. *(2) can also be expressed as*

$$C_{1:n} = C_{1:k-1} C_{k+1:n} (q_k - b_k^2/p_{n-k}).$$

In addition, although u should not be an eigenvalue of A in Lemma 1, (2) also holds for all λ_i valuesof A. To prove this, we need to check the existence of x and y first, as $A - \lambda_i I$ is a singular matrix. We have $F_1 = 0$ in (4), which means x and y are both eigenvectors. Consider the eigenvectors-from-eigenvalues formula (see [37])

$$v_{ij}^2 \prod_{k=1;k\neq i}^{n} (\lambda_i - \lambda_k) = \prod_{k=1}^{n-1} (\lambda_i - \lambda_k(A_{\ominus j})), \tag{7}$$

where $A_{\ominus j}$ denotes the $n-1 \times n-1$ minor formed from A by deleting the jth row and column of A. As A is symmetric and tridiagonal, (7) can be expressed as

$$v_{ij}^2 \prod_{k=1;k\neq i}^{n} (\lambda_k - \lambda_i) = C_{1:j-1}(\lambda_i) C_{j+1:n}(\lambda_i). \tag{8}$$

Let $i = n$, from (8) we have

$$v_{nj}^2 \prod_{k=1}^{n-1} (\lambda_k - \lambda_n) = C_{1:n-1}(\lambda_n).$$

Consider Theorem 1; then, it shows that the eigenvector of an ST matrix has no zero components at both ends. So, existence is guaranteed. Then, the result can be easily verified by the continuous prolongation theorem.

Remark 2. *The determinant formula is introduced in [36] (page 518, Equation (5)), which gives a form of a general tridiagonal matrix, not having to be symmetric. (2) is the specific form for symmetry. Nevertheless, we insist on presenting this different proof here because some intermediate products of the derivation process consist of the basis of Theorem 4, which is one key technology to accelerate Algorithm 5. See more details in Section 4.*

Theorem 3 (Interlacing Property). *If $C_{1:k-1}$ and $C_{k:n}$ do not have a common root, the roots of $C_{1:k-1}C_{k:n}$ (i.e., the eigenvalues of $A_{\ominus k}$) separate the eigenvalues of A strictly; if not, the common roots are some eigenvalues of A and the others still separate strictly. In addition, Corollary 1 also holds for $C_{1:k-1}C_{k:n}$ and $C_{1:n}$.*

Proof. According to [12,38], we have

$$\lambda_1 \leq s_1 \leq \lambda_2 \leq s_2 \leq \cdots \leq s_{n-1} \leq \lambda_n \tag{9}$$

where $s_i (i = 1, \ldots, n-1)$ denotes the ith eigenvalue of $A_{\ominus k}$.

If $C_{1:k-1}$ and $C_{k:n}$ have a common root, it can be easily seen from (2) that $C_{1:n} = 0$; if not, we have $C_{1:n} \neq 0$ similarly.

So, the equal signs hold if and only if $C_{1:k-1}$ and $C_{k:n}$ have a common root. □

With Theorem 2 and 3, we now divide the unreduced ST matrix A into $A_{1:k-1}$ and $A_{k+1:n}$, and we count the negative Sturm sequences of a tentative eigenvalue u independently. In $A_{1:k-1}$, ζ_1 is the number of negative q_i values ($i = 1, \ldots, k-1$) and ζ_2 is the negative p_i values ($i = 1, \ldots, n-k$) in $A_{k+1:n}$. Let $\zeta = \zeta_1 + \zeta_2$; apparently, it is equal to the number of eigenvalues of $A_{\ominus k}$ that are less than u. Thus, the sign of $C_{1:k-1}C_{k:n}$ is $(-1)^{\zeta}$. According to Theorem 3, this also means $u \in (\lambda_\zeta, \lambda_{\zeta+2})$. Theorem 2 shows the connection between the sign of $C_{1:k-1}C_{k:n}$ and the sign of $C_{1:n}$. Thus, the final ζ, which is either equal to the previous $\zeta_1 + \zeta_2$ or $\zeta_1 + \zeta_2 + 1$, can be concluded with a cheap merging calculation. See more details in the next section.

3. Computing One ST Eigenvalue

We now consider more details of the Divisional Bisection method. First, we introduce Algorithm 1 for computing q_i, ζ and $C_{1:n}$ in an unreduced $n \times n$ ST matrix A according to [34], and the simplified variant Algorithm 2, for the determinant only.

Algorithm 1: Bisection Iteration

Input : a, b^2, n
// a is the diagonal of A, b is the sub-diagonal and n is the size
Output: $\zeta, q, C_{1:n}$
// $q = q_{1:n}$
$q \leftarrow a_1, C_{1:n} \leftarrow 1$;
if $q < 0$ **then**
  $\zeta = 1$;
else
  $\zeta = 0$;
end
for *each* $k \in [1 : n]$ **do**
  if $q == 0$ **then**
    $q \leftarrow \varepsilon$ // ε is a positive small value
  end
  $q \leftarrow a_k - b_{k-1}^2/q$;
  $C_{1:n} \leftarrow qC_{1:n}$;
  if $q < 0$ **then**
    $\zeta \leftarrow \zeta + 1$;
  end
end

Algorithm 2: Computing ST Determinant

Input : a, b^2, n
Output: $C_{1:n-1}, C_{1:n}$
$q \leftarrow a_1, C_{1:n-1} \leftarrow 1$;
for *each* $k \in [1 : n-1]$ **do**
  if $q == 0$ **then**
    $q \leftarrow \varepsilon$ // ε is a positive small value
  end
  $q \leftarrow a_k - b_{k-1}^2/q$;
  $C_{1:n-1} \leftarrow qC_{1:n-1}$;
end
$q \leftarrow a_n - b_{n-1}^2/q$;
$C_{1:n} \leftarrow qC_{1:n-1}$.

If $u \in (\lambda_\zeta, \lambda_{\zeta+2})$ as discussed in Section 2, we have

$$sign(C_{1:n}) = \begin{cases} (-1)^\zeta, & q_k > b_k^2/p_{n-k}; \\ (-1)^{\zeta+1}, & q_k < b_k^2/p_{n-k}; \\ 0, & q_k = b_k^2/p_{n-k}, \end{cases} \tag{10}$$

according to (2) and Corollary 1. Then, we have

$$\zeta = \begin{cases} \zeta, & q_k \geqslant b_k^2/p_{n-k}; \\ \zeta + 1, & q_k < b_k^2/p_{n-k}, \end{cases} \tag{11}$$

and $u = \lambda_{\zeta+1}$ when $q_k = b_k^2/p_{n-k}$. When $q_k p_{n-k} = 0$, which means (10) cannot be calculated, we directly obtain $\zeta = \zeta$ according to Theorem 3. Similarly, we have $u = \lambda_{\zeta+1}$ if q_k and p_{n-k} are both zeros.

In the lower level, $A_{1:k-1}$ is divided into $A_{1:t-1}$ and $A_{t+1:k-1}$. Independently, we calculate

1. $\zeta_{1:t-1}$, $q_t(A_{1:k-1})$, and $C_{1:t-1}$ in $A_{1:t-1}$ by Algorithm 1;
2. $\zeta_{t+1:k-1}$, $p_{k-t-1}(A_{1:k-1})$ and $C_{t+1:k-1}$ in $A_{t+1:k-1}$ by Algorithm 1;
3. $C_{t+2:k-2}$ and $C_{t+1:k-2}$ in $A_{t+1:k-2}$ by Algorithm 2.

And the same in $A_{k+1:n}$.
By substituting these outputs into (2), (10) and (11),

1. $\zeta_{1:k-1}$, $\zeta_{k+1:n}$;
2. $C_{1:k-1}$, q_k;
3. $C_{k+1:n}$, p_{n-k}.

These are determined, and then, we have $\zeta_{1:n}$ finally, completing one Bisection iteration. The new Divisional Bisection iteration method is given by Algorithm 3.

Algorithm 3: Divisional Bisection Iteration

Input : a, b^2, n, p
// u is a tentative eigenvalue, p is the number of dividing parts
Output: ζ
distribute a, b^2 into $m+w$ parts evenly such that $m+w=p$;
// so that each pair of a_i and b_i^2 forms a submatrix of A
then get $a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+w}$, $b_1^2,\ldots,b_m^2,b_{m+1}^2,\ldots,b_{m+w}^2$;
foreach $i\in[2:m]\&\&i=m+w$ **do**
    reverse a_i, b_i^2;
end
foreach $i\in[1:m+w]$ **do**
    call Algorithm 1 $\Leftarrow a_i$,b_i^2,N_i // N_i is the length of a_i
    then get ζ_i, q_i, C_i;
end
foreach $i\in[2:m]\cap[m+1:m+w-1]$ **do**
    call Algorithm 2 $\Leftarrow a_i(2:end)$,$b_i^2(2:end)$,N_i-1;
    // eliminate 1st component
    then get $C_{2:N_i}$, $C_{2:N_i-1}$;
end
$s\leftarrow 0$, $C_l\leftarrow C_1$, $q_l\leftarrow q_1$, $i\leftarrow 2$;
while $i\le m$ **do**
    substitute C_l, q_l, C_i, q_i, $C_{2:N_i}$, $C_{2:N_i-1}$ into (2);
    then get C_l, q_l;
    // C_l, q_l is substituted by those of the merged matrix
    $s\leftarrow s$ or $s\leftarrow s+1$ according to (10) and (11), $i\leftarrow i+1$;
end
$C_r\leftarrow C_{m+w}$, $q_r\leftarrow q_{m+w}$, $i\leftarrow w+m-1$;
while $i\ge m+1$ **do**
    substitute C_r, q_r, C_i, q_i, $C_{2:N_i}$, $C_{2:N_i-1}$ into (2);
    then get C_r, q_r;
    // C_r, q_r is substituted by those of the merged matrix
    $s\leftarrow s$ or $s\leftarrow s+1$ according to (10) and (11), $i\leftarrow i-1$;
end
substitute C_l, q_l, C_r, q_r into (2);
$s\leftarrow s$ or $s\leftarrow s+1$ according to (10) and (11);
$\zeta\leftarrow s+\sum\zeta_i$.

Algorithm 3 calls Algorithm 2 to compute $p-2$ extra determinants of the submatrices compared to the traditional method. So, the parallel efficiency of Algorithm 3 is $p/(2p-2)$, given that the cost of the merging part is negligible compared to the cost of Algorithms 1 and 2 called during computation. It should be noted that counting non-negative q values

instead is more efficient if a high-order eigenvalue is desired. By replacing the iterative process, we give the new Divisional Bisection Algorithm 4 for computing one ST eigenvalue.

Algorithm 4: Computing One ST Eigenvalue

Input : a, b, n, p, r, tol
// compute the rth eigenvalue with the expected precision tol
Output: λ_r
set the original interval $[x, y]$, $b \leftarrow b^2$;
// $x, y = \mp||A||_\infty$, for example
while $|y - x| \geq 2tol$ **do**
  $u \leftarrow (y - x)/2$;
  call Algorithm 3 $\Leftarrow a - u, b, n, p$
  if *any* $\zeta_i \geq r$ *when executing Algorithm 3* **then**
    stop Algorithm 3;
    $\zeta \leftarrow r$;
  else
    complete Algorithm 3;
    then get ζ
  end
  if $\zeta \geq r$ **then**
    $y \leftarrow u$;
  else
    $x \leftarrow u$;
  end
end
$\lambda_r \leftarrow (y - x)/2$.

In addition, it can be predicted that a considerable number of Divisional Bisection iterations will end early, especially for the lower or higher order eigenvalues. To find the smallest eigenvalue of a matrix, for example, we can break the iteration in advance if any $\zeta_i \geq 1$, which means the final number will inevitably exceed 1 according to Theorem 3. This strategy can save substantial time in the early computation and more if a larger p is available.

4. Computing All ST Eigenvalues

The Bisection algorithm has many practical advantages but earns the disrepute of being slow when computing all ST eigenvalues. A significant contributor is the excessive number of iterations. The Bisection algorithm permits an eigenvalue to be computed with 53 iterations in IEEE double-precision arithmetic. When an eigenvalue is isolated in an interval, we have some faster root-finders such as Laguerre's method [12,39], the Zeroin scheme [40,41] and the fzero scheme [42] ('fzero' function in Matlab). These competitors can finish the work in less than 10 iterations but seem to stumble when eigenvalues cluster. Another trouble is that so much more has to be completed in the inner loop [39,43] to obtain isolating intervals, costing embarrassingly more time.

Our strategy is to obtain isolating intervals by the eigenvalues of $A_{\ominus k}$. These eigenvalues can be obtained by QR or a Bisection algorithm on each submatrix. The clustering eigenvalues, which can be challenging problems otherwise, accelerate the calculation in our method according to Theorem 3. The submatrix under continuing division (if necessary) has no eigenvalues clustered eventually. Then, we can compute all the eigenvalues by dividing and merging. For convenience, we choose the 'fzero' function in Matlab as the root-finder, which requires an average of 7.5 iterations per root. Our numerical experience supports this conclusion.

It has been found in [31,38] that the deflation properties and techniques of the DC algorithm allow it to converge quickly when the eigenvalues of submatrices cluster or the eigenvectors have zero ends in finite precision arithmetic. These deflation cases are quite

common in ST matrices and should be utilized in the Divisional Bisection algorithm. Let *tol* be the expected precision and $s_i (i = 1, \ldots, n-1)$ be the eigenvalues of $A_{\ominus k+1}$, which can be divided into T_1 and T_2. From [38] we have

$$\begin{aligned} A &= QDQ^T \\ &= \begin{bmatrix} 0 & Q_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & Q_2 \end{bmatrix} \begin{bmatrix} a_{k+1} & b_k l_k^T & b_{k+1} r_1^T \\ b_k l_k & D_1 & 0 \\ b_{k+1} r_1 & 0 & D_2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ Q_1^T & 0 & 0 \\ 0 & 0 & Q_2^T \end{bmatrix} \end{aligned} \tag{12}$$

where

- $T_1 = Q_1 D_1 Q_1^T$ and $T_2 = Q_2 D_2 Q_2^T$ are the eigendecomposition of T_1 and T_2;
- l_k^T is the last row of Q_1;
- r_1^T is the first row of Q_2;
- the diagonals of D_1 and D_2 are arranged in ascending order.

Now, consider how deflation occurs during the calculation and how our algorithm can perceive it. In (12), the close eigenvalues of D_1 and D_2 can be easily detected, since we do the calculation by dividing and merging. However, the connection between zero ends of $b_k l_k$ or $b_{k+1} r_1$ and the intermediate results of Bisection iterations are not easily accessible. Therefore, we give Theorem 4, especially Theorem 4b, to show the deflation properties and to suggest an approach to detecting. First, we introduce the following Lemma 2 as an auxiliary for our proof of Theorem 4.

Lemma 2. *Let A_1 and A_2 be $n \times n$ real symmetric matrices with eigenvalues $\lambda_1^{A_1}, \ldots, \lambda_n^{A_1}$ and $\lambda_1^{A_2}, \ldots, \lambda_n^{A_2}$, respectively. Then*

$$\max_i |\lambda_i^{A_1} - \lambda_i^{A_2}| \leq \|A_1 - A_2\|_2.$$

Proof. See [44]. □

Theorem 4 (Deflation Properties)**.**

a. *If $|\bar{s}_{i+1} - \bar{s}_i| \leq tol$ where $\bar{s}_i$ and $\bar{s}$ are arithmetic approximations of s_i and s_{i+1}, then $\bar{s}_i$ or $\bar{s}_{i+1}$ is an arithmetic approximation of λ_{i+1};*

b. *Let u be an arithmetic approximation to s_i which is one of the $s_j^{T_1}$'s and $s_i = s_h^{T_1} (h \in [1, k])$. If*

(1) $(C_{1\to k-1}^{T_1}(u) / C_{1\to k}^{T_1}(u))(s_i - u) < 0$;

(2) $|b_k| \sqrt{\left(1/g - \left|C_{1\to k-1}^{T_1}(u) / C_{1\to k}^{T_1}(u)\right|\right)} < \sqrt{tol}$,

where $g = \min_{j \neq t} |s_j^{T_1} - u|$, then u is an arithmetic approximate eigenvalue of A, and the similar holds in T_2.

Proof.

a. It can be easily seen from Theorem 3.

b. Without loss of generality, we assume s_i is an isolated eigenvalue of $A_{\ominus k+1}$ because if not, we can turn to Theorem 4a.

From (3) and (4), it shows $1/q_k^{T_1}(u) = (C_{1\to k-1}^{T_1}(u) / C_{1\to k}^{T_1}(u))$ is the last component on the diagonal of $(T_1 - uI)^{-1}$. Then, we have

$$\begin{aligned} & 1/q_k^{T_1}(u) = e_k^T (T_1 - uI)^{-1} e_k, \\ \Rightarrow\ & 1/q_k^{T_1}(u) = e_k^T Q (D_1 - uI)^{-1} Q^T e_k, \\ \Rightarrow\ & 1/q_k^{T_1}(u) = \sum_{j=1}^{k} v_{jk}^2 \frac{1}{s_j^{T_1} - u} \end{aligned} \tag{13}$$

where v_j is the jth eigenvector of T_1.

As $C_{1\to k}^{T_1}(u)$ is the determinant of $T_1 - uI$, $q_k^{T_1}$ should be close to zero when $u \to s_i$. However, in IEEE double precision arithmetic, this is not true if v_{ik}^2 is also small when compared to $s_i - u$. (13) can be expressed as

$$1/q_k^{T_1}(u) = \frac{v_{ik}^2}{s_i - u} + \sum_{j=1\neq i}^{k} v_{jk}^2 \frac{1}{s_j - u} = v_{ik}^2/(s_i - u) + R_i, \tag{14}$$

where apparently (recall that $g = \min_{j\neq t} |s_j^{T_1} - u|$)

$$|R_i| \in \left[0, \frac{1}{g}\right). \tag{15}$$

Given that u is the previous computation result, we have $|s_i - u| \leq tol$. When $q_k^{T_1}(u)(s_i - u) > 0$, (14) and (15) can be united as

$$\begin{aligned} &\left|v_{ik}^2/(s_i - u)\right| < 1/g + \left|1/q_k^{T_1}\right|, \\ &\Rightarrow |v_{ik}| < \sqrt{(1/q_k^{T_1} + 1/g)tol}. \end{aligned} \tag{16}$$

In addition, we have

$$|v_{ik}| < \sqrt{(1/q_k^{T_1} - 1/g)tol} \tag{17}$$

similarly when $q_k^{T_1}(u)(s_i - u) < 0$.

The condition of Theorem 4b shows $|b_k v_{ik}| < tol$ according to (17). By taking a review of (12) and Lemma 2, the proof is completed. □

Theorem 4 is satisfying because q_i values of T_1 and p_i values of T_2 happen to be accompanying products of Algorithm 2, which can be utilized as the basic iteration of the 'fzero' scheme. The condition of Theorem 4b is sufficient but not necessary, as there are many other possibilities that make $|v_{ik}| < tol$, even when $(C_{1\to k-1}^{T_1}(u)/C_{1\to k}^{T_1}(u))(s_i - u) \geq 0$. A trivial plan is to calculate and check v_{ik} once one s_i is solved and the accompanying $|1/q_i^{T_1}|$ is suspiciously small. Although this idea already saves a large number of unnecessary computations compared to the DC algorithm, we are still concerned that it is too expensive to call the Inverse Iteration algorithm here.

Our scheme is to mark those suspicious small $|1/q_i^{T_1}|$ values by a rough discriminant, for example $|1/q_i^{T_1}| < 1$, then to substitute the corresponding $\bar{s}_i \pm tol$ values into Algorithm 1 to check if deflation is available. We have found in our numerical experiments that it is difficult to cover all the deflation situations by this method, even if we set the discriminant quite loosely. Even filtrating directly by $|v_{ik}|$, as in the DC algorithm, would still leave some out. We applied these methods to 20 randomly generated 2001×2001 matrices for computation, where T_1 and T_2 are both 1000×1000 matrices. The averages were calculated and are shown in Table 1. We collected the hit rate of the DC algorithm by checking how many $\bar{s}_i$ values, which had negligible corresponding v_{ik} values, were really close to λ_i values. In Table 1, the plan 1 refers to "rough discriminant + Inverse Iteration algorithm", the plan 2 refer to "rough discriminant + Algorithm 1", and the hit rates of them were collected similarly. It can be seen that the hit rate and accuracy of our method are acceptable or at least no worse than the DC algorithm. The errors in Table 1 refer to the difference between $\bar{s}_i$ values selected during deflations and $\bar{\lambda}_i$ values obtained by the Bisection method. The data were collected on an Intel Core i5-4590 3.3 GHz CPU and 16 GB RAM machine. All codes were written in Matlab2017b and executed in IEEE double precision. The machine precision is $eps \approx 2.2 \times 10^{-16}$.

Table 1. Comparison of deflation detecting methods (average of 20 2001 × 2001 matrices).

Methods	Time Cost (s)	Hit Rate	Average Error ($\times 10^{-16}$)	Maximum Error ($\times 10^{-16}$)
DC algorithm	/	58.5	1.39	4.44
plan 1	0.30	62.1	1.32	4.44
plan 2	0.19	62.1	0.91	1.00

We give the Divisional Bisection method for all eigenvalues by Algorithm 5 and the following subroutine Algorithm 6.

Algorithm 5: Computing all ST Eigenvalues

Input : a, b, n, p, tol
Output: $\vec{d}$

```
// all eigenvalues lie in the vector d⃗ in ascending order
```
distribute a, b into p parts evenly, $F \leftarrow \max(|a_i| + 2|b_i|)$, **calculate** b^2 in each part;
call PWK version of QR Algorithm in each part;
then get $\vec{d}_1, \ldots, \vec{d}_p$ `// eigenvalues of each submatrix lie on` $\vec{d}_i$
call Algorithm 2 $\Leftarrow$ each a_i, b_i^2, N_i;
then get q_i's correspondingly;
check deflation, form $\vec{t}_1, \ldots, \vec{t}_{p/2}$ by determined $\vec{d}_i(j)$'s, eliminate corresponding components in each $\vec{d}_i$;
if *there are clustering* $\vec{d}_i(j)$*'s* **then**
    call Algorithm 7 to recheck;
    `// Algorithm 7 is provided in Section 5`
end
while $p \geq 2$ **do**
    $m \leftarrow p/2$, $i \leftarrow 1$ $s \leftarrow 1$;
    while $i<p$ **do**
        $\vec{v}_s \leftarrow [-F; sort([\vec{d}_i; \vec{d}_{i+1}]); F]$;
        $s \leftarrow s+1$, $i \leftarrow i+2$;
    end
    call Algorithm 6 $\Leftarrow \vec{v}_1, \ldots, \vec{v}_m$;
    then get $\vec{d}_1, \ldots, \vec{d}_m$ and corresponding q_i's;
    combine each $\vec{d}_j$ and $\vec{t}_j$, $j \in [1:m]$;
    `// eigenvalues of each merged matrix lie on` $\vec{d}_i$
    check deflation, form $\vec{t}_1, \ldots, \vec{t}_m$ by determined $\vec{d}_i(j)$'s, eliminate corresponding components in each $\vec{d}_i$;
    $p \leftarrow p/2$;
end
combine $\vec{d}_1$ and $\vec{t}_1$, $\vec{d} \leftarrow \vec{d}_1$.

Algorithm 6: Fzero by Determinant

Input : a, b^2, n, V, tol

```
// searh one root in a isolating interval V
```
Output: x, q_n

call Algorithm 2 $\Leftarrow a, b^2, n$;
call 'fzero' function in Matlab $\Leftarrow$ Algorithm 2, V, tol;
then get x;
save q_n of the last iteration.

5. Accuracy Analysis and Numerical Results

5.1. Accuracy Analysis

After the eigenvalues of the original submatrices are calculated by the QR Algorithm, as shown by line 3 in Algorithm 5, it is not safe to take $(\bar{s}_i - \bar{s}_{i-1})/2$ as a λ_i if one $\bar{s}_i - \bar{s}_{i-1} \leq tol$, because the QR algorithm is not always as accurate as the Bisection method or fzero scheme. So, in practice, we do an extra check for the selected $\bar{s}_i$ values by Theorem 4a when checking deflation from results of the QR Algorithm. Suppose m sub-eigenvalues (denoted by $s_1, \ldots, s_m$) cluster in the interval $[x, y]$ where $y - x \leq tol$; the process is shown as Algorithm 7.

Algorithm 7: Recheck the Results of QR

Input : clustering sub-eigenvalues $s_1, \ldots, s_m$, interval $[x, y]$
Output: $\lambda_1, \ldots, \lambda_m - 1$

1 // the subscripts of λ's denote the order in this subroutine, not globally
2 Determine how many eigenvalues lie in $[x : y]$ by Algorithm 1 and save the number as w; **if** $w = m - 1$ **then**
3 **foreach** $i \in [1 : m - 1]$ **do**
4 $\lambda_i \leftarrow (s_i + s_{i+1})/2$;
5 **end**
6 **else**
7 **foreach** $i \in [1 : w - 1]$ **do**
8 $\lambda_i \leftarrow x + i * (y - x)/(w - 1)$;
9 **end**
10 **call** Bisection algorithm to search the remain $m - 1 - w$ λ's in $[x - 10tol, x) \cap (y, y + 10tol]$.
11 **end**

In Algorithm 7, $10tol$ is a pessimistic estimation of QR algorithm error, which means it decuples that of the Bisection error. The data in Table 2, which are present in a later paragraph, supports our point. Line 2 in Algorithm 7 costs 2 Bisection iterations for $w - 1$ λ values and line 10 costs 3 to 4 per λ compared to about 7.5 iterations per λ in Algorithm 6 and 53 iterations per λ in the Bisection algorithm.

When arithmetic approximations $\bar{s}_i$ are treated as the boundaries of isolating intervals in the next level, they do not affect the accuracy because if the number of λ's in an interval is not one, Algorithm 6 fails. The troublesome number could be 0 or 2, but it is certainly not bigger than 3. When there are 4 or more λ's in an interval, it means there are clustering $\bar{s}_i$'s of the previous results which can be perceived during the deflation check. For example, if 4 λ's lie in $[\bar{s}_j, \bar{s}_{j+1}]$ as

$$\bar{s}_j < \lambda_{j-1} < s_{j-1} < \lambda_j < s_j < \lambda_{j+1} < s_{j+1} < \lambda_{j+2} < \bar{s}_{j+1}, \tag{18}$$

we have $s_{j-1} - \bar{s}_j < \epsilon$ where ϵ is the previous computation error. (18) shows that $\bar{s}_{j-1}$ and $\bar{s}_j$ both lie in $(s_{j-1} - \epsilon, s_{j-1}]$, which could not happen because we do the deflation check previously.

We regard this as a beneficial situation. It can be seen in (18) that the troublesome number arises only when $\bar{s}_j < \lambda_j$ (or $\bar{s}_j > \lambda_{j+1}$), contrary to Theorem 3. As the accurate $s_j > \lambda_j$ and $s_j - \bar{s}_j \leq \epsilon$, we have $\lambda_j - \bar{s}_j \leq \epsilon$ and then can speed up the calculation. Finally, the accuracy of Theorem 5 is as good as the Bisection algorithm.

We checked the accuracy of Algorithm 5 by computing the eigenvalues of a 2001×2001 Toeplitz ST matrix, which has all 2's on its diagonal and all -1's on its sub-diagonal. The results of each method were then compared with the exact value, i.e., $\lambda_i = 2 - 2\cos(i\pi/2002)$, and are shown in Table 2. In addition, all eigenvalues of 20 randomly generated matrices

were calculated for testing the efficiency on serial machines, and we show the average results of 20 in Table 3. We set $p = 2$ in Algorithm 5 for the serial execution.

Table 2. Accuracy Result.

Method	Time Cost (s)	Average Error $\times eps$	Maximum Error $\times eps$
QR	0.10	4.2	32.0
PWK QR	0.09	3.9	32.0
MRRR	0.13	15.1	34.0
Bisection	1.55	1.0	6.0
Our method	0.41	1.0	6.0

Table 3. Time Cost Result.

Method	Time Cost (s) of		
	2500 × 2500 Matrix	5000 × 5000 Matrix	10,000 × 10,000 Matrix
QR	0.16	0.86	2.30
PWK QR	0.13	0.77	1.96
MRRR	0.17	0.92	2.55
Bisection	2.25	12.49	34.10
Our method	0.61	2.30	9.21

Table 2 demonstrates that our method substantially improves the speed of the Bisection method without losing accuracy. In addition, Table 3 confirms that Algorithm 5 is $O(n^2)$ as its iteration based on Algorithm 2. In the following subsections, we illustrate more test results of several different types of matrices. All results in Section 5 were collected on an Intel Core i5-4590 3.3-GHz CPU and 16-GB RAM machine, except for the last figure, which will be introduced in Section 5.4 specifically. All codes were written in Matlab2017b and executed in IEEE double precision. The machine precision is $eps \approx 2.2 \times 1-16$.

5.2. Matrices Introduction and Accuracy Test

In the following subsections, we present a numerical comparison among the Divisional Bisection algorithm and four other algorithms for solving the ST eigenvalue problem:

1. Bisection, by calling subroutine 'dstebz' from LAPACK in Matlab;
2. MRRR, by calling subroutine 'dstegr' from LAPACK in Matlab;
3. QR, by calling subroutine 'dsteqr' from LAPACK in Matlab;
4. PWK version of QR (which would be denoted by QR-pwk in the figures), by calling subroutine 'dsterf' from LAPACK in Matlab.

We use the following sets of test $n \times n$ matrices:

1. Matrix A:

$$\text{Matrix A} = \text{tridiagonal}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2 & \cdots & 2 \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

i.e., the Toeplitz matrix [1,2,1] to test the accuracy and efficiency, which has $\lambda_i = 2 - 2\cos(i\pi/(n+1))$;

2. Matrix T1:

$$\text{Matrix T1} = \text{tridiagonal}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

to test the accuracy and efficiency, which has $\lambda_i = -2\cos(2i\pi/(2n+1))$. Matrix T1 is from [45], as well as the following Matrix T2 and T3;

3. Matrix T2 [45]:

$$\text{Matrix T2} = \text{tridiagonal}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

to test the accuracy and efficiency, which has $\lambda_i = -2\cos(i\pi/n)$;

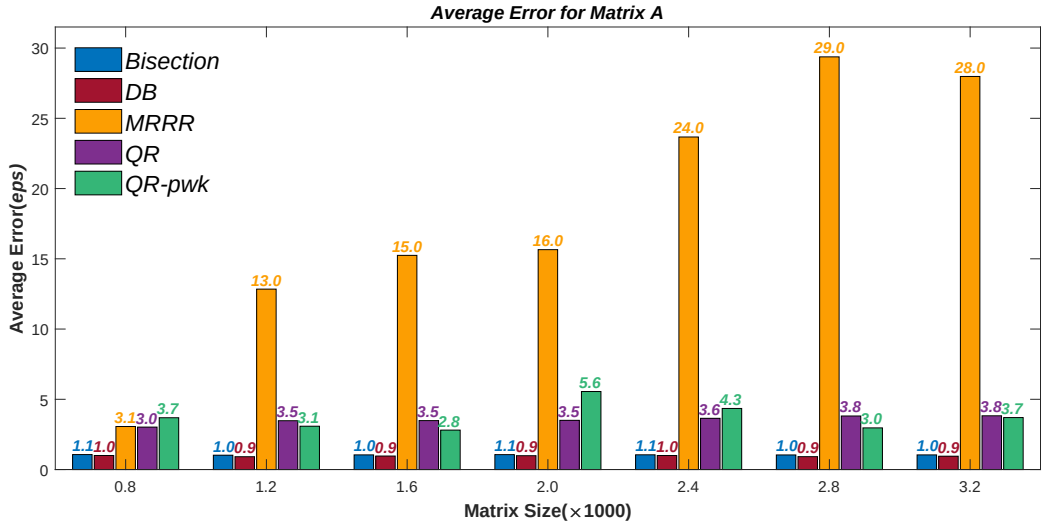
4. Matrix T3 [45]:

$$\text{Matrix T3} = \text{tridiagonal}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & -1 \\ 1 & 1 & \cdots & 1 \end{bmatrix},$$

to test the accuracy and efficiency, which has $\lambda_i = 2\cos((2i-1)\pi/(2n))$;

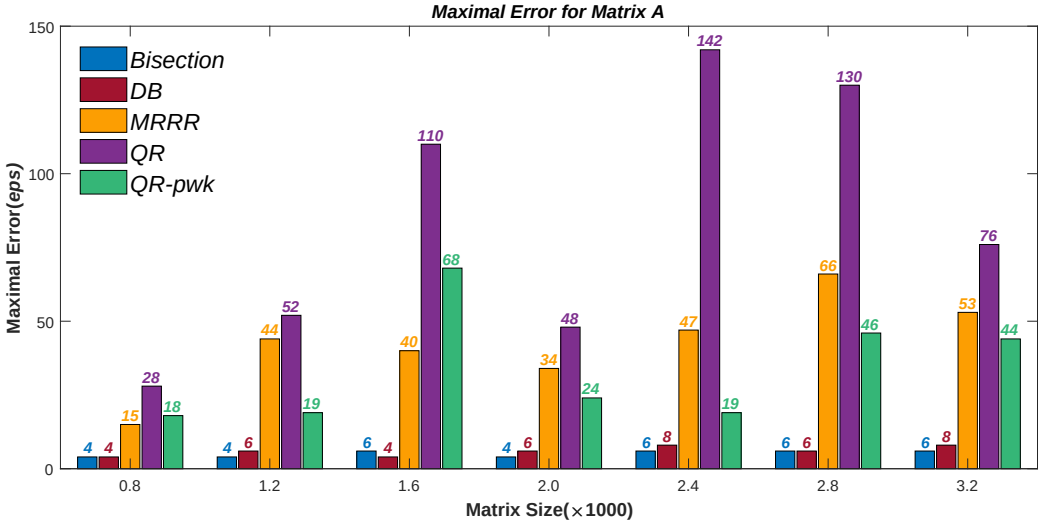
5. Matrix W [12,46], which has the ith diagonal component equal to $|(n+1)/2 - i|$(n is odd) and all off-diagonal components equal to 1, to test the efficiency only as its exact eigenvalues are not accessible;
6. Random Matrix with both diagonal and off-diagonal elements being uniformly distributed random numbers in $[-1,1]$ to test the efficiency only as its exact eigenvalues are not accessible.

Figures 1–4 present the test results of accuracy, where the Average Errors denote the means of errors of all the calculated eigenvalues and the Maximal Errors denote the maximum. Seven different sizes are used, from 800×800 to 3200×3200. All errors have been divided by the machine precision *eps* for clarity. It can be seen that the new Divisional Bisection algorithm has the best accuracy as well as the Bisection method, considerably higher than the others.



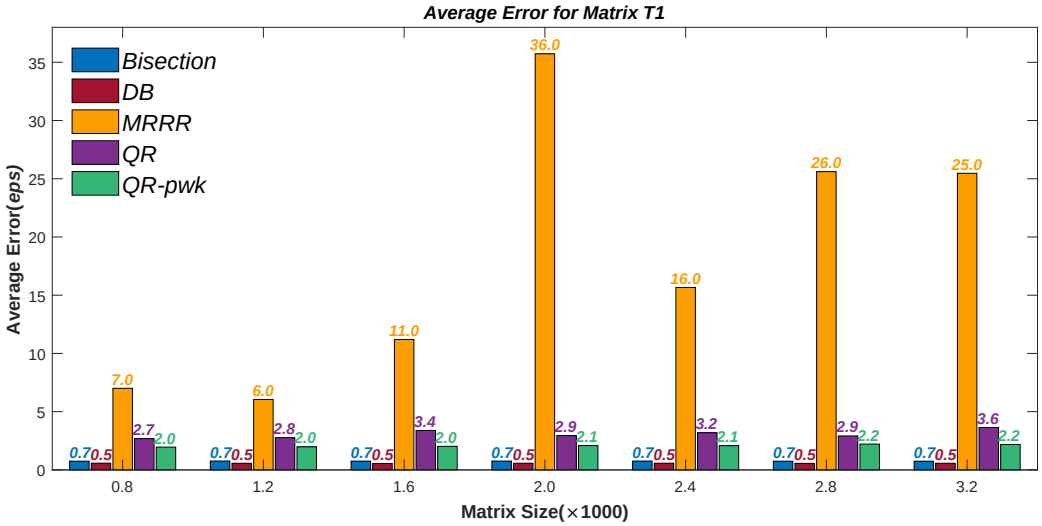
(**a**)

Figure 1. *Cont.*

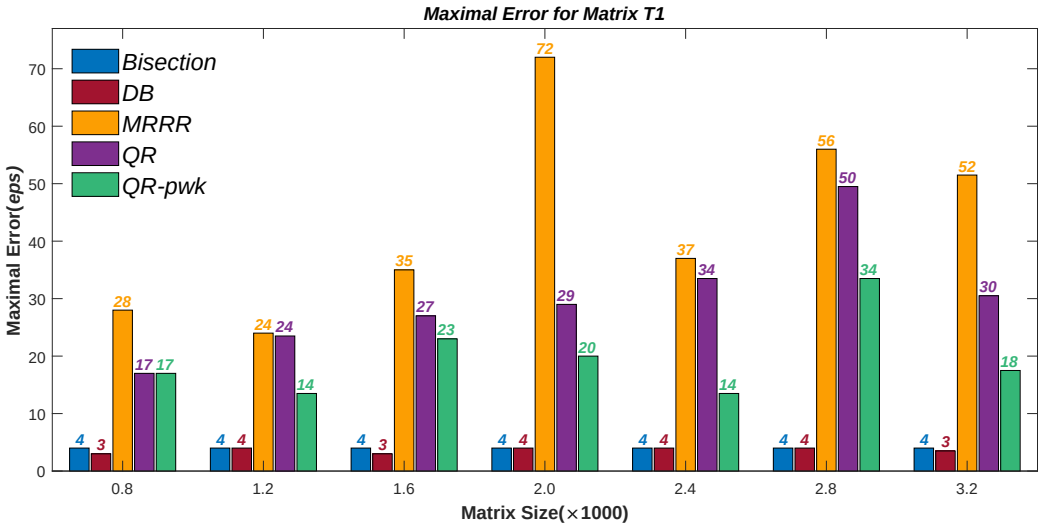


(**b**)

Figure 1. Results of Matrix A: (**a**) the Average Errors; (**b**) the Maximal Errors.



(**a**)



(**b**)

Figure 2. Results of Matrix T1: (**a**) the Average Errors; (**b**) the Maximal Errors.

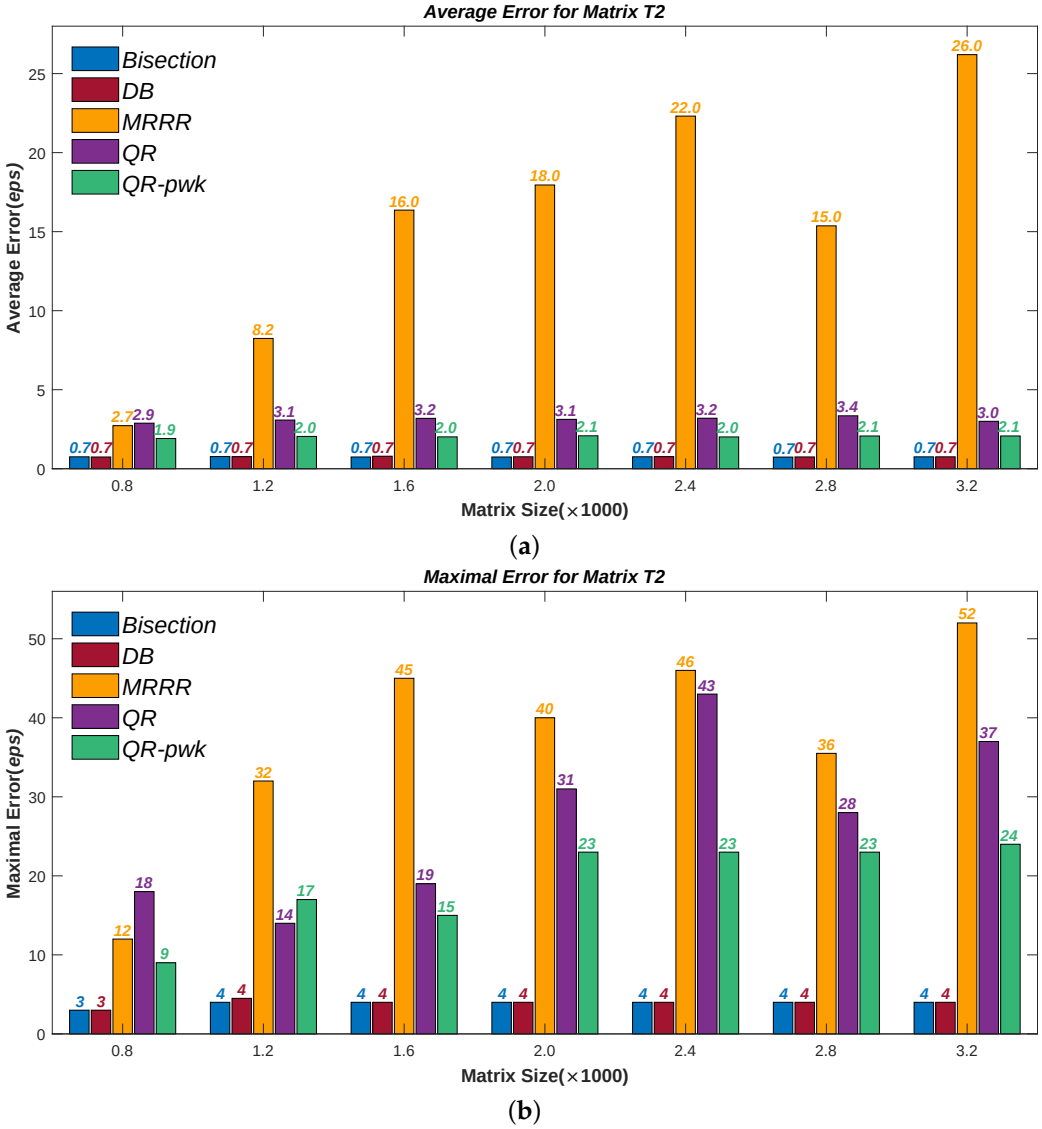


Figure 3. Results of Matrix T2: (**a**) the Average Errors; (**b**) the Maximal Errors.

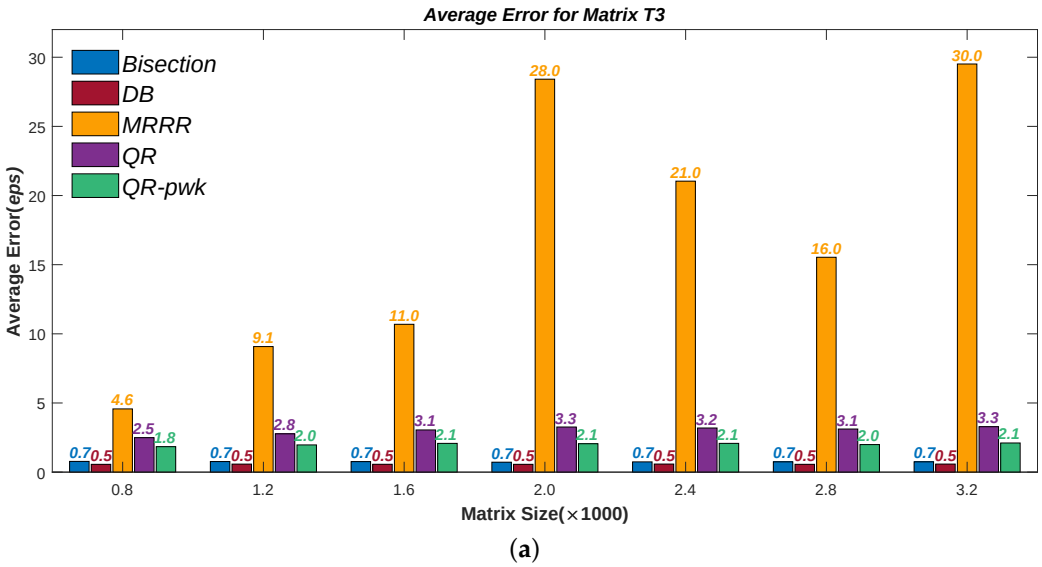
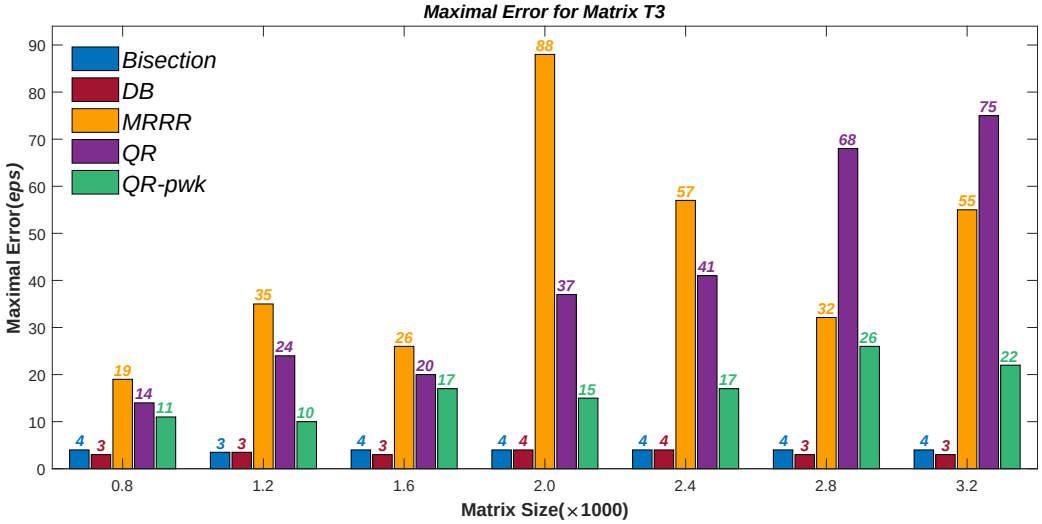


Figure 4. *Cont*.



(**b**)

Figure 4. Results of Matrix T3: (**a**) the Average Errors; (**b**) the Maximal Errors.

5.3. Efficiency Test for Computing all the Eigenvalues

Figure 5 presents the test results of time cost. Seven different sizes are used, from 800×800 to 3200×3200. Note that the results of the Random Matrix of each size are the mean data of 20 tests. Therefore, we use the plural form in the figures.

When the eigenvalues clutter, as in Matrix W, the Divisional Bisection method improves the Bisection method by about 70%. Such a good result can also be in Matrix T1 and Matrix T3. However, the improvement is less than 50% in Matrix A and Matrix T2. The reason is their submatrices have close eigenvalues to the global one but are not equal in finite precision arithmetic. For example, the sub-eigenvalues give an interval for Algorithm 6 and have an upper or lower bound that has a distance between λ_i less than 5×10^{-14}. The 'fzero' scheme uses the linear interpolation to accelerate convergence; such a bound produces poor slopes during the linear interpolation process. As a consequence, more iterations are needed to guarantee convergence, which finally results in the efficiency loss of the Divisional Bisection method. Recall that Algorithm 7 is for checking similar situations. However, a distance of 5×10^{-14} could not be detected, because it does not meet the conditions of Theorem 4.

Nevertheless, we are not pessimistic about the Divisional Bisection method. First, it still improves more than 35% in such cases and performs well for Random Matrices. Secondly, the 'fzero' scheme is not a prerequisite or non-replaceable in our method, which could be modified or substituted by a more powerful competitor in future follow-up studies.

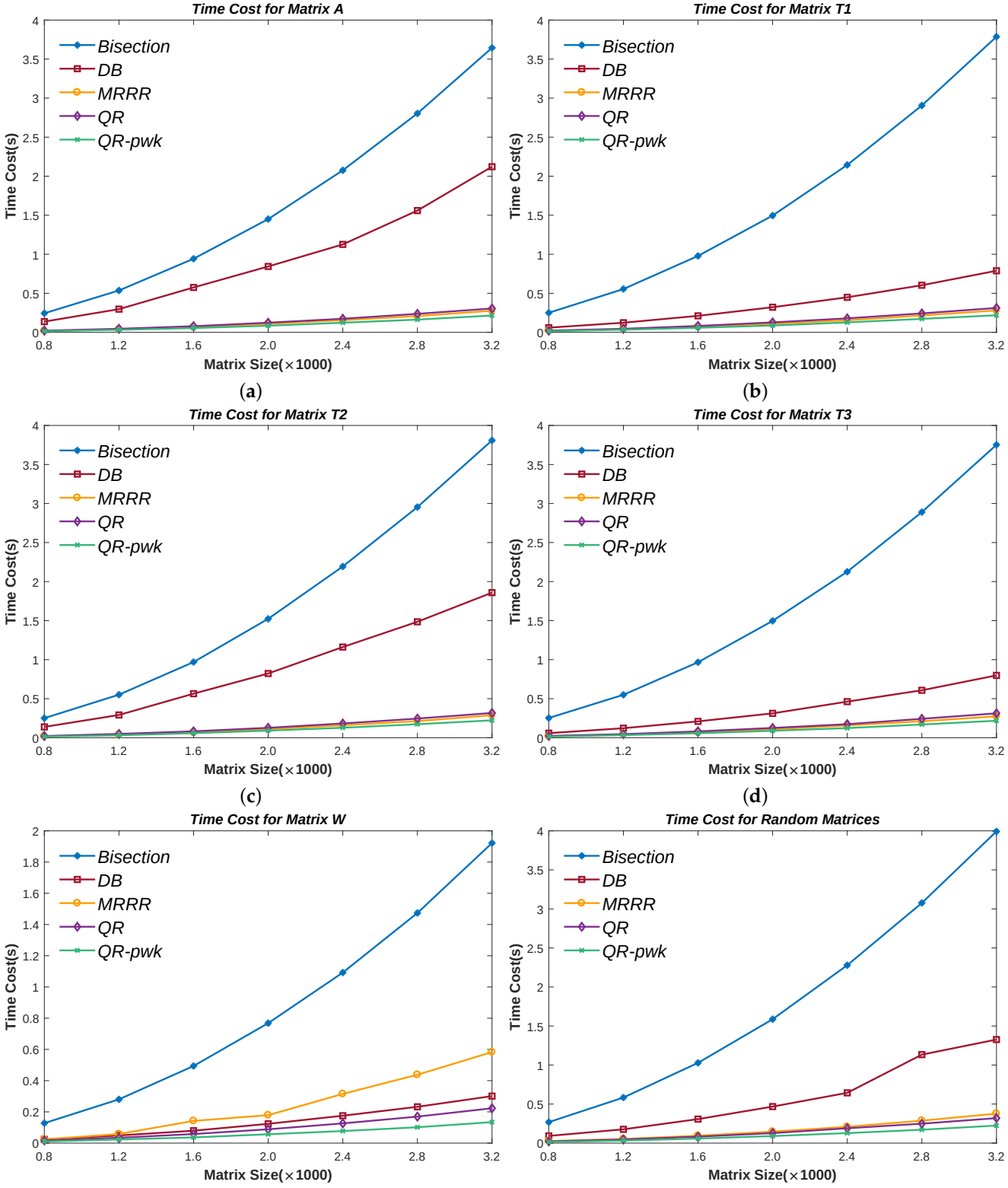


Figure 5. Time cost for: (**a**) Matrix A; (**b**) Matrix T1; (**c**) Matrix T2; (**d**) Matrix T3; (**e**) Matrix W; (**f**) Random Matrices.

5.4. Efficiency Test for Computing a Part of the Eigenvalues

All along, the Bisection method undertakes the task of computing a part of eigenvalues, especially when the size of the matrix is large. When Algorithm 5 obtains all the sub-eigenvalues, as shown in lines 2–11 in Algorithm 5, it is an easy task to calculate any

parts of λ_i's. For example, if eigenvalues in a certain interval are wanted, we can drop the sub-eigenvalues which are outside and substitute $\pm F$, in Algorithm 5 line 2 and line 15, with the upper and lower bounds of the given interval. If $r1$th$\sim r2$th eigenvalues are wanted, we need to drop the sub-eigenvalues that are of the order lower than $r1-1$ or higher than $r2$. When s_{r1-1} and s_{r2} are the substitutions of $\pm F$, the problem can be solved.

Figure 6 shows the time cost in Random Matrices of four relatively large size, i.e., 5000 × 5000, 10,000 × 10,000, 15,000 × 15,000, and 20,000 × 20,000. We calculated 1%, 10%, 30%, and 50% λ_i's of each size. Note the results are mean data of 40 tests, 20 for computing λ_i's in a certain interval and 20 for computing λ_i's in a certain order. Given that there is no evident difference between the test results of calculating λ_i's in an interval or order, we mixed them for averaging.

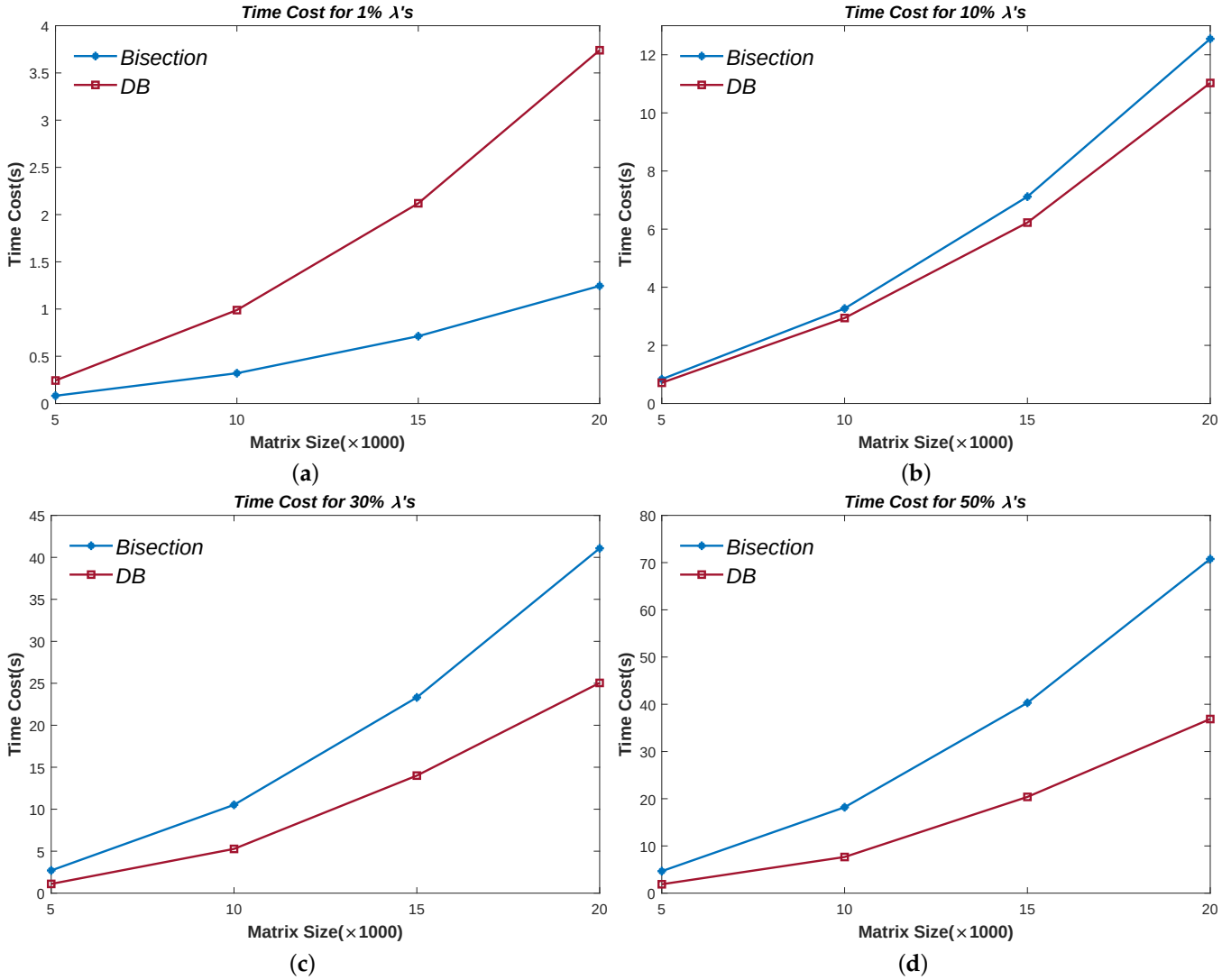


Figure 6. Time cost for: (**a**) 1% λ's; (**b**) 10% λ's; (**c**) 30% λ's; (**d**) 50% λ's.

The results show that the Divisional Bisection method is not suitable for computing a small group of eigenvalues, despite the matrix being relatively large. We consider 10% as an applicable threshold. Although we can replace the QR method with the Bisection method in Algorithm 5 line 3, which could avoid the calculation of all the sub-eigenvalues, the result seems even worse. As the matrix size increases, the efficiency disadvantage of the Bisection method becomes increasingly severe, which could ignore only a quite small number of wanted λ_i, for example, 0.1%. In this case, the 'fzero' loops (line 12 to line 24 in

Algorithm 5) become a heavy burden to the Divisional Bisection method. Therefore, we insist on using the PWK version of the QR method in Algorithm 5.

We now consider the situation of calculating one λ in parallel. The problem also arises when the number of wanted λ is less than the number of CPU cores or not divisible by it. Algorithm 4 solves the problem and makes it available for computing with any number of CPU cores. Of course, the need to compute an eigenvalue in parallel must occur in a very large matrix. Therefore, we use three Random Matrices with sizes of $10^6 \times 10^6$, $10^7 \times 10^7$, and $10^8 \times 10^8$ for the test of parallel efficiency. The results, presented in Figure 7, were collected on an Intel Xeon(R) Core E5-2687 3.1-GHz CPU and 256-GB RAM machine, which has 20 CPU cores. Note that the results are mean data of 20 tests.

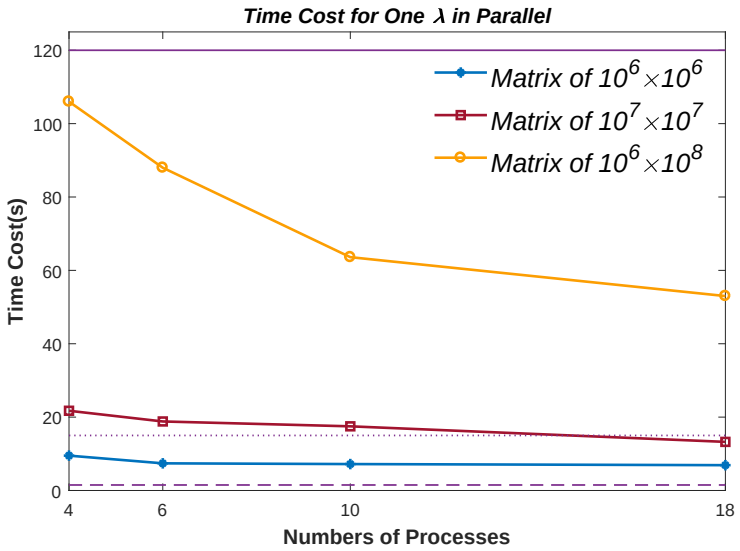


Figure 7. Computing one λ in parallel.

The three purple horizontal lines in Figure 7 denote the time cost of the serial Bisection algorithm. Specifically, the top one denotes the time cost for $10^8 \times 10^8$ Random Matrices, the middle $10^7 \times 10^7$, and the bottom $10^6 \times 10^6$. The parallel efficiency is unsatisfactory, especially for the $10^7 \times 10^7$ and $10^6 \times 10^6$ Random Matrices, which are even worse than the serial Bisection algorithm. The reason is that Matlab is not available for multi-threaded computation. Instead, we run the codes in multi-processes. The task of copying inputs and distributing them to the processes takes up the vast majority of the time. The script time consumption analysis tool in Matlab confirms our point, which shows at least 75% time was consumed during copying and distributing. Therefore, we would focus on the version written in C or Fortran of the Divisional Bisection algorithm in future follow-up studies. Nevertheless, Figure 7 verifies the feasibility of Algorithm 4, which to our knowledge is the only algorithm that works in parallel for computing any one ST eigenvalue. This paper also focuses on the serial version.

6. Conclusions

In this paper, a novel $O(n^2)$ Divisional Bisection method is given for the ST eigenvalue problem by Algorithms 4 and 5. When computing all eigenvalues, the results show that the time cost is reduced by more than 35–70% on serial machines compared to the Bisection algorithm. In addition,

1. The algorithms are easy to implement fully in parallel;
2. By Algorithm 4, even one eigenvalue can be calculated in parallel and distributed on any number of CPU cores;
3. As with the Bisection algorithm, it is flexible to set the expected accuracy and the computing error archives machine precision;
4. By Algorithm 4, it is practicable to calculate a single eigenvalue of any order;

5. Combining Algorithms 4 and 5, it is practicable to calculate eigenvalues in any interval in parallel or any orders.

The Divisional Bisection method offers a novel idea for solving the ST eigenvalue problem and a new choice, especially for readers who care about an algorithm of good parallelization, flexibility, and warranted accuracy.

Author Contributions: Formal analysis, W.C., Y.Z. and H.Y.; investigation, W.C. and Y.Z.; writing—original draft, W.C.; writing—review and editing, H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by the Talent Team Project of Zhangjiang City in 2021 and the R & D and industrialization project of the offshore aquaculture cage nets system of Guangdong Province of China (grant No. 2021E05034). Huazhong University of Science and Technology funds the APC.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the editors and reviewers for their constructive comments, which will improve the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DC (algorithm)	Divided and Conquer (algorithm)
MRRR (algorithm)	Multiple Relatively Robust Representations (algorithm)
ST (matrix)	Symmetric Tridiagonal (matrix)

References

1. Penke, C.; Marek, A.; Vorwerk, C.; Draxl, C.; Benner, P. High performance solution of skew-symmetric eigenvalue problems with applications in solving the Bethe-Salpeter eigenvalue problem. *Parallel Comput.* **2020**, *96*, 102639. [CrossRef]
2. Xu, W.R.; Bebiano, N.; Chen, G.L. On the construction of real non-self adjoint tridiagonal matrices with prescribed three spectra. *Electron. Trans. Numer. Anal.* **2019**, *51*, 363–386. [CrossRef]
3. Wei, Y.; Zheng, Y.; Jiang, Z.; Shon, S. A Study of Determinants and Inverses for Periodic Tridiagonal Toeplitz Matrices with Perturbed Corners Involving Mersenne Numbers. *Mathematics* **2019**, *7*, 893. [CrossRef]
4. Tanasescu, A.; Carabas, M.; Pop, F.; Popescu, P.G. Scalability of k-Tridiagonal Matrix Singular Value Decomposition. *Mathematics* **2021**, *9*, 3123. [CrossRef]
5. Bala, B.; Manafov, M.D.; Kablan, A. Inverse Spectral Problems for Spectral Data and Two Spectra of N by N Tridiagonal Almost-Symmetric Matrices. *Appl. Appl. Math.* **2019**, *14*, 1132–1144.
6. Bartoll, S.; Jiménez-Munguía, R.R.; Martínez-Avendaño, R.A.; Peris, A. Chaos for the Dynamics of Toeplitz Operators. *Mathematics* **2022**, *10*, 425. [CrossRef]
7. Nesterova, O.P.; Uzdin, A.M.; Fedorova, M.Y. Method for calculating strongly damped systems with non-proportional damping. *Mag. Civ. Eng.* **2018**, *81*, 64–72. [CrossRef]
8. Bahar, M.K. Charge-Current Output in Plasma-Immersed Hydrogen Atom with Noncentral Interaction. *Ann. Der Phys.* **2021**, *533*, 2100111. [CrossRef]
9. Geng, X.; Lei, Y. On the Kirchhoff Index and the Number of Spanning Trees of Linear Phenylenes Chain. *Polycycl. Aromat. Compd.* **2021**. [CrossRef]
10. Neo, V.W.; Naylor, P.A. Second order sequential best rotation algorithm with householder reduction for polynomial matrix eigenvalue decomposition. In Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019, pp. 8043–8047.
11. Vazquez, A. Transition to multitype mixing in d-dimensional spreading dynamics. *Phys. Rev. E* **2021**, *103*, 022309. [CrossRef]
12. Wilkinson. The Algebraic Eigenvalue Problem. In *Handbook for Automatic Computation*; Volume II: Linear Algebra; Oxford University Press: Oxford, UK, 1969.
13. Alqahtani, A.; Gazzola, S.; Reichel, L.; Rodriguez, G. On the block Lanczos and block Golub-Kahan reduction methods applied to discrete ill-posed problems. *Numer. Linear Algebra Appl.* **2021**, *28*, e2376. [CrossRef]
14. Marques, O.; Demmel, J.; Vasconcelos, P.B. Bidiagonal SVD Computation via an Associated Tridiagonal Eigenproblem. *ACM Trans. Math. Softw.* **2020**, *46*, 1–25. [CrossRef]

15. Chen, M.F.; Li, Y.S. Development of powerful algorithm for maximal eigenpair. *Front. Math. China* **2019**, *14*, 493–519. [CrossRef]
16. Coelho, D.F.G.; Dimitrov, V.S.; Rakai, L. Efficient computation of tridiagonal matrices largest eigenvalue. *J. Comput. Appl. Math.* **2018**, *330*, 268–275. [CrossRef]
17. Tang, T.; Yang, J. Computing the Maximal Eigenpairs of Large Size Tridiagonal Matrices with O(1) Number of Iterations. *Numer. Math. Theory Methods Appl.* **2018**, *11*, 877–894. [CrossRef]
18. Francis, J.G. The QR transformation a unitary analogue to the LR transformation—Part 1. *Comput. J.* **1961**, *4*, 265–271. [CrossRef]
19. Francis, J.G. The QR transformation—Part 2. *Comput. J.* **1962**, *4*, 332–345. [CrossRef]
20. Myllykoski, M. Algorithm 1019: A Task-based Multi-shift QR/QZ Algorithm with Aggressive Early Deflation. *ACM Trans. Math. Softw.* **2022**, *48*, 11. [CrossRef]
21. Ortega, J.M.; Kaiser, H.F. The LLT and QR methods for symmetric tridiagonal matrices. *Comput. J.* **1963**, *6*, 99–101. [CrossRef]
22. Parlett, B.N. *The Symmetric Eigenvalue Problem*; SIAM: Philadelphia, PA, USA, 1997.
23. Stewart, G.W. A parallel implementation of the QR-algorithm. *Parallel Comput.* **1987**, *5*, 187–196. [CrossRef]
24. Granat, R.; Kagstrom, B.; Kressner, D. A novel parallel QR algorithm for hybrid distributed memory HPC systems. *SIAM J. Sci. Comput.* **2010**, *32*, 2345–2378. [CrossRef]
25. Matstoms, P. Parallel sparse QR factorization on shared memory architectures. *Parallel Comput.* **1995**, *21*, 473–486. [CrossRef]
26. Kaufman, L. A Parallel QR Algorithm for the Symmetrical Tridiagonal Eigenvalue Problem. *J. Parallel Distrib. Comput.* **1994**, *23*, 429–434. [CrossRef]
27. Ballard, G.; Demmel, J.; Grigori, L.; Jacquelin, M.; Knight, N. A 3d parallel algorithm for qr decomposition. In Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, Vienna, Austria, 16–18 July 2018; pp. 55–65.
28. Dhillon, I.S. A New O (N^2) Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem. Doctoral Thesis, University of California, Berkeley, CA, USA, 1997.
29. Parlett, B.N.; Marques, O.A. An implementation of the dqds algorithm (positive case). *Linear Algebra Its Appl.* **2000**, *309*, 217–259. [CrossRef]
30. Fukuda, A.; Yamamoto, Y.; Iwasaki, M.; Ishiwata, E.; Nakamura, Y. Convergence acceleration of shifted LR transformations for totally nonnegative hessenberg matrices. *Appl. Math.* **2020**, *65*, 677–702. [CrossRef]
31. Cuppen, J.J. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.* **1980**, *36*, 177–195. [CrossRef]
32. Liao, X.; Li, S.; Lu, Y.; Roman, J.E. A Parallel Structured Divide-and-Conquer Algorithm for Symmetric Tridiagonal Eigenvalue Problems. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 367–378. [CrossRef]
33. Li, S.; Rouet, F.H.; Liu, J.; Huang, C.; Gao, X.; Chi, X. An efficient hybrid tridiagonal divide-and-conquer algorithm on distributed memory architectures. *J. Comput. Appl. Math.* **2018**, *344*, 512–520. [CrossRef]
34. Kahan, W. *Accurate Eigenvalues of a Symmetric Tri-Diagonal Matrix*; Report; Dept. of Computer Science, Stanford University: Stanford, CA, USA, 1966.
35. Ralha, R. Mixed Precision Bisection. *Math. Comput. Sci.* **2018**, *12*, 173–181. [CrossRef]
36. Muir, T.; Metzler, W.H. *A Treatise on the Theory of Determinants*; Dover Publications: Mineola, NY, USA, 1960.
37. Denton, P.; Parke, S.; Tao, T.; Zhang, X. Eigenvectors from eigenvalues: A survey of a basic identity in linear algebra. *Bull. Am. Math. Soc.* **2022**, *59*, 31–58. [CrossRef]
38. Gu, M.; Eisenstat, S.C. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.* **1995**, *16*, 172–191. [CrossRef]
39. Li, T.Y.; Zeng, Z. The Laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited. *SIAM J. Sci. Comput.* **1994**, *15*, 1145–1173. [CrossRef]
40. Dekker, T.J. Finding a zero by means of successive linear interpolation. In *Constructive Aspects of the Fundamental Theorem of Algebra*; Wiley: Hoboken, NJ, USA, 1969; pp. 37–51.
41. Wilkinson, J.H. *Two Algorithms Based on Successive Linear Interpolation*; Stanford University: Stanford, CA, USA, 1967.
42. Brent, R.P. *Algorithms for Minimization without Derivatives*; Prentice-Hall: Hoboken, NJ, USA, 1973.
43. Bernstein, H.J. An accelerated bisection method for the calculation of eigenvalues of a symmetric tridiagonal matrix. *Numer. Math.* **1984**, *43*, 153–160. [CrossRef]
44. Bhatia, R. *Perturbation Bounds for Matrix Eigenvalues*; SIAM: Philadelphia, PA, USA, 2007.
45. Da Fonseca, C.M.; Kowalenko, V. Eigenpairs of a family of tridiagonal matrices: Three decades later. *Acta Math. Hung.* **2020**, *160*, 376–389. [CrossRef]
46. Ferreira, C.; Parlett, B. Eigenpairs of Wilkinson Matrices. *SIAM J. Matrix Anal. Appl.* **2020**, *41*, 1388–1415. [CrossRef]