# The Design and Implementation of the MRRR Algorithm

INDERJIT S. DHILLON
University of Texas, Austin
and
BERESFORD N. PARLETT and CHRISTOF VÖMEL
University of California, Berkeley

In the 1990's, Dhillon and Parlett devised the algorithm of multiple relatively robust representations (MRRR) for computing numerically orthogonal eigenvectors of a symmetric tridiagonal matrix $T$ with $\mathcal{O}(n^2)$ cost. While previous publications related to MRRR focused on theoretical aspects of the algorithm, a documentation of software issues has been missing. In this article, we discuss the design and implementation of the new MRRR version STEGR that will be included in the next LAPACK release. By giving an algorithmic description of MRRR and identifying governing parameters, we hope to make STEGR more easily accessible and suitable for future performance tuning. Furthermore, this should help users understand design choices and tradeoffs when using the code.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra

General Terms: Algorithms, Design

Additional Key Words and Phrases: Multiple relatively robust representations, eigenvalues, eigenvectors, symmetric matrix, LAPACK, numerical software, design, implementation

## 1. INTRODUCTION

The algorithm of multiple relatively robust representations (MRRR, or MR$^3$) for the symmetric tridiagonal eigenvalue problem is an important improvement on standard inverse iteration [Ipsen 1997] in that it computes orthogonal eigenvectors without needing Gram-Schmidt orthogonalization. While the theoretical foundation of the MRRR algorithm has already been described [Dhillon 1997;

This work has been supported by Grant no. ACI-9619020, National Science Foundation Cooperative Agreement (NPACI), NSF grant CCF-0431257, and NSF Career Award ACI-0093404.
Authors' addresses: I. S. Dhillon, Department of Computer Science, University of Texas, Austin, TX 78712-1188; email: inderjit@cs.utexas.edu; B. N. Parlett, Mathematics Department and Computer Science Division, University of California, Berkeley, CA 94720; email: parlett@math.berkeley.edu; C. Vömel, Computer Science Division, University of California, Berkeley, CA 94720; email: voemel@cs.berkeley.edu.

Parlett and Dhillon 1997, 2000; Dhillon and Parlett 2004a, 2004b], this article addresses the question of how to translate the theory into reliable numerical software. Our presentation is based on the design and implementation of STEGR, the successor to the first LAPACK 3.0 [Anderson et al. 1999] version of the MRRR algorithm.

This article is structured as follows. In Section 2, we give a brief summary of the most important principles behind the MRRR algorithm that are necessary for understanding the code design and implementation. In Section 3, we describe the features of the main LAPACK driver, STEGR. Later sections describe the computational routines, the most important being Sections 4 and 5 for computing the eigenvalues and eigenvectors, respectively. These two functions depend upon and call a number of auxiliary functions that are explained in Sections 6, 7, 8, 9, and 10. Finally, in Section 11 we discuss the roles of special topics, such as performance analysis, IEEE arithmetic, parallelization, and vectorization.

As a guiding principle, we first give a rough, but more easily understandable description of the main algorithm prior to a more exhaustive explanation of the algorithmic details. Every section contains a subsection that explains the importance of key parameters and their impact on both accuracy and efficiency.

In addition to the algorithmic description of the routines STEGR, LARRE, and LARRV, we give an overview of storage requirements and a graphical illustration of the functionalities in order to clarify the structure and design of the software.

## 2. OVERVIEW OF MRRR

For a symmetric tridiagonal matrix $T \in \mathcal{R}^{n\times n}$, the MRRR algorithm promises to compute a set of eigenvalues $\hat{\lambda}_1, \dots, \hat{\lambda}_k$ and a set of vectors $\hat{v}_1, \dots, \hat{v}_k$, $\|\hat{v}_i\|_2 = 1$, satisfying

$$\|(T - \hat{\lambda}_i I)\hat{v}_i\| = \mathcal{O}(n\epsilon\|T\|) \tag{1}$$

$$|\hat{v}_i^T \hat{v}_j| = \mathcal{O}(n\epsilon), \quad i \neq j \tag{2}$$

using $\mathcal{O}(nk)$ arithmetic operations. Here $\epsilon$ is the roundoff unit. Pairs of vectors that satisfy Eq. (2) are said to be numerically orthogonal or orthogonal to working precision. A stricter criterion for small residual norms is

$$\|(T - \hat{\lambda}_i I)\hat{v}_i\| = \mathcal{O}(n\epsilon|\hat{\lambda}_i|), \tag{3}$$

demanding *relatively* small residuals.

Section 2.1 presents the main ideas behind the MRRR algorithm. A more technical description is given in Section 2.2.

### 2.1 Informal Outline of the Underlying Ideas

Our discussion is based on the results from Dhillon [1997], Dhillon and Parlett [2004a, 2004b], and Parlett and Dhillon [1997, 2000] that justify the MRRR algorithm. For a review of the theory of inverse iteration, we refer to Ipsen [1997].

2.1.1 *Issues in Standard Inverse Iteration.* We consider inverse iteration for a general symmetric matrix $T$ and some of its important properties.

—**(1A)** Small residuals, in the sense of Eq. (1), are not enough to guarantee orthogonality to working precision when eigenvalues are close.

—**(1B)** For an approximation $\hat{\lambda}$ to eigenvalue $\lambda$, define $\text{gap}(\hat{\lambda}) = \min\{|\hat{\lambda} - \mu| : \lambda \neq \mu, \mu \in \text{spectrum}(T)\}$. If two eigenvalues are of the same magnitude as their gaps, then their approximate eigenvectors whose residuals satisfy Eq. (3) will be orthogonal to working precision. Stated differently: When two computed eigenvalues have a large relative gap ($\text{relgap}(\hat{\lambda}) = \text{gap}(\hat{\lambda})/|\hat{\lambda}|$) and the corresponding computed eigenvectors satisfy Eq. (3), then they also satisfy (2).

—**(1C)** If $T$ determines an eigenvalue $\lambda$ to high relative accuracy, and if $\hat{\lambda}$ approximates $\lambda$ to high relative accuracy, then it is possible to use one step of inverse iteration with a good starting vector in order to achieve Eq. (3). Moreover, at least one column of $(T - \hat{\lambda}I)^{-1}$ is sufficiently good as a starting vector to guarantee (3), namely, the column with the largest norm.

2.1.2 *A High Level Description of the MRRR Algorithm.* The observations from Section 2.1.1 suggest the following "ideal" Algorithm 1. This has been described in more detail as Algorithm Getvec in Dhillon and Parlett [2004b].

2.1.3 *Difficulties.* The principal challenges facing Algorithm 1 are:

—The eigenvalue must be defined to high relative accuracy by the data. No computed eigenvalue can have an error of less than the uncertainty in the eigenvalue. Uncertainties in matrix entries can arise from the data itself, or as a result of transformations. Unfortunately, small relative changes in the entries of some tridiagonal matrices can cause large relative changes in tiny eigenvalues.

—An approximation to an eigenvalue (however small) must be computed to high relative accuracy.

—A good starting vector for inverse iteration must be found as inexpensively as possible, ideally with $\mathcal{O}(n)$ work.

—Eigenvalues might not have large relative gaps.

—Rounding errors must not spoil the computed output.

2.1.4 *MRRR: How to Overcome the Obstacles.* The MRRR algorithm addresses all of these issues, resulting in an $\mathcal{O}(nk)$ algorithm for computing $k$ eigenpairs. Specifically, it is based on the following results.

**Algorithm 1.** High-level description of the MRRR algorithm

```
for each eigenvalue with a large relative gap do
  Compute an eigenvalue approximation that is good to high relative accuracy.
  Find the column $r$ of $(T - \hat{\lambda}I)^{-1}$ with largest norm.
  Perform one step of inverse iteration, $(T - \hat{\lambda}I)z = e_r$.
end for
```

—**(2A)** For a tridiagonal matrix $T$ and most, but not all, shifts $\sigma$, the standard triangular factorization $T - \sigma I = LDL^T$, $L$ unit bidiagonal, has the property that small relative changes in the nontrivial entries of $L$ and $D$ cause small relative changes in each small eigenvalue of $LDL^T$ (despite possible element growth in the factorization). Thus, the algorithm works with $LDL^T$ factorizations, instead of $T$ [Dhillon 1997; Dhillon and Parlett 2004b, Parlett 2000, 2003, 2005].

—**(2B)** For a given bidiagonal factorization $LDL^T$ that determines its eigenvalues to high relative accuracy, bisection can compute an eigenvalue to high relative accuracy with $\mathcal{O}(n)$ work.

—**(2C)** For symmetric tridiagonal matrices, it is possible to find the column of $(LDL^T - \hat{\lambda}I)^{-1}$ with the largest norm in $\mathcal{O}(n)$ operations.

—**(2D)** In order to apply Algorithm 1, eigenvalues must have large relative gaps. MRRR uses the observation that shifts alter relative gaps. If the relative gaps within a cluster of eigenvalues are too small, then they can be increased by shifting the origin close to one end of the group. Furthermore, the procedure can be repeated for clusters within clusters of close eigenvalues.

—**(2E)** Differential qd algorithms compute the different (shifted) $LDL^T$ factorizations with a special property: Tiny relative changes to the input $(L, D)$ and output $(L_+, D_+)$ with $LDL^T - \sigma I = L_+D_+L_+^T$ give an exact relation [Dhillon 1997; Dhillon and Parlett 2004b]. This property ensures that roundoff does not spoil Algorithm 1.

2.1.5 *Glossary of Useful Terms.* The following terms are used in the rest of this article.

—*(Sub)block*: If $T$ has any negligible off-diagonal entries, then it is split into principal submatrices, called blocks, each of which has off-diagonal entries that exceed some threshold, either absolute or relative to neighboring diagonal entries.

—*Representation*: A triangular factorization $T - \sigma I = LDL^T$ of a symmetric tridiagonal matrix $T$.

—*Relatively robust representation (RRR)*: A representation that determines a subset $\Gamma$ of the eigenvalues to high relative accuracy. This means that small relative changes in the entries of $L$ and $D$ cause small relative changes in each $\lambda \in \Gamma$ (the sensitivity can be quantified by a condition number [Parlett and Dhillon 2000]).

—*Root representation* and *representation tree*: The MRRR procedure for computing eigenpairs is best represented by a rooted tree. Each node of the graph is a pair consisting of a representation $(L, D)$ and subset $\Gamma$ of the desired eigenvalues for which it is an RRR. The root node of the representation tree is the initial representation that is an RRR for all the desired eigenvalues (note that every positive or negative definite bidiagonal factorization is an RRR for all eigenvalues).

—*Singleton*: a (shifted) eigenvalue whose relative separation from its neighbors exceeds a given threshold.

We now can give a more detailed description of the essence of the MRRR algorithm that extends Algorithm 1.

## 2.2 More on the Ideas Behind the MRRR Algorithm

It is difficult to understand MRRR without appreciating the limitations of standard inverse iteration. We give here a more detailed account of the issues raised in Section 2.1.1 and the properties of MRRR summarized in Section 2.1.4.

2.2.1 *Issues and Challenges.* **(1A)** First, for any symmetric matrix $A$, let $\lambda, \mu$ denote machine representations of two eigenvalues and $v, w$ those of the corresponding (normalized) eigenvectors. Define the residuals $r = (A - \lambda I)v$ and $s = (A - \mu I)w$, then

$$v^T w = \frac{1}{\mu - \lambda}(r^T w - v^T s). \tag{4}$$

The two dot products on the righthand side will not vanish, in general. Furthermore, there is no intrinsic reason why there should be cancellation in the numerator. Thus $|v^T w| = \mathcal{O}\left(\frac{n\epsilon\|A\|}{|\mu-\lambda|}\right)$ is realistic. Consequently, it has been standard practice for inverse iteration to employ orthogonalization of $v$ and $w$ explicitly when the eigenvalues are close. This makes the algorithm expensive for large clusters, in the extreme case, up to $\mathcal{O}(nk^2)$ work.

**(1B)** Now, suppose that, Eq. (3) holds for $\lambda$ and for $\mu$. Substitute in Eq. (4) and take absolute values in (1) to find

$$|v^T w| = \mathcal{O}\left(\frac{n\epsilon(|\lambda| + |\mu|)}{|\mu - \lambda|}\right). \tag{5}$$

Hence, (5) guarantees numerically orthogonal eigenvectors, provided that the separation $|\mu - \lambda|$ is not *much* smaller than $|\lambda|$ and $|\mu|$. Thus, our goal is to achieve Eq. (3). For a general symmetric matrix, we do not know how to do this, but for tridiagonal matrices it is feasible.

**(1C)** If the approximation $\hat{\lambda}$ is closer to $\lambda$ than to any other eigenvalue, then $\|(A - \hat{\lambda}I)^{-1}\|^{-1} = |\lambda - \hat{\lambda}|$. Furthermore, this is an attainable lower bound for the residual $r(\hat{v}, \hat{\lambda}) = (A - \hat{\lambda}I)\hat{v}$, with $\|\hat{v}\| = 1$. In general, it is too expensive to find an optimal vector that minimizes the residual norm. However, in the 1960's,

**Algorithm 2.** Essence of the MRRR algorithm

```
Given an RRR $LDL^T$ for a set of eigenvalues
for each eigenvalue $\hat{\lambda}$ with a large relative gap do
  Compute the eigenvalue $\hat{\lambda}$ to high relative accuracy.
  Find the column $r$ of $(LDL^T - \hat{\lambda}I)^{-1}$ with largest norm.
  Perform one step of inverse iteration, $(LDL^T - \hat{\lambda}I)v_r = e_r$.
end for
for each of the remaining clusters of eigenvalues do
  Choose shift $\sigma$ outside the cluster.
  Compute new RRR, $L_+D_+L_+^T = LDL^T - \sigma I$.
  Refine the eigenvalues.
end for
```

Varah and Wilkinson [Ipsen 1997] showed that for at least one canonical vector, say $e_r$, $\|(A-\hat{\lambda}I)^{-1}e_r\| \geq \|(A-\hat{\lambda}I)^{-1}\|/\sqrt{n}$. Let $\widetilde{v} = (A-\hat{\lambda}I)^{-1}e_r$ and $\bar{v} = \widetilde{v}/\|\widetilde{v}\|$. Then, in exact arithmetic,

$$\|r(\bar{v},\hat{\lambda})\| = 1/\|\widetilde{v}\| \leq \sqrt{n}|\lambda-\hat{\lambda}|. \tag{6}$$

Thus, the canonical vector $e_r$, when used as the starting vector for inverse iteration, yields an iterate with residual that is within a factor of $\sqrt{n}$ of the optimum. Moreover, if $\hat{\lambda}$ approximates $\lambda$ to high relative accuracy, that is, $|\lambda - \hat{\lambda}| = \mathcal{O}(\epsilon|\lambda|)$, then $\bar{v}$ achieves (3).

2.2.2 *Key Results and Principles of MRRR.* **(2A)** Kahan discovered that the Cholesky factor $\widetilde{L}$ of a symmetric positive definite tridiagonal matrix determines all of the eigenvalues of $\widetilde{L}\widetilde{L}^T$ to high relative accuracy [Demmel and Kahan 1990]. More surprising still is the fact that most, but not all, *indefinite* $LDL^T$ representations determine their tiny eigenvalues to high relative accuracy, despite element growth. If $\lambda, v$ is an eigenpair of $LDL^T$, $\|v\| = 1, \lambda \neq 0$ and $\rho$ denotes the spectral diameter, then $\lambda$ is determined to high relative accuracy by $L$ and $D$, provided that

$$\max\left\{\frac{v^TL|D|L^Tv}{|\lambda|}, \frac{\|Dv\|_\infty}{\rho}\right\} \leq K, \quad K = \mathcal{O}(1).$$

This shows that a factorization can still be an RRR for $\lambda$, despite possibly large entries in $D$ and $L$, as long as these are neutralized by small entries in $v$ [Dhillon and Parlett 2004a]. Also, when there is no element growth, $LDL^T$ is an RRR for all the eigenvalues, for example, when $LDL^T$ is positive (or negative) definite.

**(2B)** Bisection is based on a function that counts the number of eigenvalues of a given matrix of less than a given value. A short proof of the backward stability of this computation for a symmetric tridiagonal matrix $T$ in floating point arithmetic is given in Demmel [1997], and a more careful analysis in Demmel et al. [1995]. A comparable analysis with commutative diagrams and proofs of the mixed stability of the differential stationary qds factorization $LDL^T - \sigma I = L_+D_+L_+^T$, the differential progressive qds factorization $LDL^T - \sigma I = U_-D_-U_-^T$, and twisted factorizations $LDL^T - \sigma I = N_r\Delta_rN_r^T$ (see 2E) are given in Dhillon and Parlett [2004b].

**(2C)** Inexpensive ways to find the column of $(T-\hat{\lambda}I)^{-1}$ with largest norm were discovered in the mid-1990's independently by Godunov and by Fernando (see Parlett and Dhillon [1997] and the references therein). Given an RRR, let $\hat{\lambda}$ be a relatively accurate approximation of $\lambda$. Define for $k = 1,\ldots,n$ the vector

$$(LDL^T - \hat{\lambda}I)v_k = e_k\gamma_k, \quad v_k(k) = 1. \tag{7}$$

The normalization factors $\gamma_k$,

$$\gamma_k^{-1} = [(LDL^T - \hat{\lambda}I)^{-1}]_{kk}, \tag{8}$$

can be computed from a double factorization

$$(LDL^T - \hat{\lambda}I) = L_+D_+L_+^T = U_-D_-U_-^T \tag{9}$$

with $\mathcal{O}(n)$ work. Various formulae for the $\gamma_k$ with different stability properties are given in Dhillon [1997] and Dhillon and Parlett [2004b]. Once all $\gamma_k$ are known, we choose a suitable one, say $k = r$, and solve the corresponding Eq. (7) in a careful way. The choice is guided by the idea of obtaining a righthand side with a small angle to the (true) eigenvector.

Let $v$ denote the eigenvector for the eigenvalue $\lambda$ closest to the approximation $\hat{\lambda}$, and define $\delta\lambda = |\lambda - \hat{\lambda}|$. Then from an eigenvector expansion, it is shown in Parlett and Dhillon [1997] that

$$\frac{\|(LDL^T - \hat{\lambda}I)v_r\|_2}{\|v_r\|_2} = \frac{|\gamma_r|}{\|v_r\|_2} \le \frac{\delta\lambda}{\|v\|_\infty} \le \frac{\delta\lambda}{|v(r)|}. \tag{10}$$

In particular, if $r$ is chosen such that $|v(r)| \ge 1/\sqrt{n}$, then an upper bound on Eq. (10) is $\sqrt{n}\,\delta\lambda$. In practice, instead of finding the minimizing $r$ for the quotient $|\gamma_r|/\|v_r\|_2$ from (10), we choose

$$r = \arg \min_{1\le k\le n} |\gamma_k|. \tag{11}$$

In Parlett and Dhillon [1997], it is shown that as $\delta\lambda \to 0$, $(LDL^T - \hat{\lambda}I)^{-1}$ becomes essentially a rank-one matrix whose $k$th diagonal element converges to the square of the $k$th entry of the true eigenvector divided by $\delta\lambda$. Thus, by Eq. (8) and for small enough $\delta\lambda$, finding the largest component of the true eigenvector is equivalent to finding the minimum $\gamma_r$ of Eq. (11).

In Parlett and Dhillon [1997], the resulting $v_r$ from (7) is called the FP vector, (FP for Fernando and Parlett [1994]). The Davis-Kahan gap theorem [Davis and Kahan 1970; Parlett 1998; Parlett and Dhillon 1997] applied to (10) shows that $v_r$ approximates the true eigenvector $v$ with an error angle $\phi$ satisfying

$$|\sin\phi| \le \frac{\|LDL^T v_r - \hat{\lambda}v_r\|_2}{\|v_r\|_2\,\mathrm{gap}(\hat{\lambda})} \le \frac{\delta\lambda}{\|v\|_\infty\,\mathrm{gap}(\hat{\lambda})}, \tag{12}$$

where $\mathrm{gap}(\hat{\lambda}) = \min\{|\hat{\lambda} - \mu| : \lambda \ne \mu, \mu \in \mathrm{spectrum}(LDL^T)\}$. Furthermore, note that using (7), the Rayleigh quotient correction to $\hat{\lambda}$ can be shown to equal $\gamma/\|v_r\|_2^2$.

**(2D)** The guiding principle behind the MRRR algorithm is to compute, by differential stationary qds transformations, a sequence of RRRs that achieve large relative gaps by shifting as close as possible to eigenvalue groups identified as clusters. By this principle of *multiple representations*, MRRR refines clusters until it finds singletons with large enough relative gaps, and the computational path is described by a representation tree. In this tree, the root representation is given by an $LDL^T$ factorization of $T$ (with appropriate shift) that defines all the desired eigenvalues to high relative accuracy. The RRR of a child (which corresponds to either a subcluster or singleton within the parent) is computed from the parent representation by a differential qd transformation. Each leaf of the representation tree is a singleton from which the corresponding FP vector is computed. Algorithmically, the tree is built from the root down. The current node is processed by inspecting its set $\Gamma$ of (local) eigenvalues. As soon as a relative gap exceeding a threshold is encountered, the inspected subset is declared to be a new child node. If the child consists of more than one eigenvalue, a new RRR is computed and stored; if the child is a singleton, the FP vector is computed. The inspection then continues for the unexamined eigenvalues.

Dhillon [1997] observed that the intermediate RRRs (of nonsingletons) can be stored in the eigenvector matrix. Thus, no extra memory is needed for storing the RRRs.

**(2E)** One goal of Dhillon and Parlett [2004a, 2004b] is to show that roundoff does not spoil the procedure. Specifically, it is shown that (1) The computed pairs $\hat{\lambda}_i, \hat{v}_i$ have small residuals with respect to the root representation, and (2) the vectors computed from different representations are orthogonal to working accuracy and satisfy Eq. (2). The mixed relative stability of the differential qds algorithms is essential. Another key ingredient of the procedure for computing the FP vector is the use of a *twisted factorization* to solve Eq. (7). With the multipliers from (9), let

$$N_r = \begin{pmatrix} 1 & & & & & & & & \\ L_+(1) & 1 & & & & & & & \\ & \ddots & \ddots & & & & & & \\ & & L_+(r-2) & 1 & & & & & \\ & & & L_+(r-1) & 1 & U_-(r) & & & \\ & & & & & 1 & U_-(r+1) & & \\ & & & & & & 1 & \ddots & \\ & & & & & & & \ddots & U_-(n-1) \\ & & & & & & & & 1 \end{pmatrix},$$

and, with the pivots from (9) and $\gamma_r$ from (7), define

$$\Delta_r = \operatorname{diag}(D_+(1), \ldots, D_+(r-1), \gamma_r, D_-(r+1), \ldots, D_-(n)).$$

Then the twisted factorization at $\hat{\lambda}$ (with twist index $r$) is given by

$$LDL^T - \hat{\lambda} I = N_r \Delta_r N_r^T \qquad (13)$$

(note that the double factorizations in Eq. (9) are special cases of (13), the forward factorization $L_+D_+L_+^T$ corresponds to $r = n$, and the backward factorization $U_-D_-U_-^T$ to $r = 1$). When we compute the FP vector of the singleton $\hat{\lambda}$, we choose $r$ according to Eq. (11).

Hence, the solution of (7) is equivalent to solving

$$N_r^T v_r = e_r, \quad v_r(r) = 1. \qquad (14)$$

The advantage of using this equation is that $v_r$ can be computed solely with multiplications; no additions or subtractions are necessary. By Eq. (14), we obtain

$$v_r(i) = -L_+(i) v_r(i+1), \quad i = r-1, \ldots, 1, \qquad (15)$$

$$v_r(i+1) = -U_-(i) v_r(i), \quad i = r, \ldots, n-1. \qquad (16)$$

This feature permits error analysis of the computed FP vector in Dhillon and Parlett [2004b].

For an illustration of how the MRRR algorithm proceeds for a given matrix, in particular, the construction of the representation tree, we refer to the examples in Dhillon and Parlett [2004b], and Dhillon et al. [2005], and to Marques et al. [2005] for the subset case.

**Algorithm 3.** Outline of STEGR: Given a real symmetric tridiagonal matrix $T$, compute its eigenvalues and optionally eigenvectors

```
(A1) Data preprocessing and parameter error checking.
(A2) Determine the unreduced blocks, their root representations, and their
eigenvalues.
if eigenpairs are desired then
    (A3) For each eigenvalue compute its corresponding eigenvector.
end if
(A4) Postprocessing. Return the computed eigenvalues (and eigenvectors)
ordered from smallest to largest eigenvalue.
```

## 3. THE MAIN SUBROUTINE STEGR

STEGR is the main LAPACK driver and the interface for users who wish to use the MRRR algorithm. The goal of this section is to present an overview of the code. In Section 3.1, we give a simplified outline of the code, and a more detailed one is given in Section 3.2. Detailed descriptions of all computational and auxiliary subroutines called by STEGR are given in later sections.

The new version STEGR will be an important part of the next release of LAPACK [Anderson et al. 1999]. It is going to replace the preliminary version that is in the current 3.0 release of LAPACK. In particular, the eigenvalue driver routines, like SYEVR, will make extensive use of STEGR.

### 3.1 Principal Algorithmic Structure and Functionalities

For an overview of the algorithmic structure, see Algorithm 3. A more detailed description is given in Algorithm 4 in Section 3.1. The two major components, eigenvalue **(A2)** and eigenvector computation **(A3)**, both require $\mathcal{O}(n^2)$ work for the full set of eigenpairs.

For the illustration of the functionalities and code dependencies, see Figure 1.

### 3.2 Detailed Algorithmic Structure and Functionalities

We now present a detailed description of the algorithmic structure in Algorithm 4. Figure 2 illustrates the functionalities and code dependencies, expanding on Figure 1.

### 3.3 Governing Parameters and Workspace Requirements

STEGR has one important parameter:

—*The precision to which the eigenvalues are computed initially*: If eigenvectors are not desired, then eigenvalues are computed to full accuracy in LARRE. If eigenvectors have to be computed, the initial approximations will be refined in LARRV. In this case, the precision required in LARRE can be lower. For a subset of eigenpairs, LARRE uses bisection to compute the desired eigenvalues to limited precision. However, if the proportion of desired eigenpairs is high enough or the full spectrum has to be computed, then all eigenvalues are computed by the more efficient dqds algorithm and unwanted ones are discarded.

**Algorithm 4.** Details of STEGR: Given a real symmetric tridiagonal matrix $T$, compute desired eigenvalues and optionally, eigenvectors. A subset of the spectrum can be specified by either an index set IL : IU or interval [VL, VU]

```
(A1) Data preprocessing and parameter error checking:
Scale $T$ to the allowable range if necessary.
Check parameters and workspace.
(In particular, if eigenvalues from a range [VL, VU] have to be computed,
compute the number of eigenvalues in that interval.)
(A2) Find the unreduced blocks of $T$, their root representations, and their
eigenvalues:
if the matrix $T$ defines its eigenvalues to high relative accuracy then
  Enable relative splitting criterion that respects relative accuracy.
else
  Enable absolute splitting criterion only based on $\|T\|$.
end if
For each unreduced block, compute a root representation and its eigenvalues.
  if eigenpairs are desired then
  (A3) For each eigenvalue compute its corresponding eigenvector and support:
  Build the representation tree. Find suitable RRRs so that all eigenvalues
  become singletons.
  for each eigenvalue with a large relative gap do
    Use the simplified Algorithm 1 to compute the eigenvector.
  end for
end if
  (A4) Postprocessing. Return the computed eigenvalues (and eigenvectors):
Ensure that all eigenvalues are consistent with the original matrix $T$, shift
them if necessary.
if the matrix $T$ defines its eigenvalues to high relative accuracy then
  Refine the computed eigenvalues through bisection on $T$.
end if
if the matrix $T$ has been scaled then
  Undo the scaling of the computed eigenvalues.
end if
if the matrix $T$ was split into sub-blocks then
  Arrange the eigenvalues (previously stored by blocks) in increasing
order, likewise the eigenvectors.
end if
```

### 3.4 Workspace Requirements

STEGR uses two different kinds of workspace: real and integer. In the rest of this article, WORK denotes the real and IWORK the integer workspace. The matrix size is denoted by $N$.

STEGR partitions the available workspace into two segments: One part is used to store the persistent data that is available during the entire computation, and the second part is reused by different subroutines. The persistent data is computed by LARRE and used (and, if necessary, refined) by LARRV.
Real persistent data includes

—the Gersgorin intervals (size $2N$),

—uncertainty bounds for each computed eigenvalue (size $N$),

—the separation of each eigenvalue from its right neighbor (size $N$).

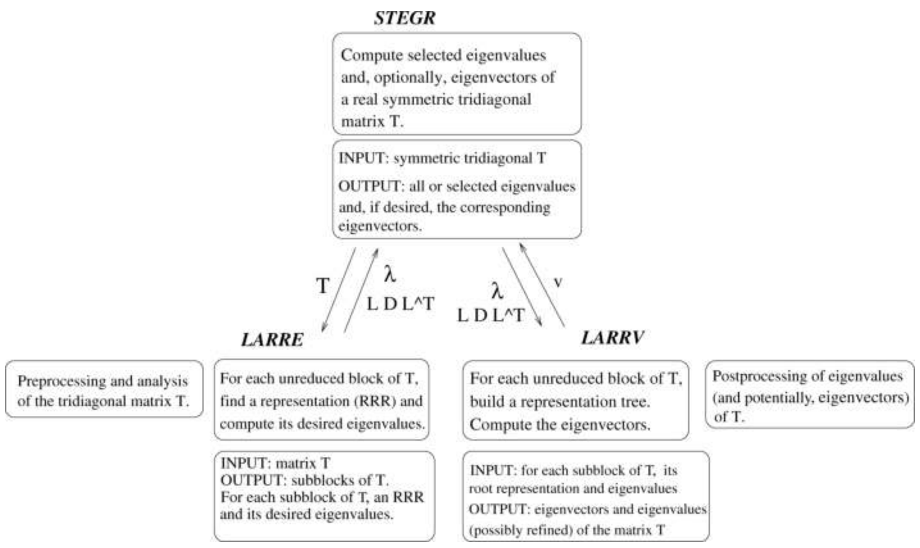

STEGR
Compute selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T.
INPUT: symmetric tridiagonal T
OUTPUT: all or selected eigenvalues and, if desired, the corresponding eigenvectors.
T
λ
L D L^T
λ
L D L^T
v
LARRE
LARRV
Preprocessing and analysis of the tridiagonal matrix T.
For each unreduced block of T, find a representation (RRR) and compute its desired eigenvalues.
INPUT: matrix T
OUTPUT: subblocks of T.
For each subblock of T, an RRR and its desired eigenvalues.
For each unreduced block of T, build a representation tree. Compute the eigenvectors.
INPUT: for each subblock of T, its root representation and eigenvalues
OUTPUT: eigenvectors and eigenvalues (possibly refined) of the matrix T
Postprocessing of eigenvalues (and potentially, eigenvectors) of T.

Fig. 1. Principal functionalities of STEGR. The subroutine LARRE computes the root representation and eigenvalues to suitable accuracy, while LARRV computes the representation tree and eigenvectors.

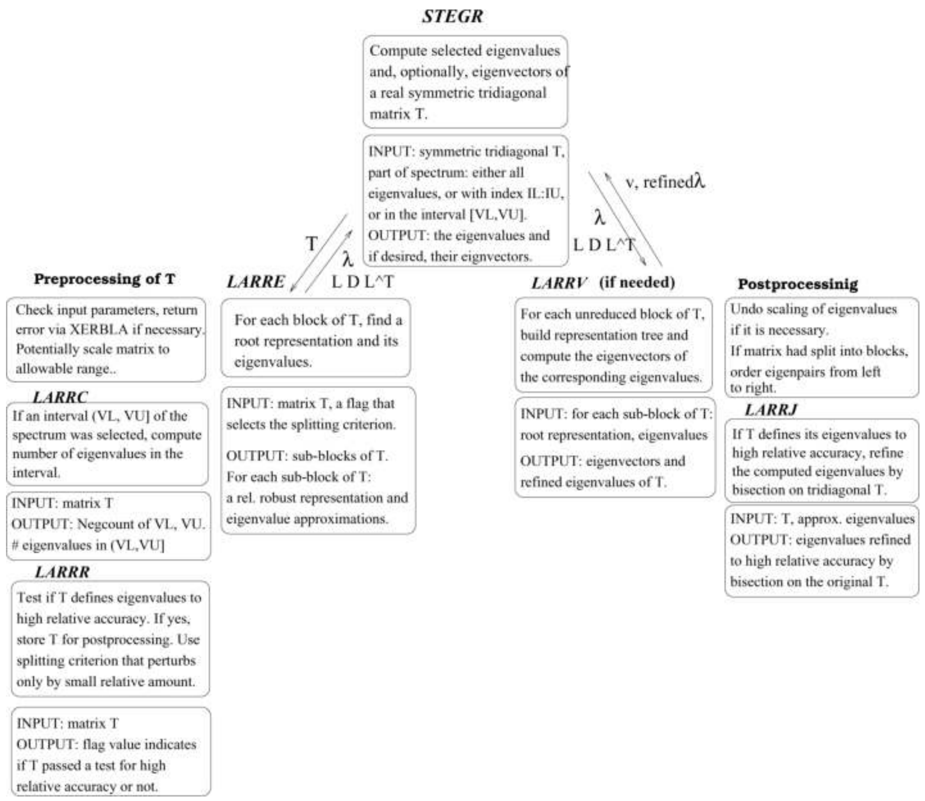

STEGR
Compute selected eigenvalues and, optionally, eigenvectors of a real symmetric tridiagonal matrix T.
INPUT: symmetric tridiagonal T, part of spectrum: either all eigenvalues, or with index IL:IU, or in the interval [VL,VU].
OUTPUT: the eigenvalues and if desired, their eignvectors.
T
λ
L D L^T
λ
L D L^T
v, refinedλ
Preprocessing of T
Check input parameters, return error via XERBLA if necessary. Potentially scale matrix to allowable range..
LARRC
If an interval (VL, VU] of the spectrum was selected, compute number of eigenvalues in the interval.
INPUT: matrix T
OUTPUT: Negcount of VL, VU. # eigenvalues in (VL,VU]
LARRR
Test if T defines eigenvalues to high relative accuracy. If yes, store T for postprocessing. Use splitting criterion that perturbs only by small relative amount.
INPUT: matrix T
OUTPUT: flag value indicates if T passed a test for high relative accuracy or not.
LARRE
For each block of T, find a root representation and its eigenvalues.
INPUT: matrix T, a flag that selects the splitting criterion.
OUTPUT: sub-blocks of T. For each sub-block of T: a rel. robust representation and eigenvalue approximations.
LARRV (if needed)
For each unreduced block of T, build representation tree and compute the eigenvectors of the corresponding eigenvalues.
INPUT: for each sub-block of T: root representation, eigenvalues
OUTPUT: eigenvectors and refined eigenvalues of T.
Postprocessinig
Undo scaling of eigenvalues if it is necessary.
If matrix had split into blocks, order eigenpairs from left to right.
LARRJ
If T defines its eigenvalues to high relative accuracy, refine the computed eigenvalues by bisection on tridiagonal T.
INPUT: T, approx. eigenvalues
OUTPUT: eigenvalues refined to high relative accuracy by bisection on the original T.

Fig. 2. Detailed functionalities of STEGR.

Integer persistent data includes

—the splitting indices at which the matrix $T$ breaks up into blocks (size $N$),
—the block number of each eigenvalue (size $N$),
—the local indices of the eigenvalues within each block (size $N$).

The remaining real and integer workspace is shared and reused by the subroutines LARRR, LARRE, LARRV, and potentially, LARRJ (see Figure 2). Note that the required workspace is linear in the matrix dimension.

### 3.5 A Note on Design Alternatives

The current code design follows the LAPACK style of first computing all desired eigenvalues and afterwards, the corresponding eigenvectors.

There is an alternative way of organizing the computation. After splitting, the eigenpairs of each matrix sub-block can be computed independently from those of the other sub-blocks. Thus, it would be possible to compute all eigenpairs of a given block first, before moving on to the next.

## 4. SUBROUTINE LARRE

LARRE is one of the two major computational subroutines used by STEGR. For each unreduced block, LARRE computes its root representation and the desired eigenvalues by either bisection or dqds.

### 4.1 Principal Algorithmic Structure and Functionalities

For the algorithmic structure, see Algorithm 5. The functionalities are illustrated in Figure 3.

**Algorithm 5.** Principal algorithmic structure of LARRE: Given a real symmetric tridiagonal matrix $T$, find its unreduced sub-blocks. For each sub-block, compute its root representation and the desired eigenvalues. Subsets of the spectrum can be designated either by indices IL : IU or by an interval [VL, VU]

```
Record Gersgorin intervals.
Record splitting indices, set negligible off-diagonals to zero.
if only a subset of the spectrum is desired then
  if the desired subset is given by its indices IL : IU then
    Find a corresponding interval [VL, VU].
  else
    Find the (global) indices IL : IU corresponding to the given interval [VL, VU].
    end if
    Compute crude approximations to the desired eigenvalues by bisection.
end if
for each unreduced block of $T$ do
  (B1) Decide whether dqds or bisection will be used for the eigenvalue
computation.
  (B2) Choose a shift for the root representation.
  (B3) Compute the root RRR.
  (B4) Perturb the root RRR by a few ulps.
  (B5) Compute desired eigenvalues of the RRR by bisection or dqds.
end for
```

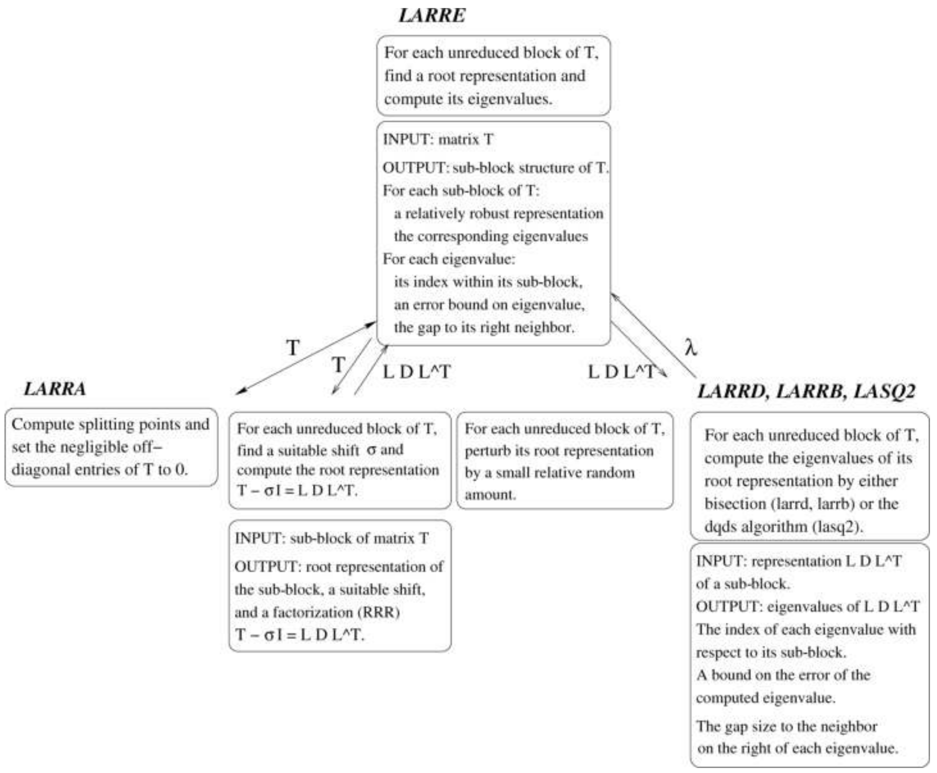

LARRE
For each unreduced block of T, find a root representation and compute its eigenvalues.
INPUT: matrix T
OUTPUT: sub-block structure of T. For each sub-block of T: a relatively robust representation the corresponding eigenvalues For each eigenvalue: its index within its sub-block, an error bound on eigenvalue, the gap to its right neighbor.
T
T
L D L^T
L D L^T
λ
LARRA
Compute splitting points and set the negligible off-diagonal entries of T to 0.
For each unreduced block of T, find a suitable shift σ and compute the root representation T − σI = L D L^T.
INPUT: sub-block of matrix T
OUTPUT: root representation of the sub-block, a suitable shift, and a factorization (RRR) T − σI = L D L^T.
For each unreduced block of T, perturb its root representation by a small relative random amount.
LARRD, LARRB, LASQ2
For each unreduced block of T, compute the eigenvalues of its root representation by either bisection (larrd, larrb) or the dqds algorithm (lasq2).
INPUT: representation L D L^T of a sub-block.
OUTPUT: eigenvalues of L D L^T The index of each eigenvalue with respect to its sub-block. A bound on the error of the computed eigenvalue. The gap size to the neighbor on the right of each eigenvalue.

Fig. 3. Principal functionalities of LARRE.

## 4.2 Detailed Algorithmic Structure and Functionalities

In this section, we expand our description of the central part of Algorithm 5. Algorithm 6 describes the algorithm once an unreduced block of $T$ has been found. It shows how the root representation is chosen. It also gives details on the computation of the (local) eigenvalues, as well as their indices and block numbers, by either bisection or dqds. Note that the dqds algorithm requires a definite RRR, whereas bisection does not.

Figure 4 expands on Figure 3, illustrating the functionalities and dependencies on other subroutines.

## 4.3 Governing Parameters

—*Initial precision of bisection*: Dqds always computes the eigenvalues to full accuracy. However, if bisection is used, it can be more efficient not to compute the eigenvalues to full precision, since they can be improved through Rayleigh quotient correction in LARRV.

—*Maximum allowable element growth for the root factorization*: When dqds is used, the root representation is definite and thus an RRR for all eigenvalues. An $LDL^T$ factorization for a shift inside the spectrum is an RRR for all eigenvalues if the element growth is small enough (see Parlett and Dhillon [2000] for details).

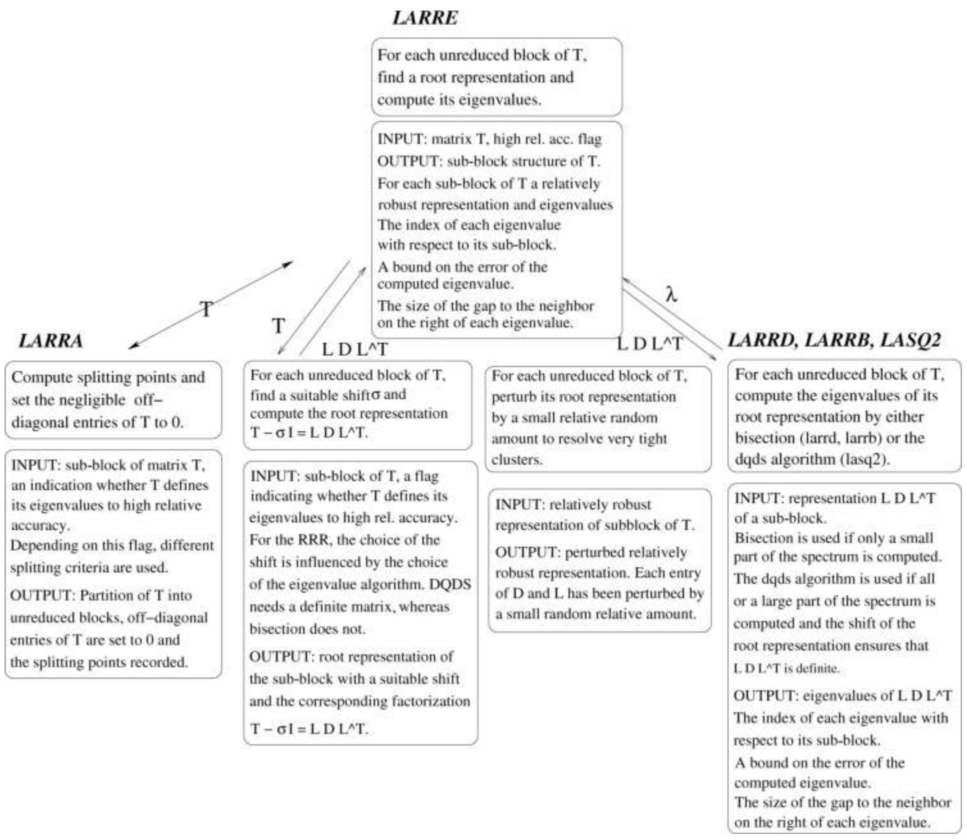

LARRE
For each unreduced block of T, find a root representation and compute its eigenvalues.
INPUT: matrix T, high rel. acc. flag
OUTPUT: sub-block structure of T.
For each sub-block of T a relatively robust representation and eigenvalues
The index of each eigenvalue with respect to its sub-block.
A bound on the error of the computed eigenvalue.
The size of the gap to the neighbor on the right of each eigenvalue.
T
T
L D L^T
L D L^T
λ
LARRA
Compute splitting points and set the negligible off–diagonal entries of T to 0.
INPUT: sub-block of matrix T, an indication whether T defines its eigenvalues to high relative accuracy.
Depending on this flag, different splitting criteria are used.
OUTPUT: Partition of T into unreduced blocks, off–diagonal entries of T are set to 0 and the splitting points recorded.
For each unreduced block of T, find a suitable shift σ and compute the root representation T − σI = L D L^T.
INPUT: sub-block of T, a flag indicating whether T defines its eigenvalues to high rel. accuracy. For the RRR, the choice of the shift is influenced by the choice of the eigenvalue algorithm. DQDS needs a definite matrix, whereas bisection does not.
OUTPUT: root representation of the sub-block with a suitable shift and the corresponding factorization
T − σI = L D L^T.
For each unreduced block of T, perturb its root representation by a small relative random amount to resolve very tight clusters.
INPUT: relatively robust representation of subblock of T.
OUTPUT: perturbed relatively robust representation. Each entry of D and L has been perturbed by a small random relative amount.
LARRD, LARRB, LASQ2
For each unreduced block of T, compute the eigenvalues of its root representation by either bisection (larrd, larrb) or the dqds algorithm (lasq2).
INPUT: representation L D L^T of a sub-block.
Bisection is used if only a small part of the spectrum is computed.
The dqds algorithm is used if all or a large part of the spectrum is computed and the shift of the root representation ensures that L D L^T is definite.
OUTPUT: eigenvalues of L D L^T
The index of each eigenvalue with respect to its sub-block.
A bound on the error of the computed eigenvalue.
The size of the gap to the neighbor on the right of each eigenvalue.

Fig. 4. Detailed functionalities of LARRE.

—*The threshold to switch between bisection and dqds*: If a large enough proportion of eigenvalues is desired, the code uses dqds to find all eigenvalues and discards those that are unwanted.
—*The size of the random perturbation applied to the root representation*: To break up very tight clusters, tiny random perturbations are made to the root representation (see Dhillon et al. [2005] for details).

### 4.4 Workspace Requirements

LARRE requires $6N$ real workspace that is shared and reused by the subroutines for bisection (LARRD, LARRB) and the dqds algorithm (LASQ2). The storage requirements are dictated by dqds for which $6N$ storage is needed in total. Integer workspace IWORK requirements are dominated by LARRD.

## 5. SUBROUTINE LARRV

LARRV is the second major computational subroutine used by STEGR. It computes, from the output generated by LARRE, the desired eigenvectors.

**Algorithm 6.** Detailed algorithmic structure of the central part of LARRE: For each unreduced sub-block, compute an RRR and its eigenvalues

```
for each unreduced block of $T$ do
  Deal with $1 \times 1$ block and proceed to next block immediately.
  Find spectral diameter of the current block.

  (B1) Decide whether dqds or bisection will be used for the eigenvalue computation:
  if only a subset of the spectrum is desired then
    Count the number of desired eigenvalues in the block.
    If a big percentage of the spectrum is desired, dqds will be used, otherwise bisection is invoked.
  else
    Dqds is used for the full spectrum.
  end if

  (B2) Choose a shift for the root representation:
  if the dqds algorithm is to be used then
    Compute the extremal eigenvalues accurately to yield bounds $VL$ and $VU$ on the desired eigenvalues.
  else
    Find adequate lower and upper bounds $VL$ and $VU$ on extreme desired eigenvalues as initial shifts.
  end if
  Compute the Sturm counts at the 0.25 and 0.75 points of the desired interval [VL, VU].
  Choose shift $\sigma$ for the root RRR at the most crowded end.

  (B3) Compute the root RRR:
  while no factorization has been accepted as an RRR do
    Compute factorization $T - \sigma I = LDL^T$, accept if element growth is less than the tolerance.
    if the dqds algorithm is to be used then
      Accept if the factorization is positive or negative definite.
    else
      Change shift if factorization is not accepted.
    end if
  end while

  (B4) Perturb the root RRR by a few ulps:
  Perturb each element of $D$ and $L$ by a tiny random relative amount.
  Store the representation in the corresponding block of $T$, the matrix $D$ in the diagonal part and the nontrivial entries of $L$ in the off-diagonal part.

  (B5) Compute desired eigenvalues of the RRR by bisection or dqds:
  if Dqds is used then
    Compute all eigenvalues, discard unwanted ones.
    Record uncertainties, gaps, local indices and block numbers.
  else
    Change previously computed approximate eigenvalues of $T$ by shift $\sigma$ (of root RRR).
    Refine approximate eigenvalues, uncertainties, and gaps by bisection to a specified accuracy.
    Record local indices and block numbers.
  end if
end for
```

**Algorithm 7.** Principal algorithmic structure of LARRV: Build the representation tree and refine the eigenvalues. Compute eigenvectors of singletons by inverse iteration with a twisted factorization

```
for each unreduced block of $T$ do
  if the block does not contain any desired eigenvalues then
    skip block.
  end if
  (C1) Build representation tree from the root down one level at a time as follows:
  while there are still eigenvectors of this block to be computed do
    (C2) Process nodes of current level of representation tree as follows:
    for each node do
      (C3) Retrieve the RRR of the node and refine its eigenvalues, if needed.
      while the node is not entirely processed do
        (C4)Identify the next child.
        if a child is not a singleton then
          (C5) Compute an RRR for the child and store it.
        else
          (C6) Refine the eigenvalue and compute the corresponding eigenvector.
        end if
      end while
    end for
  end while
end for
```

## 5.1 Principal Algorithmic Structure and Functionalities

See Algorithm 7 for the structure, and Figure 5 for an illustration of the functionalities.

## 5.2 Detailed Algorithmic Structure and Functionalities

In this section, we expand our description of the central part of Algorithm 7. Algorithm 8 describes the computations for a sub-block. Note that we use the word “eigenvector” for an accepted FP vector: Algorithmically, we compute an FP vector which has the numerical properties of an eigenvector when the corresponding eigenvalue is approximated to high relative accuracy see Section (2).

The functionalities and code dependencies of LARRV are illustrated in Figure 6.

## 5.3 Governing Parameters

—*Precision of bisection when refining the eigenvalues of a child*: Bisection is used to refine the local eigenvalues of a child after they have been translated by the new incremental shift. Shifting generally leads to a loss of the leading digits that the eigenvalues have in common. In general, it is not necessary to compute the eigenvalues to full precision by bisection, since Rayleigh quotient correction is available when computing the vector. However, a minimal number of leading figures have to be accurate so that relative gaps can be reliably identified.

—*The number of Rayleigh quotient correction steps*: Once a singleton is encountered, its eigenvalue with respect to the current RRR must have maximal

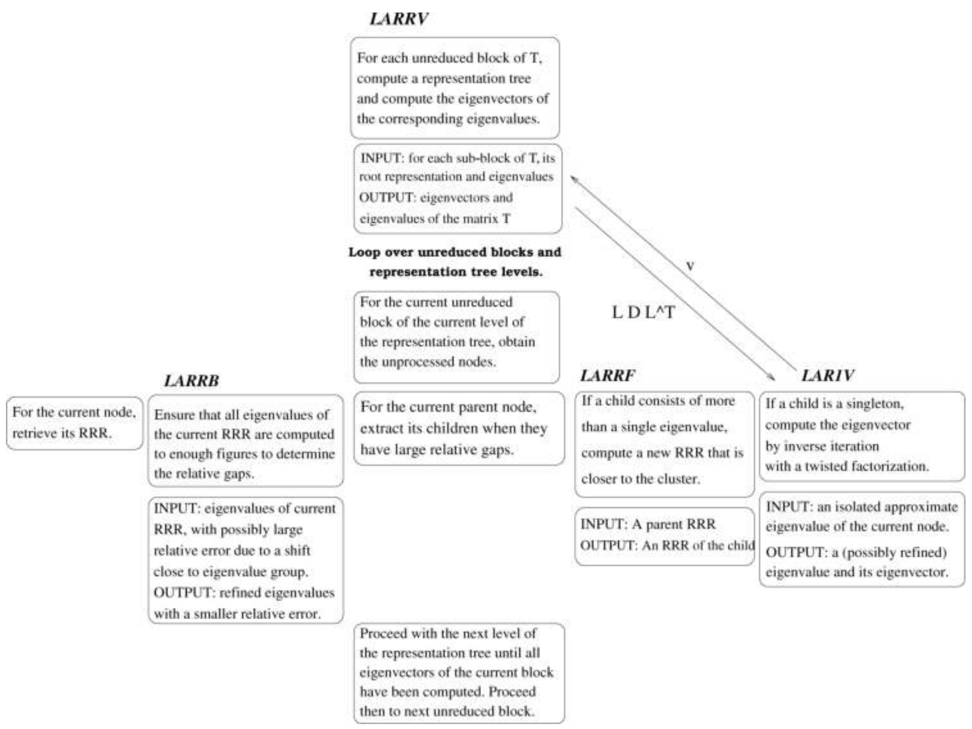

LARRV
For each unreduced block of T, compute a representation tree and compute the eigenvectors of the corresponding eigenvalues.
INPUT: for each sub-block of T, its root representation and eigenvalues
OUTPUT: eigenvectors and eigenvalues of the matrix T
Loop over unreduced blocks and representation tree levels.
For the current unreduced block of the current level of the representation tree, obtain the unprocessed nodes.
For the current parent node, extract its children when they have large relative gaps.
Proceed with the next level of the representation tree until all eigenvectors of the current block have been computed. Proceed then to next unreduced block.
v
L D L^T
LARRB
For the current node, retrieve its RRR.
Ensure that all eigenvalues of the current RRR are computed to enough figures to determine the relative gaps.
INPUT: eigenvalues of current RRR, with possibly large relative error due to a shift close to eigenvalue group.
OUTPUT: refined eigenvalues with a smaller relative error.
LARRF
If a child consists of more than a single eigenvalue, compute a new RRR that is closer to the cluster.
INPUT: A parent RRR
OUTPUT: An RRR of the child
LARIV
If a child is a singleton, compute the eigenvector by inverse iteration with a twisted factorization.
INPUT: an isolated approximate eigenvalue of the current node.
OUTPUT: a (possibly refined) eigenvalue and its eigenvector.

Fig. 5. Principal functionalities of LARRV.

accuracy (if the the separation is small; accuracy requirements can be relaxed if the separation is large). Rayleigh quotient correction is more efficient in the asymptotic region than bisection, but need not be reliable within clusters of close eigenvalues. Since the asymptotic region is not known *a priori*, the algorithm tries to improve the approximation by a number of Rayleigh quotient correction steps. Bisection can guard the Rayleigh quotient correction against convergence to the neighboring eigenvalue.

—*The minimum relative gap between children*: The error in the FP vector is proportional to the reciprocal of the relative gap between eigenvalues. So, we insist on the relative gap being larger than a threshold. If the threshold is increased, then more representations are used and the representation tree will become deeper.

## 5.4 Workspace Requirements

LARRV partitions the available workspace into persistent storage and storage that is shared and reused by called subroutines.

Real persistent data includes:

—the local eigenvalues, that is, the eigenvalues with respect to the current RRR (size $N$); and

—the quantities $l_i d_i$ and $l_i^2 d_i$ for the current representation (size $2N$), needed for dqds transformations.

**Algorithm 8.** Detailed algorithmic structure of the central part of LARRV: Given the root representation and its eigenvalues, compute the corresponding eigenvectors

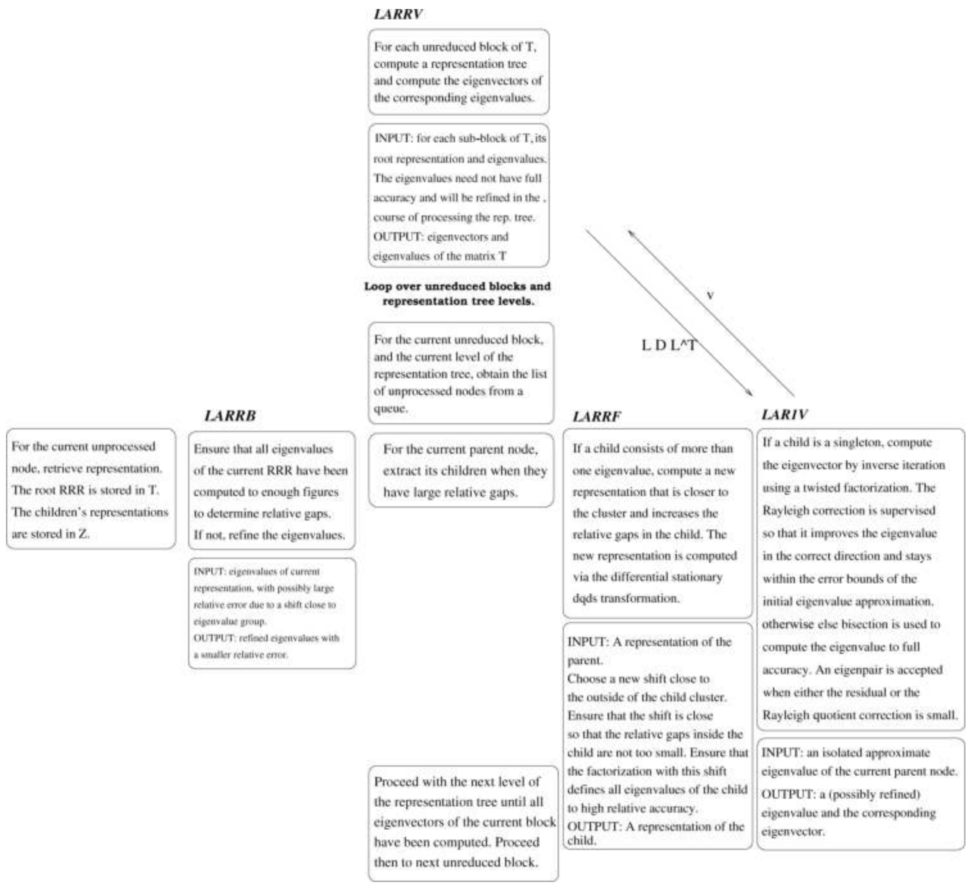
```
(C1) Build representation tree from the root down one level at a time as follows:
Retrieve root representation.
Initialize queue of unprocessed nodes with root.
while there are still eigenvectors of this block to be computed do

  (C2) Process nodes of current level of representation tree:
  The unprocessed nodes are stored in a queue.
  Nodes of the current level are dequeued, children (except singletons), as detected,
  are enqueued.
  This corresponds to a breadth-first construction of the representation tree.
  for each unprocessed node of the current level do

    (C3) Retrieve the RRR of the node and refine its eigenvalues:
    Retrieve the RRR of the node (stored in $T$ if root, otherwise in the unused part
    of the eigenvector matrix $Z$).
    Refine the local eigenvalues to enough figures so that the relative gaps can be
    assessed accurately enough and singletons can be detected.
    while the node is not entirely processed do

      (C4) Identify the next child:
      Starting with the leftmost unprocessed eigenvalue of the node, proceed
      to the right to identify the next child, that is, until a large relative gap is detected.
      if a child is not a singleton then

        (C5) Compute an RRR for the child and store it:
        Store RRR in $Z$ at the index corresponding to the first and second
        eigenvalues of the node.
        Add the child node to the queue to be processed on the next level of the tree.
      else
        (C6) Refine the eigenvalue (if needed) and compute the corresponding
        eigenvector:
          while the FP vector is not accepted do
            Compute FP vector from one step of inverse iteration with a twisted
            factorization.
            Record Rayleigh Quotient correction and residual norm.
            if Rayleigh quotient correction or residual norm is small enough then
              Signal convergence.
              Store refined eigenvalue with appropriate shift.
            else
              if Rayleigh quotient correction is accepted and maximum allowed
              number of steps is not exceeded then
                Improve eigenvalue approximation by Rayleigh quotient correction.
              else
                Refine eigenvalue to full accuracy by bisection
              end if
          end if
        end while
        Scale computed vector to unit length and store it.
      end if
    end while
  end for
end while
```



For each unreduced block of T, compute a representation tree and compute the eigenvectors of the corresponding eigenvalues.

INPUT: for each sub-block of T, its root representation and eigenvalues. The eigenvalues need not have full accuracy and will be refined in the , course of processing the rep. tree. OUTPUT: eigenvectors and eigenvalues of the matrix T

**Loop over unreduced blocks and representation tree levels.**

For the current unreduced block, and the current level of the representation tree, obtain the list of unprocessed nodes from a queue.

For the current unprocessed node, retrieve representation. The root RRR is stored in T. The children's representations are stored in Z.

Ensure that all eigenvalues of the current RRR have been computed to enough figures to determine relative gaps. If not, refine the eigenvalues.

INPUT: eigenvalues of current representation, with possibly large relative error due to a shift close to eigenvalue group. OUTPUT: refined eigenvalues with a smaller relative error.

For the current parent node, extract its children when they have large relative gaps.

If a child consists of more than one eigenvalue, compute a new representation that is closer to the cluster and increases the relative gaps in the child. The new representation is computed via the differential stationary dqds transformation.

INPUT: A representation of the parent. Choose a new shift close to the outside of the child cluster. Ensure that the shift is close so that the relative gaps inside the child are not too small. Ensure that the factorization with this shift defines all eigenvalues of the child to high relative accuracy. OUTPUT: A representation of the child.

If a child is a singleton, compute the eigenvector by inverse iteration using a twisted factorization. The Rayleigh correction is supervised so that it improves the eigenvalue in the correct direction and stays within the error bounds of the initial eigenvalue approximation, otherwise else bisection is used to compute the eigenvalue to full accuracy. An eigenpair is accepted when either the residual or the Rayleigh quotient correction is small.

INPUT: an isolated approximate eigenvalue of the current parent node. OUTPUT: a (possibly refined) eigenvalue and the corresponding eigenvector.

Proceed with the next level of the representation tree until all eigenvectors of the current block have been computed. Proceed then to next unreduced block.

Fig. 6. Detailed functionalities of LARRV.

Integer persistent data includes:

—the twist index for each singleton (size $N$); and
—the queue for nodes of the current level of the representation tree and parent level (size $2N$).

The remaining real and integer workspace is shared and reused by LAR1V, LARRF, and LARRB.

Furthermore, the matrix of the eigenvectors $Z$ is used as intermediate storage for representations.

## 6. SUBROUTINE LARRF

LARRF is an auxiliary subroutine called by LARRV to compute a new RRR for a child from the representation of its parent.

### 6.1 Principal Algorithmic Structure

For the algorithmic structure, see Algorithm 9.

**Algorithm 9.** Principal algorithmic structure of LARRF: Compute an RRR of a child from the RRR of a parent

```
Given a node with representation $LDL^T$ and (local) eigenvalues $\lambda_i \in [l, r]$
Set $\sigma_l = l$ and $\sigma_r = r$
while No RRR has been found do
  Compute factorization at left end: $\hat{L}\hat{D}\hat{L}^T = LDL^T - \sigma_l I$.
  if factorization has little element growth then
    Accept factorization as RRR and exit.
  end if
  Compute factorization at right end: $\check{L}\check{D}\check{L}^T = LDL^T - \sigma_r I$.
  if factorization has little element growth then
    Accept factorization as RRR and exit.
  end if
  (D1) Inspect the factorizations at each end of the child.
  if (D2) the better of the two factorizations passes the refined test for an RRR then
    Accept that factorization as RRR and exit.
  else
    if maximum number of trials not exceeded then
      (D3) Save characteristics of better factorization for later inspection.
      (D4) Choose new shifts by backing off from the cluster ends by a small
amount.
    else
      (D5) Inspect all computed factorizations and choose the one with smallest
element growth as RRR.
    end if
  end if
end while
```

## 6.2 Detailed Algorithmic Structure

In this section, we describe in detail the algorithm for finding an RRR of a child. From the RRR of the parent, the algorithm computes two new factorizations, with shifts $\sigma_l = l$ and $\sigma_r = r$, using the differential stationary qds transform. If little element growth occurs in a factorization, it can be accepted as an RRR. Even with (moderately) large element growth, a factorization can still be an RRR for the desired subset. A heuristic test is employed to check. If both factorizations are unsatisfactory, the shifts $\sigma_l$ and $\sigma_r$ are changed and new factorizations are computed. For any factorization without a NaN, we save the shift and the element growth to select the factorization with smallest growth as backup safeguard (see Algorithm 10).

## 6.3 Governing Parameters

—*The maximum allowable element growth to accept a factorization as RRR immediately*: A factorization with essentially no element growth defines all its eigenvalues to high relative accuracy. For this reason, such a factorization can immediately be accepted as an RRR. However, there is the danger of selecting a factorization prematurely, without looking at others if there is element growth.

—*The amount by which to back off from the cluster ends*: As mentioned before, we have to carefully select the amount by which to back off. The goal of backing off is to reduce element growth, while preserving large relative gaps.

**Algorithm 10.** Detailed algorithmic structure of the central part of LARRF. Finding a suitable factorization as RRR for a child when element growth occurs

```
(D1) Inspect the factorizations at each end of the child:
if no NaN has occurred then
  choose the one with smaller element growth as the better one.
else
  Disregard any factorization with a NaN completely.
  If there is one factorization without NaN, select it as the better one.
  Ignore the following RRR test and go directly to (D2a).
end if

(D2) the factorization passes the refined test for an RRR:
Compute an approximate eigenvector $v$ from product of entries of bidiagonal
factor $L$ at the poorer end of cluster.
RRR test: Compute relative condition number $\|Dv\|_\infty$, $D$ from the better end.
if the RRR test is passed then
    Accept that factorization as RRR and exit.
end if

if (D2a) maximum number of trials not exceeded then
    (D3) Save characteristics of better factorization for later inspection:
    Save the shift and the element growth.

    (D4) Choose new shifts by backing off from the cluster ends by a small amount:
    Choose $\delta_l$ and $\delta_r$ large enough to decrease the element growth and small enough
  to preserve large relative gaps.
    $\sigma_l = \sigma_l - \delta_l$, $\sigma_r = \sigma_r - \delta_r$.
    Double $\delta_l$ and $\delta_r$.
else

    (D5) Inspect all computed factorizations and choose the best:
    From all factorizations without NaNs, take the one with the smallest
  element growth.
end if
```

If a shift near the end of a cluster coincides with a Ritz value (an eigenvalue of a principal submatrix), then the factorization breaks down. The code tries a new shift by backing away from the end. However, by selecting the shift of the representation farther away from the eigenvalues, the relative gaps inside the child decrease and thus, the algorithm has to do more work to compute orthogonal eigenvectors. Backing off too far even might, in an extreme case, make the algorithm fail to find a relative gap above the threshold. On the other hand, backing off too little will not reduce the element growth. A natural choice is to back away by the distance to the next eigenvalue (in the child) or the average gap inside the cluster. More research is needed on this subject.

—*The number of times the algorithm should back away from the ends of a child.*

## 7. SUBROUTINE LAR1V

LAR1V is an auxiliary subroutine called by LARRV to compute an FP vector of a singleton. This is essentially Algorithm Getvec from Dhillon and Parlett

**Algorithm 11.** Principal algorithmic structure of LAR1V: Compute an FP vector for a relatively isolated eigenvalue approximation from a twisted factorization

```
Given a parent RRR $LDL^T$ and a relatively isolated approximate eigenvalue $\hat{\lambda}$.
(E1) Compute forward and backward factorization of $LDL^T - \hat{\lambda}I$.
if the location of the twist index has not been previously chosen then
    (E2) Compute $\gamma_k, k = 1, \ldots, n$ and find the twist index $r = \arg\min |\gamma_k|$.
end if
Form the twisted factorization $LDL^T - \hat{\lambda}I = N_r \Delta_r N_r^T$ using the data from (E1).
(E3) Compute the FP vector by solving $N_r^T v = e_r (\Leftrightarrow N_r \Delta_r N_r^T v = \gamma_r e_r), v(r) = 1$.
```

[2004b]. Apart from the FP vector $v_r$, it also returns the twist index $r$ from the twisted factorization used, the Sturm count (or negcount, the number of pivots smaller than $\hat{\lambda}$), and $\gamma_r$. Here, $\gamma_r$ can be used to compute the residual of the FP vector and Rayleigh quotient correction $\gamma / \|v_r\|_2^2$ to $\hat{\lambda}$.

The optional Sturm count can be used to ensure the correct direction (sign) of the proposed Rayleigh quotient correction: If the proposed correction step has the correct sign and lies within the uncertainty interval, it is accepted. On the other hand, if the Sturm count indicates that the proposed correction step goes in the wrong direction or that the size of the correction step would take the new iterate outside the known uncertainty interval, the algorithm switches to bisection to compute the eigenvalue.

### 7.1 Principal Algorithmic Structure and Functionalities

In order to simplify the presentation, we only describe the main parts of the computation of the FP vector in Algorithm 11. For the computation of the Sturm count and a more detailed description, we refer to Section 7.2.

### 7.2 Detailed Algorithmic Structure and Functionalities

This section gives the details on how to compute an FP vector. Note that the computation of the twist index $r$ is only done once per eigenvalue and omitted if multiple Rayleigh corrections are applied to the same eigenvalue approximation. Furthermore, the computation of the Sturm count of $\hat{\lambda}$ can be omitted if $\hat{\lambda}$ is accurate. One detail is omitted in Algorithm 12: We must ensure that the Sturm count is only computed from factorizations without a NaN.

## 8. SUBROUTINE LARRA

It is a very common technique in eigensolvers to split a given matrix into unreduced blocks and then work in turn on each of the blocks. One major reason is efficiency: Since all solvers have a complexity that is superlinear with respect to the matrix size, smaller sub-blocks will improve performance with respect to storage and operations. There is a second reason for splitting in MRRR: The theory assumes that all eigenvalues are simple.

MRRR offers two different splitting strategies:

—An "absolute" splitting criterion that sets the off-diagonal value to zero whenever it is small compared to $\|T\|$.

**Algorithm 12.** Detailed algorithmic structure of LAR1V: Compute an FP vector for $\hat{\lambda}$ from a twisted factorization. Optionally, supply a Sturm count of $\hat{\lambda}$

```
(E1) Compute forward and backward factorization of $LDL^T - \hat{\lambda}I$:
Compute forward factorization $LDL^T - \hat{\lambda}I = L_+D_+L_+^T$ by the differential stationary qds transform.
Compute the Sturm count of $\hat{\lambda}$.
if the twist index $r$ has been previously chosen then
  Only compute the factorization (and the Sturm count) down to the index $r$.
end if
Compute backward factorization $LDL^T - \hat{\lambda}I = U_-D_-U_-^T$ by the differential progressive qds transform.
if the twist index $r$ has been previously chosen then
  Only compute the factorization (and the Sturm count) up to the index $r$.
end if

if the location of the twist index has not been previously chosen then
  (E2) Compute $\gamma_k, k = 1, \ldots, n$ and find the twist index $r = \arg\min |\gamma_k|$:
  $\gamma_k = s_k + p_k, k = 1, \ldots, n$. ($s_k$ arise in the forward, $p_k$ in the backward factorization.)
  Choose $r$ as the index of the minimum $|\gamma_r|$.
  If dstqds and dqds factorization have been stopped at index $r$, the sign of $\gamma_r$ completes the Sturm count.
end if
Form the twisted factorization $L_iD_iL_i^T - \hat{\lambda}I = N_r\Delta_rN_r^T$ using the data from (E1).

(E3) Compute the FP vector by solving $N_r^Tv = e_r (\Leftrightarrow N_r\Delta_rN_r^Tv = \gamma_re_r), v(r) = 1$:
Find the solution of $N_r\Delta_rN_r^Tv = \gamma_re_r$ by solving $N_r^Tv = e_r$:
Solve backward using $L_+$ with indices $r-1 : -1 : 1$ and forward using $U_-$ with indices $r+1 : n$.
```

—A "relative" splitting criterion that sets the off-diagonal value to zero whenever it is small compared to the geometric mean of its neighboring diagonal elements.

The relative splitting criterion is designed so that neglecting an off-diagonal is equivalent to tiny relative changes in the neighboring diagonal elements.

## 9. ROUTINES FOR COMPUTING EIGENVALUES

Bisection plays a central role when computing the eigenvalues to high relative accuracy. This is based on Sturm counts computed from the differential qds algorithm. Originally proposed by Rutishauser [1954, 1976], [Fernando and Parlett 1994; Parlett 1995] discovered the shifted differential variant that MRRR uses.

LARRB implements bisection for tridiagonal matrices in factored form $LDL^T$ and has two alternative criteria for stopping the refinement of an eigenvalue. Consider neighboring eigenvalue approximations $\hat{\lambda}_i$ and $\hat{\lambda}_{i+1}$ with uncertainties $\pm\delta\lambda_i$ and $\pm\delta\lambda_{i+1}$ so that the true $i$th eigenvalue lies in $[\hat{\lambda}_i - \delta\lambda_i, \hat{\lambda}_i + \delta\lambda_i]$ and the true $i+1$-th in $[\hat{\lambda}_{i+1} - \delta\lambda_{i+1}, \hat{\lambda}_{i+1} + \delta\lambda_{i+1}]$. Denote by $\hat{\lambda}_{i+1} - \delta\lambda_{i+1} - \hat{\lambda}_i - \delta\lambda_i$ the right gap of $\hat{\lambda}_i$. Then eigenvalue $\hat{\lambda}_i$ has converged if either of the following is true:

—The width of the uncertainty interval of $\hat{\lambda}_i$ is smaller than a threshold times its right gap. This means that the uncertainty in the eigenvalue cannot change the gap by more than a relative amount determined by the threshold.

—The width of the uncertainty interval of $\hat{\lambda}_i$ is smaller than a threshold times the maximum (in absolute value) of the two interval boundaries. This implies that the eigenvalue is correct to a number of digits determined by the threshold.

LARRB is used each time a child has been computed from its parent in order to recover lost figures in precision due to shifting. Furthermore, it is used for the computation of singletons in the case that Rayleigh quotient correction does not converge to the desired eigenvalue. It has an input parameter that allows a user to choose between different Sturm counts: The twist index is an integer between 1 and $n$; setting it to $n$ corresponds to a differential stationary qds transform, and setting it to 1 to a differential progressive qds transform, otherwise, it is a twisted factorization with twist index $r$.

LARRC does Sturm counts on the matrix $T$ at two fixed points $l$ and $r$ and returns the number of eigenvalues in [l,r]. It is used by STEGR to compute the needed workspace when the eigenvalues in an interval $[VL, VU]$ have to be computed. Another use is to find the most crowded end of the spectrum when computing the root representation of a block in LARRE.

LARRD implements bisection for tridiagonal matrices $T$ and has a single convergence criterion: the relative width of the convergence interval. LARRD is a variant of the LAPACK code STEBZ that has been adapted for STEGR. Apart from computing initial approximations of eigenvalues, it can also be used to compute the index of an eigenvalue in its block when the matrix splits.

LARRJ implements bisection for tridiagonal matrices $T$ and has a single convergence criterion: the relative width of the convergence interval. Its functionality is similar to LARRD, with the difference that it refines given initial intervals around the selected eigenvalues. It is used for postprocessing of the eigenvalues computed in LARRE and LARRV in case the user asks STEGR to compute eigenvalues to high relative accuracy and the matrix $T$ allows this (see Section (10)). LARRD is not easily adaptable to the task of refining eigenvalues from a given initial interval.

LASQ2 is the LAPACK implementation of the dqds algorithm [Parlett and Marques 2000] that internally computes the eigenvalues of a (positive or negative) definite $LDL^T$ factorization. For this reason, if dqds is used, the root representation has to be definite.

## 10. LARRR: TEST IF $T$ DEFINES ITS EIGENVALUES TO HIGH RELATIVE ACCURACY

STEGR aims to compute the eigenvalues of $T$ to high relative accuracy if the matrix deserves it. There are a number of sufficient criteria that $T$ can be tested on (see Barlow and Demmel [1990]). Currently, we test whether $T$ is scaled diagonally dominant (s.d.d.). For the algorithmic structure, see Algorithm 13.

**Algorithm 13.** LARRR: Test if $T$ is s.d.d. and thus defines its eigenvalues to high relative accuracy

```
S.D.D. test:
Compute matrix $\hat{T} = D^{-1/2} T D^{-1/2}$ with diagonal entries $\pm 1$.
if $|\hat{e}_i| + |\hat{e}_{i-1}| < 1$ , $\forall i$ then
  return TRUE
else
  return FALSE
t end if
```

## 11. SPECIAL TOPICS

In this article, we have shown how the theory of the MRRR algorithm translates into software. We have described the design and implementation of STEGR, which will be part of the next release of LAPACK. We have identified governing parameters of the algorithm and discussed their influence on accuracy and performance. There are a number of additional topics regarding current and future research that we address in this section.

### 11.1 Test Matrices and Performance Comparison

A comprehensive comparison of all LAPACK eigensolvers on a large set of today's computer architectures is in preparation [Demmel et al. 2005]. The article compares inverse iteration, Divide and Conquer [Bunch et al. 1978; Cuppen 1981; Gu and Eisenstat 1995], the QR algorithm, and the MRRR algorithm when computing all eigenpairs. It covers both accuracy (orthogonality and residual norm) and performance (timing) aspects. The new subset functionality of the MRRR algorithm [Marques et al. 2005] is compared to subset computations with inverse iterations; note that the QR algorithm and Divide and Conquer do not permit subset computations at reduced cost.

In theory, QR has $\mathcal{O}(n^3)$ and MRRR has $\mathcal{O}(n^2)$ complexity. Inverse iteration has $\mathcal{O}(n^3)$ complexity in the worst case, depending on the amount of reorthogonalization necessary. Divide and Conquer has $\mathcal{O}(n^3)$ complexity in the worst case, but can be substantially faster, depending on the amount of deflation.

In practice, the MRRR algorithm is usually faster than all other methods on matrices from industrial applications. However, there are matrix classes, such as Wilkinson and glued matrices, that can represent difficulties for the MRRR algorithm (see Dhillon et al. [2005]). In such cases, the Divide and Conquer algorithm can be substantially faster.

### 11.2 The Role of IEEE-754 Arithmetic

As illustrated in the previous sections, bidiagonal factorizations by differential qds algorithms play central roles in the MRRR algorithm that are summarized in the following:

(1) The construction of the representation tree. Each child RRR is computed from its parent by solving $L_+ D_+ L_+^T = LDL^T - \sigma I$ (see Algorithm 2).

(2) The computation of the twisted factorization $LDL^T - \hat{\lambda} I = N_r \Delta_r N_r^T$ (see Eq. (13)). It permits us to solve Eq. (7) by (14) using only multiplications.
(3) The computation of the eigenvalues of a given RRR by bisection using Sturm sequences on $LDL^T$ or the dqds algorithm (see Section 9).

In the implementation of the differential qds algorithms, we rely on IEEE-754 arithmetic [ANSI/IEEE 1985; Overton 2001] to avoid branched tests in loops [Demmel and Li 1994]. Whenever a pivot ($d_+$ in the stationary, $d_-$ in the progressive case) is zero, subsequent computations lead to an invalid operation $\infty/\infty$, denoted by NaN ("not a number"). In such a case, the computation is repeated by a slower variant that avoids the NaN. See Marques et al. [2005] for a detailed exposition.

## 11.3 Parallelization

The MRRR algorithm naturally allows for a parallel computation with perfect $\mathcal{O}(n^2/p)$ memory per processor ($p$ denoting the number of processors used). This is substantially better than parallel inverse iteration, such as PDSYEVX in ScaLAPACK (see Antonelli and Vömel [2005] for details).

There are two parallel implementations: ParEig [Bientinesi et al. 2005] within PLAPACK [van de Geijn 1997], and PDSYEVR [Antonelli and Vömel 2005] for ScaLAPACK [Choi et al. 1996].

ParEig relies on dynamic memory allocation and MPI [Snir et al. 1996], while PDSYEVR works with static workspace and uses the BLACS [Dongarra and Whaley 1995; Dongarra and van de Geijn 1991] and PBLAS [Choi et al. 1995].

## 11.4 Vectorization

In general, LAPACK [Anderson et al. 1999] computational routines make use of basic linear algebra subprograms (BLAS) as much as possible. In particular, matrix-matrix operations, the level-3 BLAS [Dongarra et al. 1990; Blackford et al. 2002], have proved to be powerful for obtaining close to peak performance on many modern architectures. They amortize the cost of obtaining data from main memory by reusing data in the cache or high-level memory (see Dongarra et al. [1998]). As an example, the Divide and Conquer [Bunch et al. 1978; Cuppen 1981; Gu and Eisenstat 1995] algorithm uses the BLAS 3 matrix-matrix multiplication kernel DGEMM to efficiently compute eigenvectors as a product of two orthogonal matrices (at higher storage cost).

The MRRR algorithm does *not* lend itself to the use of BLAS 3 for higher performance. The main computational loop, the bidiagonal factorization through differential qds, is inherently sequential. However, it is possible to perform several of these factorizations at the same time. To explore such possibilities for vectorization is a subject of current research.

## ACKNOWLEDGMENTS

The authors are very grateful to Osni Marques for his invaluable help with thoroughly testing and improving our software. We also thank two referees and the editors for their many detailed comments.

## REFERENCES

ANDERSON, E., BAI, Z., BISCHOF, C., BLACKFORD, S., DEMMEL, J., DONGARRA, J., CROZ, J. D., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. 1999. *LAPACK User's Guide*, 3rd. ed. SIAM, Philadelphia, PA.

ANSI/IEEE 1985. *IEEE Standard for Binary Floating Point Arithmetic*. Std 754-1985 ed. ANSI/IEEE, New York.

ANTONELLI, D. AND VÖMEL, C. 2005. LAPACK working note 168: PDSYEVR. ScaLAPACK's parallel MRRR algorithm for the symmetric eigenvalue problem. Tech. Rep. UCBCSD-05-1399, University of California, Berkeley.

BARLOW, J. AND DEMMEL, J. 1990. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal. 27*, 3, 762–791.

BIENTINESI, P., DHILLON, I. S., AND VAN DE GEIJN, R. A. 2005. A parallel eigensolver for dense symmetric matrices based on multiple relatively robust representations. *SIAM J. Sci. Comput. 27*, 1, 43–66.

BLACKFORD, L. S., DEMMEL, J., DONGARRA, J. J., DUFF, I. S., HAMMARLING, S., HENRY, G., HEROUX, M., KAUFMAN, L., LUMSDAINE, A., PETITET, A., POZO, R., REMINGTON, K., AND WHALEY, R. C. 2002. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Softw. 28*, 2, 135–151.

BUNCH, J., NIELSEN, P., AND SORENSEN, D. 1978. Rank-One modification of the symmetric eigenproblem. *Numer. Math. 31*, 31–48.

CHOI, J., DEMMEL, J., DHILLON, I., DONGARRA, J., OSTROUCHOV, S., PETITET, A., STANLEY, K., WALKER, D., AND WHALEY, R. C. 1996. ScaLAPACK: A portable linear algebra library for distributed memory computers—Design issues and performance. *Comput. Phys. Commun. 97*, 1–15.

CHOI, J., DONGARRA, J. J., OSTROUCHOV, S., PETITET, A., AND WALKER, D. 1995. A proposal for a set of parallel basic linear algebra subprograms. Tech. Rep. UT-CS-95-292, University of Tennessee, Knoxville, TN.

CUPPEN, J. J. M. 1981. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math. 36*, 177–195.

DAVIS, C. AND KAHAN, W. 1970. The rotation of eigenvectors by a perturbation. III. *SIAM J. Numer. Anal. 7*, 1, 1–47.

DEMMEL, J. W. 1997. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, PA.

DEMMEL, J. W., DHILLON, I. S., MARQUES, O. A., PARLETT, B. N., AND VÖMEL, C. 2005. Performance and accuracy of the symmetric eigensolvers in LAPACK. University of California, Berkeley. In preparation.

DEMMEL, J. W., DHILLON, I. S., AND REN, H. 1995. On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic. *Electron. Trans. Num. Anal. 3*, 116–140.

DEMMEL, J. W. AND KAHAN, W. 1990. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput. 11*, 5, 873–912.

DEMMEL, J. W. AND LI, X. S. 1994. Faster numerical algorithms via exception handling. *IEEE Trans. Comp. 43*, 8, 983–992.

DHILLON, I. S. 1997. A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem. Ph.D. thesis, University of California, Berkeley, California, USA.

DHILLON, I. S. AND PARLETT, B. N. 2004b. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra Appl. 387*, 1–28.

DHILLON, I. S. AND PARLETT, B. N. 2004b. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl. 25*, 3, 858–899.

DHILLON, I. S., PARLETT, B. N., AND VÖMEL, C. 2005. Glued matrices and the MRRR algorithm. *SIAM J. Sci. Comput. 27*, 2, 496–510.

DONGARRA, J. AND WHALEY, R. C. 1995. A User's Guide to the BLACS v1.1. Tech. Rep. UT-CS-95-281, University of Tennessee, Knoxville, TN, USA.

DONGARRA, J. J., DU CROZ, J., DUFF, I. S., AND HAMMARLING, S. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1–17.

DONGARRA, J. J., DUFF, I. S., SORENSEN, D. C., AND VAN DER VORST, H. A. 1998. *Numerical Linear Algebra for High-Performance Computers*. SIAM Press, Philadelphia, PA.

DONGARRA, J. J. AND VAN DE GEIJN, R. A. 1991. Lapack working note 37: Two dimensional basic linear algebra communication subprograms. Tech. Rep. UT-CS-91-138, University of Tennessee, Knoxville, TN, USA.

FERNANDO, K. V. AND PARLETT, B. N. 1994. Accurate singular values and differential qd algorithms. *Numeri. Math. 67*, 191–229.

GU, M. AND EISENSTAT, S. C. 1995. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl. 16*, 1, 172–191.

IPSEN, I. C. F. 1997. Computing an eigenvector with inverse iteration. *SIAM Rev. 39*, 2, 254–291.

MARQUES, O. A., PARLETT, B. N., AND VÖMEL, C. 2005. LAPACK working note 167: Subset computations with the MRRR algorithm. Tech. Rep. UCBCSD-05-1392, University of California, Berkeley, Calfornia, USA.

MARQUES, O. A., RIEDY, E. J., AND VÖMEL, C. 2005. Lapack working note 172: Benefits of IEEE-754 features in modern symmetric tridiagonal eigensolvers. Tech. Rep. UCBCSD-05-1414, University of California, Berkeley, California, USA.

OVERTON, M. 2001. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM Press, Philadelphia, PA.

PARLETT, B. N. 1995. *Acta Numerica*. Cambridge University Press, Cambridge, UK. 459–491.

PARLETT, B. N. 1998. *The Symmetric Eigenvalue Problem*. SIAM Press, Philadelphia, PA.

PARLETT, B. N. 2000. For tridiagonals T replace T with $LDL^t$. *J. Comp. Appl. Math. 123*, 117–130.

PARLETT, B. N. 2003. Perturbation of eigenpairs of factored symmetric tridiagonal matrices. *Found. Comput. Math. 3*, 2, 207–223.

PARLETT, B. N. 2005. A bidiagonal matrix determines its hyperbolic SVD to varied relative accuracy. *SIAM J. Matrix Anal. Appl. 26*, 4, 1022–1057.

PARLETT, B. N. AND DHILLON, I. S. 1997. Fernando's solution to Wilkinson's problem: An application of double factorization. *Linear Algebra Appl. 267*, 247–279.

PARLETT, B. N. AND DHILLON, I. S. 2000a. Relatively robust representations of symmetric tridiagonals. *Linear Algebra Appl. 309*, 1–3, 121–151.

PARLETT, B. N. AND MARQUES, O. 2000. An implementation of the dqds algorithm (positive case). *Linear Algebra and Appl. 309*, 217–259.

RUTISHAUSER, H. 1954. Der quotienten-differenzen-algorithmus. *Z. Angew. Math. Phys. 5*, 233–251.

RUTISHAUSER, H. 1976. *Vorlesungen über Numerische Mathematik*. Birkhäuser, Basel.

SNIR, M., OTTO, S., HUSS-LEDERMAN, S., WALKER, D., AND DONGARRA, J. J. 1996. *MPI: The Complete Reference*. MIT Press, Cambridge, MA.

VAN DE GEIJN, R. A. 1997. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press, Cambridge, MA.

Received December 2004; revised December 2005; accepted March 2006