LAPACK WORKING NOTE 162: THE DESIGN AND IMPLEMENTATION OF THE MRRR ALGORITHM[§]

INDERJIT S. DHILLON[*] AND BERESFORD N. PARLETT[†] AND CHRISTOF VÖMEL[‡]

Technical Report UCB//CSD-04-1346
Computer Science Division
UC Berkeley

Abstract. In the 90s, Dhillon and Parlett devised a new algorithm (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 have as focus the theoretical aspects, a documentation of software issues has been missing. In this paper, we discuss the design and implementation of the new MRRR version STEGR that is going to be included in a future release of LAPACK. 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.

Key words. Multiple relatively robust representations, LAPACK, numerical software, design, implementation.

AMS subject classifications. 65F15, 65Y15.

1. Introduction.

The MRRR algorithm for the symmetric tridiagonal eigenvalue problem is an important improvement over standard inverse iteration [9] in that it computes orthogonal eigenvectors without needing Gram-Schmidt orthogonalization. While the theoretical foundation of the MRRR algorithm has already been described [10, 11, 12, 13, 14], this paper addresses the question of how to translate the theory into reliable numerical software. Our presentation is based on the design and implementation STEGR, the LAPACK [1] version of the MRRR algorithm.

This paper is structured as follows. In Section 2, we give a brief summary of the most important principles behind the algorithm of Multiple Relatively Robust Representations that are necessary for understanding the code design and implementation. In Section 3, we describe the features of the main LAPACK driver STEGR. The following sections describe the computational routines, the most important ones being Sections 4 and 5 for computing the eigenvalues and -vectors, respectively. These two functions depend and call a number of auxiliary functions that are explained in Sections 6, 7, 8, 9, and 10. Finally, in Section 11, we give a summary of the codes' dependencies on IEEE arithmetic.

As a guiding principle, we first give a coarse 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 STEGR, LARRE, and LARRV, we give

[*]Department of Computer Science, University of Texas, Austin, TX 78712-1188, USA. inderjit@cs.utexas.edu

[†]Mathematics Department and Computer Science Division, University of California, Berkeley, CA 94720, USA. parlett@math.berkeley.edu

[‡]Computer Science Division, University of California, Berkeley, CA 94720, USA. voemel@eecs.berkeley.edu

[§]This work has been supported by grant no. ACI-9619020, National Science Foundation Cooperative Agreement (NPACI).

an overview of storage requirements and a graphical illustration of the functionalities in order to clarify the code design and software structure.

2. Overview of MRRR. For a symmetric tridiagonal matrix $T \in R^{n\times n}$, the MRRR algorithm promises to compute a set of eigenvalues $\lambda_1, \ldots, \lambda_m$ and a set of vectors $v_1, \ldots, v_m$, $\|v_i\|_2 = 1$, satisfying

$$\|(T - \lambda_i I)v_i\| = \mathcal{O}(n\epsilon\|T\|) \tag{2.1}$$

$$|v_i^T v_j| = \mathcal{O}(n\epsilon), \ \ i \neq j \tag{2.2}$$

using $\mathcal{O}(nm)$ arithmetic operations. Here ϵ is the roundoff unit. Pairs of vectors that satisfy (2.2) are said to be orthogonal to working precision. A stricter criterion for small residual norms is

$$\|(T - \lambda_i I)v_i\| = \mathcal{O}(n\epsilon|\lambda_i|), \tag{2.3}$$

demanding *relatively* small residuals.

Section 2.1 presents the ideas of the MRRR algorithm. After the definition of some technical terms in Section 2.2, we give a short nontechnical presentation of the algorithm. Finally more technical description is given in Section 2.3.

2.1. Informal outline of the underlying ideas. Our discussion is based on the results from [10, 11, 12, 13, 14] that justify the MRRR algorithm. For a review of the theory of inverse iteration, we refer to [9].

2.1.1. Issues of standard inverse iteration. We consider inverse iteration for a general symmetric matrix and some of its important properties.

- **(1A)** Small residuals, in the sense of (2.1), are not enough to guarantee orthogonality to working precision when the eigenvalues are close.
- **(1B)** In contrast, if two eigenvalues are nearly as small as the separation between them, then approximate eigenvectors whose residuals satisfy (2.3) will be orthogonal to working precision. Stated differently: eigenvalues with a large relative gap (relgap(λ) = gap$(\lambda)/|\lambda|$) whose eigenvectors satisfy (2.3) also satisfy (2.2).
- **(1C)** If $\hat{\lambda}$ approximates an eigenvalue λ to high relative accuracy, then just one step of inverse iteration with a good starting vector will achieve (2.3). Moreover, at least one column of $(T - \hat{\lambda} I)^{-1}$ is sufficiently good as a starting vector to guarantee (2.3), namely the column with the largest norm.

2.1.2. How to overcome the limitations of inverse iteration. The observations from Section 2.1.1 suggest the following 'ideal' Algorithm 1.

Algorithm 1 The core MRRR algorithm.

```
for each eigenvalue with a large relative gap do
  Compute an approximation that is good to high relative accuracy.
  Find the column of (T − λ̂I)^{-1} with largest norm.
  Perform one step of inverse iteration.
end for
```

2.1.3. Obstacles. 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 less than the uncertainty in the eigenvalue. Uncertainties in matrix entries can arise from the data itself or from transformations. Unfortunately, small relative changes in the entries of most matrices cause large relative changes in tiny eigenvalues.

- An approximation to the eigenvalues (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 these issues, resulting in an $\mathcal{O}(nm)$ algorithm for m eigenpairs. Specifically, it is based on the following results.

- **(2A)** For a tridiagonal matrix T and most, but not all, shifts σ, 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.
- **(2B)** For a given tridiagonal matrix, 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 $(T - \hat{\lambda} I)^{-1}$ with the largest norm with $\mathcal{O}(n)$ operations.
- **(2D)** In order to apply Algorithm 1, eigenvalues must have large relative gaps. MRRR uses the observation that shifting can alter relative gaps. If the relative gaps of a group 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 the output (L_+, D_+) with $LDL^T - \sigma I = L_+ D_+ L_+^T$ give an exact relation. This property was only discovered in 1991 and ensures that roundoff does not spoil Algorithm 1.

2.2. The essence of the MRRR algorithm.

2.2.1. Glossary of useful terms. The following terms are used in the rest of this paper.

- *(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.
- *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 Γ 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 [12].)
- *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 a subset Γ of the wanted 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 wanted eigenvalues.

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

2.2.2. The essence of the MRRR algorithm. We now can give a more precise and refined description of the essence of the MRRR algorithm that extends Algorithm 1.

Algorithm 2 The essence of the MRRR algorithm.

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

2.3. 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.

2.3.1. Issues and challenges. (1A) First, for any symmetric matrix A, let λ, μ 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

$$v^T w = \frac{1}{\mu - \lambda}\left(r^T w - v^T s\right). \tag{2.4}$$

The two dot products on the right hand side will not vanish in general. Furthermore, there is no intrinsic reason why there should be cancellation in the numerator, so $\|r\|, \|s\| \le Kn\epsilon\|A\|$ is not too pessimistic.

Thus by (2.1) and Cauchy-Schwartz $|v^T w| \le \frac{2Kn\epsilon\|A\|}{|\mu-\lambda|}$ is a realistic bound. 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 $\mathcal{O}(n^2 m)$ work.

(1B) Now suppose that (2.3) holds for λ and for μ, substitute in (2.4) and take absolute values in (2.1) to find

$$|v^T w| = \mathcal{O}\left(\frac{n\epsilon(|\lambda| + |\mu|)}{|\mu - \lambda|}\right). \tag{2.5}$$

Hence, (2.5) guarantees numerically orthogonal eigenvectors provided the separation $|\mu - \lambda|$ is not *much* smaller than $|\lambda|$ and $|\mu|$. Thus, our goal is to achieve (2.3). 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 λ than to any other eigenvalue, then $\|(A - \hat\lambda I)^{-1}\|^{-1} = \delta\lambda$. If we define, for any vector $\hat v$ of unit length, the residual $r(\hat v, \hat\lambda) = (A - \hat\lambda I)\hat v$, then $\delta\lambda := |\lambda - \hat\lambda|$ is an attainable lower bound for $\|r(\hat v, \hat\lambda)\|$.

In general, it is too expensive to find an optimal vector that minimizes the residual norm. However, in the 1960's, Varah and Wilkinson [9] 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

$$\|r(\bar{v},\hat{\lambda})\| = 1/\|\widetilde{v}\| \leq \sqrt{n}\delta\lambda. \tag{2.6}$$

Thus, the canonical vector e_r, when used as 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 λ to high relative accuracy, i.e. $\delta\lambda = \mathcal{O}(\epsilon|\lambda|)$, then $\bar{v}$ achieves (2.3).

2.3.2. Key results and principles of MRRR. (2A) In the 1960's 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 [6]. 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 λ, v is an eigenpair of LDL^T, $\|v\|=1, \lambda \neq 0$ and ρ denotes the spectral diameter, then λ is determined to high relative accuracy by L and D provided that

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

This shows that a factorization can still be an RRR for λ despite possibly large entries in D and L as long as those are neutralized by small entries in v [13].

(2B) Bisection is based on a function that counts the number of eigenvalues of a given matrix 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 [4], a more careful analysis in [5]. However, a comparable analysis of the correctness of bisection for a matrix in product form LDL^T, D a diagonal and L a unit bidiagonal matrix, has not been done yet. Commutative diagrams and proofs of the mixed stability of the stationary dqds factorization $LDL^T-\sigma I = L_+D_+L_+^T$, the progressive dqds factorization $LDL^T-\sigma I = U_-D_-U_-^T$, and twisted factorizations $LDL^T-\sigma I = N_r\Delta_r N_r^T$ (see 2E) are given in [14]. It is a topic of future research to show the correctness of a bisection algorithm based on these different factorizations.

(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 [11] and the references therein. Given an RRR, let $\hat{\lambda}$ be a relatively accurate approximation of λ. Define for $k=1,\ldots,n$ the vector

$$(LDL^T-\hat{\lambda}I)v_k = e_k\gamma_k, \;\; v_k(k)=1. \tag{2.7}$$

The normalization factors

$$\gamma_k, \gamma_k^{-1} = [(LDL^T-\hat{\lambda}I)^{-1}]_{kk}, \tag{2.8}$$

can be computed from a double factorization

$$(LDL^T-\hat{\lambda}I) = L_+D_+L_+^T = U_-D_-U_-^T \tag{2.9}$$

with $\mathcal{O}(n)$ work. Various formulae for the γ_k with different stability properties are given in [11]. Once all γ_k are known, we choose a suitable one, say $k=r$, and solve the corresponding equation (2.7) in a careful way. The choice is guided by the idea of obtaining a right-hand side with a small angle to the (true) eigenvector.

Let v denote the eigenvector for the eigenvalue λ closest to the approximation $\hat{\lambda}$. Then from an eigenvector expansion, it is shown in [11] that

$$\frac{\|(LDL^T - \hat{\lambda}I)v_r\|_2}{\|v_r\|_2} = \frac{|\gamma_r|}{\|v_r\|_2} \leq \frac{\delta\lambda}{\|v\|_\infty} \leq \frac{\delta\lambda}{|v(r)|}. \tag{2.10}$$

In particular, if r is chosen such that $|v(r)| \geq 1/\sqrt{n}$, then an upper bound on (2.10) is $\sqrt{n}\,\delta\lambda$. In practice, instead of finding the minimizing r for the quotient $|\gamma_r|/\|v_r\|_2$ from (2.10), we choose

$$r = \arg \min_{1\leq k\leq n} |\gamma_k|. \tag{2.11}$$

In [11], it is shown that, as $\delta\lambda \to 0$, $(LDL^T - \hat{\lambda}I)^-1$ becomes essentially a rank-1 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 (2.8) and for small enough $\delta\lambda$, finding the largest component of the true eigenvector is equivalent to finding the minimum γ_r of (2.11).

In [11], the resulting $v = v_r$ from (2.7) is called the FP vector, FP for Fernando and Parlett. The Davis-Kahan gap theorem [11] applied to (2.10) shows that v approximates the true eigenvector with an error angle ϕ satisfying

$$|\sin\phi| \leq \frac{\|LDL^Tv - v\hat{\lambda}\|_2}{\|v\|_2 \operatorname{gap}(\hat{\lambda})} \leq \frac{\delta\lambda}{\|v\|_\infty \operatorname{gap}(\hat{\lambda})}, \tag{2.12}$$

where $\operatorname{gap}(\hat{\lambda}) = \min\left\{|\hat{\lambda} - \mu| : \lambda \neq \mu, \mu \in \operatorname{spectrum}(LDL^T)\right\}$. Furthermore, note that by (2.7), the Rayleigh Quotient correction of v to $\hat{\lambda}$ is given by $\gamma/\|v\|^2$.

(2D) The guiding principle of the MRRR algorithm is to compute, by stationary dqds transformations, a sequence of RRRs that achieve large relative gaps by shifting as close as possible to eigenvalue groups previously defined as clustered. 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 wanted eigenvalues to high relative accuracy. The representation of a child (which corresponds to either a subcluster or a singleton within the parent) is computed from the parent representation by differential qd transformations. 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 Γ of (local) eigenvalues. Immediately a relative gap exceeding a threshold is encountered, the inspected subset is declared 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 of Γ. Dhillon [8] observed that the intermediate RRRs (of non-singletons) can be stored in the eigenvector matrix.

(2E) One goal of [14, 13] is to show that roundoff does not spoil the procedure. Specifically, it is shown that 1.) the computed pairs λ_i, v_i have small residuals with respect to the root representation (a weaker form of (2.1) where T is replaced by the factorization of a translate), and 2.) the vectors computed from different representations are orthogonal to working accuracy and satisfy (2.2). The mixed relative stability of the dqds algorithms is essential. Another key ingredient of the procedure for computing the FP vector is the use of a *twisted factorization* to solve (2.7). With the multipliers from (2.9), 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 (2.9) and γ_r from (2.7), define

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

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

$$LDL^T - \hat{\lambda} I = N_r \Delta_r N_r^T. \tag{2.13}$$

(Note that the double factorizations in (2.9) are special cases of (2.13), the forward factorization corresponds to $r = n$ and the backward factorization to $r = 1$.) When we compute the FP vector of the singleton $\hat{\lambda}$, we choose r according to (2.11).

Hence, the solution of (2.7) is equivalent to solving

$$N_r^T v_r = e_r, \;\; v_r(r) = 1. \tag{2.14}$$

The advantage of using this equation is that v_r can be computed solely with multiplications, no additions or subtractions are necessary. By (2.14), we obtain

$$v_r(i) = -L_+(i) v_r(i+1), \;\; i = r-1, \ldots, 1, \tag{2.15}$$

$$v_r(i+1) = -U_-(i) v_r(i), \;\; i = r, \ldots, n-1. \tag{2.16}$$

This feature permits the error analysis of the computed FP vector in [14].

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

STEGR will be an important part of the next release of LAPACK [1]: the simple eigenvalue drivers like SYEVR will make extensive use of STEGR. Furthermore, a parallel version is planned for ScaLAPACK [3]; a possible design alternative that might lend itself better to parallelism is mentioned in Section 3.5.

3.1. Principal algorithmic structure and functionalities. For an overview of the algorithmic structure, see Algorithm 3, a more detailed description is given in the following 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.

Algorithm 3 Principles 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 representation, and their eigenvalues.
if eigenpairs are wanted **then**
  (A3) For each eigenvalue compute its corresponding eigenvector.
end if
(A4) Post-processing. Return the computed eigenvalues (and -vectors) ordered from smallest to largest eigenvalue.

For the illustration of the functionalities and the code dependencies, see Figure 3.1.

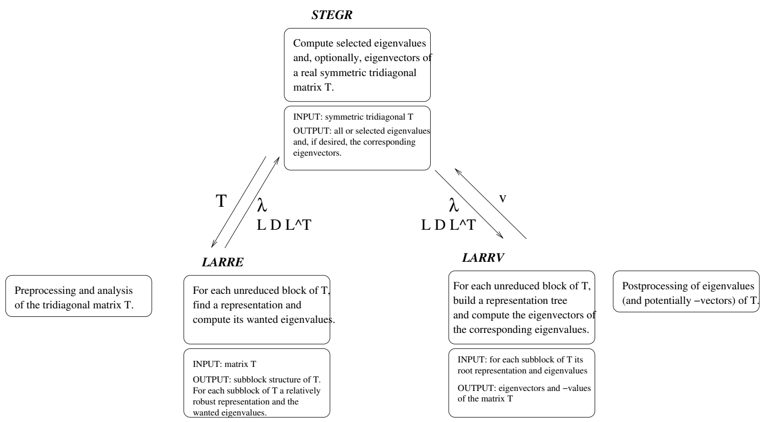

FIG. 3.1. *Principal functionalities of* STEGR.

3.2. Detailed algorithmic structure and functionalities. In this section, we give a detailed description of the algorithmic structure, see Algorithm 4. Important details regarding the pre- and post-processing phase that were omitted in the previous section are explained here. Figure 3.2 illustrates the functionalities and the code dependencies, expanding on Figure 3.1.

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

(A1) *Data preprocessing and parameter error checking:*
Scale T to the allowable range if necessary.
Check parameters and work space.
(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 representation, 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 wanted **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) *Post-processing. Return the computed eigenvalues (and -vectors):*
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**
  unscale the computed eigenvalues
end if
if the matrix T has split into sub-blocks **then**
  Put the eigenvalues (that have been stored by blocks) in increasing order. Apply the same ordering to the eigenvectors.
end if

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 wanted 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 all wanted eigenvalues. However, if the proportion of wanted eigenpairs is high enough or the full spectrum has to be computed, then all eigenvalues are computed by dqds (and, potentially, the unwanted ones are discarded).

3.4. Workspace requirements. STEGR uses two different kinds of workspace, real and integer. In the rest of this paper, WORK denotes the real and IWORK the integer work space.

STEGR partitions the available workspace into two segments, one part is used to store persistent data that is available during the whole computation, and the second part that is going to be reused by different subroutines. The persistent data is computed by LARRE. It is used (and if necessary refined) by LARRV.

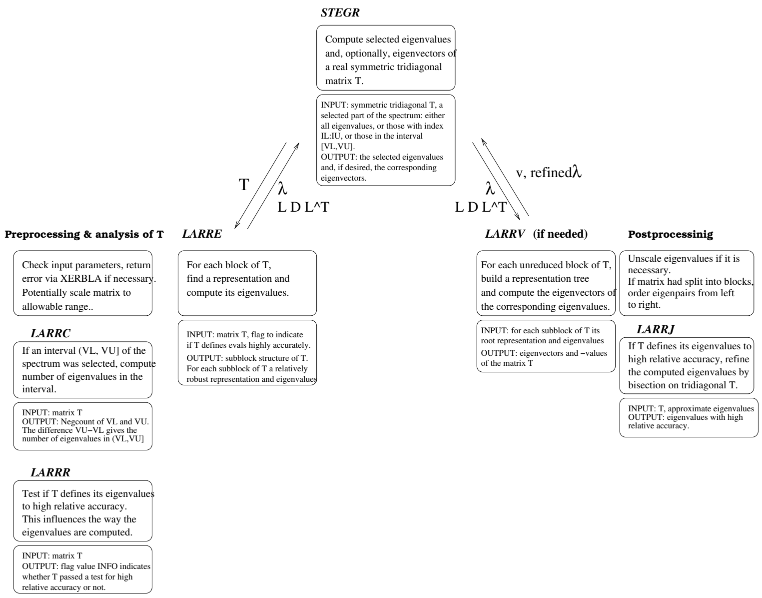

FIG. 3.2. *Detailed functionalities of* STEGR.

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).

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 LARRR, LARRE, LARRV, and potentially LARRJ.

3.5. A note on design alternatives. The computations for different subblocks are completely independent from each other. Thus, it would have been possible to write a code that, instead of computing all eigenvalue first and then the corresponding eigenvectors, consisted of a single loop over all matrix blocks and computed the eigenvectors immediately. This variant might be pursued in a future parallel version.

4. larre: Compute the root representation of each unreduced block and all wanted eigenvalues. LARRE is one of the two major computational subroutines used by STEGR. It computes for each unreduced block its root representation and the wanted eigenvalues by either bisection or dqds.

4.1. Principal algorithmic structure and functionalities. For the algorithmic structure, see Algorithm 5. For the illustration of the functionalities and the code dependencies, see Figure 4.1.

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 wanted eigenvalues. Subsets of the spectrum can be designated either by indices $\mathrm{IL} : \mathrm{IU}$ or by an interval $[\mathrm{VL}, \mathrm{VU}]$.

Record Gersgorin intervals.
Record splitting indices, set negligible off-diagonals to zero.
if only a subset of the spectrum is wanted **then**
  if the wanted subset is given by its indices $\mathrm{IL} : \mathrm{IU}$. **then**
    Find a corresponding interval $[\mathrm{VL}, \mathrm{VU}]$.
  else
    Find the (global) indices $\mathrm{IL} : \mathrm{IU}$.
  end if
  Compute crude approximations to the wanted 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 (or refine) wanted eigenvalues of the RRR by bisection or dqds.
end for

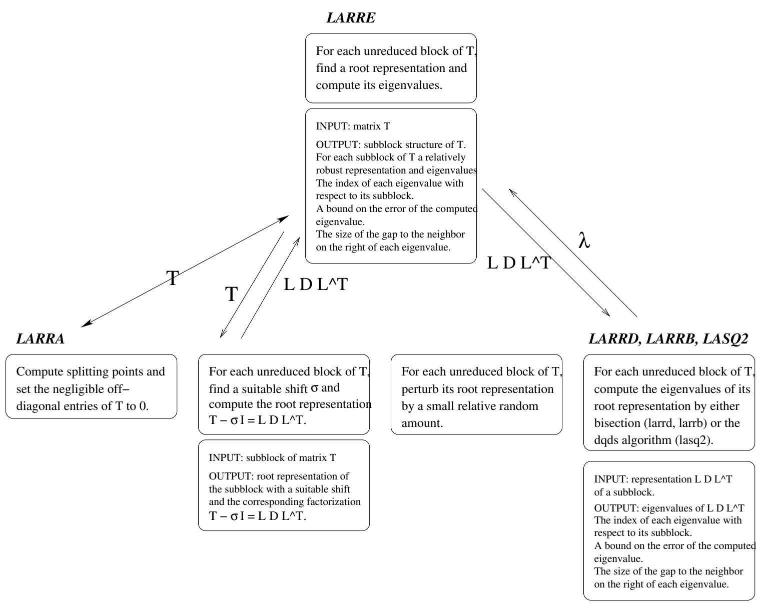

FIG. 4.1. *Principal functionalities of* LARRE.

4.2. Detailed algorithmic structure and functionalities. In this section, we expand our description of the central part of Algorithm 5. Algorithm 6 describes in detail the proceeding once an unreduced block of T has been found. It describes in more detail how the root representation is chosen, in particular the location of the shift and tests to ensure that an RRR has been found. It also gives the details on the computation of the (local) eigenvalues and their indices and block numbers by either bisection or dqds. The dqds algorithm requires a definite RRR whereas bisection does not.

Figure 4.2 expands on Figure 4.1, illustrating of the functionalities and the code dependencies on other subroutines.

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 1×1 block and ignore rest of loop.
  Find local spectral diameter.

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

  (B2) *Choose a shift for the root representation:*
  if Dqds is used **then**
    Compute the extremal eigenvalues accurately.
  else
    Find adequate lower and upper bounds on extreme wanted eigenvalues as initial shifts.
  end if
  In both cases, we have obtained bounds VL and VU on the wanted eigenvalues.
  Compute the Sturm counts at the 0.25 and 0.75 points of the wanted interval $[\mathrm{VL}, \mathrm{VU}]$.
  Choose shift σ 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 tolerance.
    If dqds is going to be used, accept only if the factorization is definite.
    Change shift carefully if factorization is not accepted.
  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 non-trivial entries of L in the off-diagonal part.

  (B5) *Compute wanted 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 σ (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

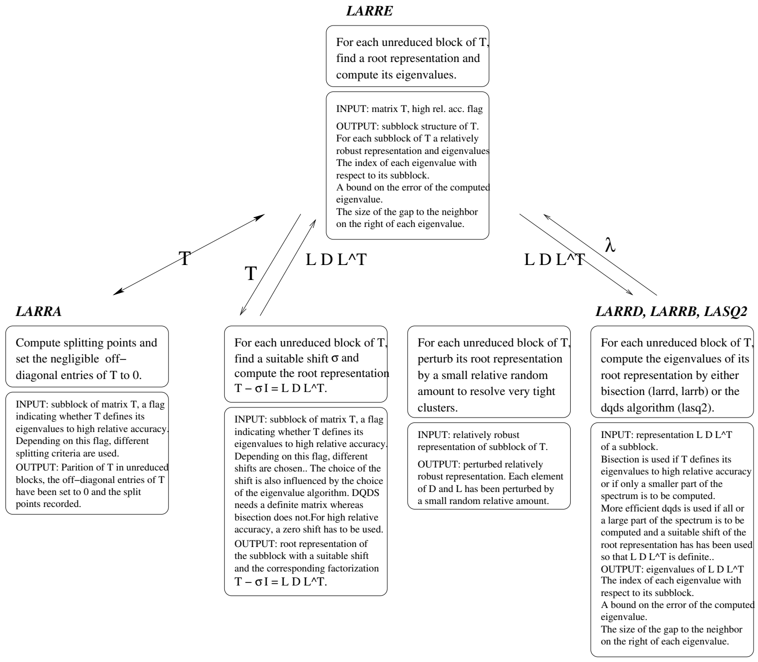

FIG. 4.2. *Detailed functionalities of* LARRE.

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 if they will be improved through Rayleigh Quotient correction in LARRV.
- *Maximum allowable element growth for the root factorization*: When dqds is used, the shift for the root representation makes the matrix definite and thus the factorization is an RRR for all eigenvalues. If the shift for a subset is inside the spectrum, then the LDL^T factorization will be indefinite. Nevertheless it is still an RRR for all eigenvalues if the element growth is small enough.
- *The threshold to switch between bisection and dqds*: If a large enough proportion of the eigenvalues is wanted, the code uses dqds to find all eigenvalues and discards the unwanted ones.
- *The size of the random perturbation that is applied to the root representation*: To break up very tight clusters, tiny random perturbations are made to the root representation.

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

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

5.1. Principal algorithmic structure and functionalities. For the algorithmic structure, see Algorithm 7. For the illustration of the functionalities and the code dependencies, see Figure 5.1.

Algorithm 7 Principal algorithmic structure of LARRV: Build the representation tree from the root representation and refine the eigenvalues. Compute eigenvectors of singletons by Rayleigh Quotient Iteration.

```
for each unreduced block of T do
  if the block does not contain any wanted 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.
      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 (if needed) and compute the corresponding eigenvector.
        end if
      end while
    end for
  end while
end for
```

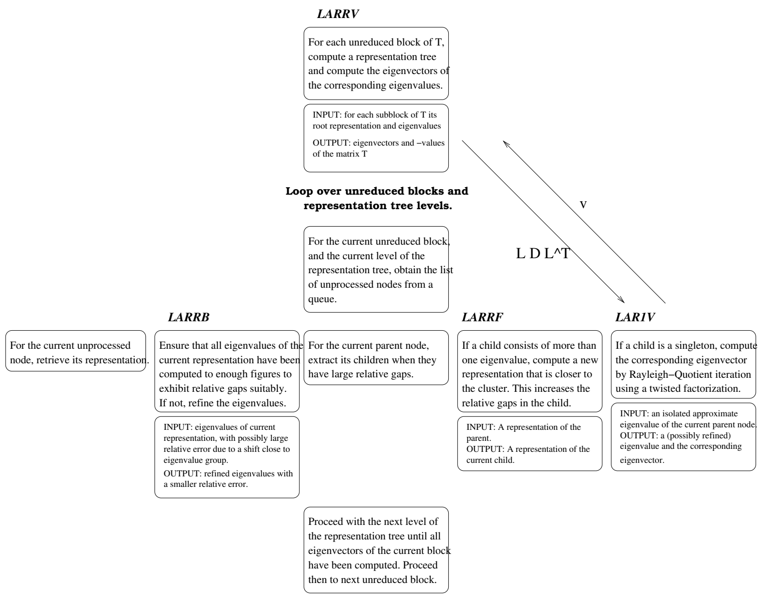

FIG. 5.1. *Principal functionalities of* LARRV.

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.

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, in the unused part of the eigenvector matrix
    Z otherwise).
    Refine the local eigenvalues to enough figures so that the relative gaps can be assessed accu-
    rately 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 until a
      large relative gap is found.
      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
            not exceeded then
              Improve eigenvalue approximation by Rayleigh correction.
            else
              Refine eigenvalue to full accuracy by bisection
            end if
          end if
        end while
        Scale computed vector to unit length and store it and its support.
      end if
    end while
  end for
end while
```

The functionalities and code dependencies of LARRV are illustrated in Figure 5.2.

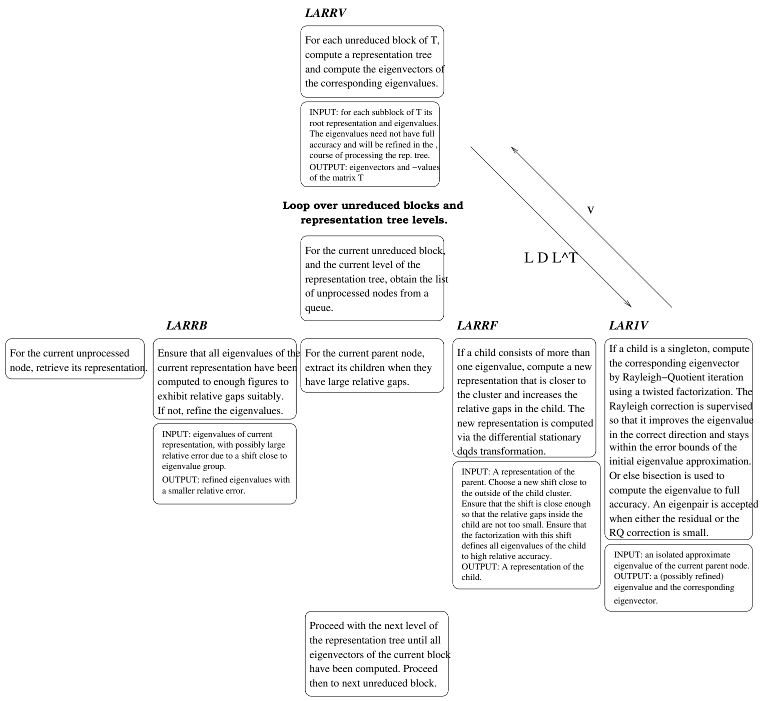

Fig. 5.2. *Detailed functionalities of* LARRV.

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 used in computing the vector. However, a minimal number of leading figures has to be accurate so that relative gaps can be reliably identified.
- *The number of Rayleigh Quotient Correction steps allowed*: Once a singleton is encountered, its eigenvalue with respect to the current RRR must have maximal 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 inside 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. If the Rayleigh Quotient iterate has not converged, the eigenvalue is refined to all its figures by slower but more reliable bisection.
- *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 raised, then the representation tree will become deeper.

5.4. Workspace requirements. LARRV partitions the available work space into a first part that persists and a second part 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),
- the quantities $l_i d_i$ and $l_i^2 d_i$ for the current representation (size $2N$), needed for dqds transformations.

Integer persistent data includes

- the twist index for each singleton (size N),
- the queue for the nodes of the current level of the representation tree and the 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. larrf: Compute the representation of a child from the representation of its parent. LARRF is an auxiliary subroutine called by LARRV to compute a new RRR for a child.

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 λ_i ∈ [l, r]
Set σ_l = l and σ_r = r
while No RRR has been found do
  Compute factorization at left end: L̂D̂L̂^T = LDL^T − σ_l I.
  if factorization has little element growth then
    Accept factorization as RRR and exit.
  end if
  Compute factorization at right end: ĽĎĽ^T = LDL^T − σ_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 best among them 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 has computed by stationary dqds transforms two new factorizations with shifts $\sigma_l = l$ and $\sigma_r = r$. Some element growth has occurred in both factorizations but there is still the possibility that one is an RRR for the wanted subset. A heuristic test is employed.

If one of them is satisfactory, it is accepted, or else the shifts σ_l and σ_r are changed and new factorizations are computed. For any factorization without an NaN, we save the shift and the element growth to select the best among all inspected factorizations as safeguard in case none of them passes our tests for immediate acceptance, see Algorithm 10.

Algorithm 10 Detailed algorithmic structure of the central part of LARRF: How to select a suitable factorization as representation for a child

(D1) *Inspect the factorizations at each end of the child:*
if no NaN has occurred **then**
  choose the one with the smaller element growth as the better one.
else
  Disregard any factorization with an 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 approximate eigenvector v from product of entries of bidiagonal factor L at 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 δ_l and δ_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 δ_l and δ_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

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 should be accepted as RRR immediately. However, there is the danger to select a factorization prematurely without looking at others with even smaller element growth.
- *The amount by which to back off from the end of a child*: As mentioned before, one has to select carefully the amount by which to back off. The goal of backing off is to reduce element growth while to preserve as well as possible large relative gaps. If a shift near the end of a cluster coincides with a Ritz value 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. lar1v: Compute an FP vector from a relatively isolated eigenvalue approximation. LAR1V is an auxiliary subroutine called by LARRV to compute an FP-vector of a singleton. Apart from the FP vector v, it also returns the twist index r from the twisted factorization used, the negcount (the number of pivots smaller than $\hat{\lambda}$), and γ_r. γ_r can be used to compute the residual of the FP vector and the Rayleigh Quotient correction to $\hat{\lambda}$. The optional negcount 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 negcount indicates that the proposed correction step goes in the wrong direction or 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, see Algorithm 11. For the computation of the negcount and a more detailed description, we refer to the next section.

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 $N_r^T v = e_r (\Leftrightarrow N_r \Delta_r N_r^T v = \gamma_r e_r), v(r) = 1$.

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 negcount of $\hat{\lambda}$ can be omitted if $\hat{\lambda}$ is accurate. One detail is omitted in Algorithm 12: it has to be ensured that the negcount is only computed from factorizations without a NaN.

Algorithm 12 Detailed algorithmic structure of LAR1V: Compute an FP vector for $\hat{\lambda}$ from a twisted factorization. Optionally, supply a negcount 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 dstqds and negcount of $\hat{\lambda}$.
if the twist index r has been previously chosen **then**
  Only compute the factorization (and the negcount) down to the index r.
end if
Compute backward factorization $LDL^T - \hat{\lambda}I = U_-D_-U_-^T$ by dqds.
if the twist index r has been previously chosen **then**
  Only compute the factorization (and the negcount) 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 γ_r completes the negcount. (The negcount has actually been done on Δ from the twisted factorization.)
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 $r+1 : n$.

8. larra: Splitting of the matrix T into unreduced blocks. 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 super-linear 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 $\|T\|$.
- 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. larrb, larrc, larrd, and larrj: Sturm counts and bisection. lasq2: dqds.. Bisection plays a central role when computing the eigenvalues to high relative accuracy. It is based on Sturm counts and MRRR makes heavy use of it.

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.

- 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.

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 wanted 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 stationary dqds transform, setting it to 1 to a progressive dqds transform, otherwise it uses 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 post-processing 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 that, see Section 10.

LASQ2 is the LAPACK implementation of the dqds algorithm [15] 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 [2]. Currently, we are testing 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 T̂ = D^{-1/2} T D^{-1/2} with diagonal entries ±1.
if |ê_i| + |ê_{i-1}| < 1 , ∀i then
  return TRUE
else
  return FALSE
end if
```

11. The role of IEEE arithmetic. The implementation of the MRRR algorithm relies on IEEE arithmetic for speed in order to avoid branched tests in loops [7].

Sturm sequences on the tridiagonal matrix T can produce zero pivots whenever the chosen shift is a Ritz value (i.e. the eigenvalue of a principal submatrix). From the recurrence

$$d_i = (T_{ii} - \sigma) - \frac{T_{ii+1}^2}{d_{i-1}}$$

we see that with $d_{i-1} = 0$, the next pivot d_i is set to $-\infty$, and the computation continues normally afterwards with $d_{i+1} = (T_{i+1i+1} - \sigma)$. This allows the inner loop to be written without tests.

Sturm sequences on the factored matrix LDL^T via stationary or progressive dqds transforms can produce a NaN during the computation. In the case of stationary dqds, if $d_{i-1}^+ \neq 0$, the pivots satisfy

$$d_i^+ = (d_i - \sigma) - (d_{i-1}^+ - d_{i-1})(l_{i-1}^2 d_{i-1}/d_{i-1}^+),$$

but if $d_{i-1}^+ = 0$, then $|d_i^+| = \infty$ and $d_{i+1}^+ = \text{NaN}$. For the progressive dqds, if $d_{i+1}^- \neq 0$, the pivots satisfy

$$d_i^- = d_{i-1} l_{i-1}^2 - \sigma + (d_{i+1}^- - d_i l_i^2) * d_i / d_{i+1}^-,$$

but if $d_{i+1}^+ = 0$, then $|d_i^+| = \infty$ and $d_{i-1}^+ = \text{NaN}$. If a NaN occurs in the stationary or the progressive dqds transforms, then all subsequent quantities will be set NaN. In contrast to Sturm sequences with T, the computation has to be repeated with a safe procedure that uses tests and branches. Fortunately, these cases are rare.

12. Summary and conclusions. In this paper, we have shown how the theory of the MRRR algorithm translates into software. We have described the design and implementation of STEGR that will be part of the next release of LAPACK; a parallel version will also be implemented for ScaLAPACK. We have identified governing parameters of the algorithm and discussed their influence on accuracy and performance.

Acknowledgments. The authors are very grateful to Osni Marques for his invaluable help with thoroughly testing and improving our software.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, Philadelphia, 3. edition, 1999.

[2] J. Barlow and J. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal.*, 27(3):762–791, 1990.

[3] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97:1–15, 1996. (also as LAPACK Working Note #95).

[4] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[5] J. W. Demmel, I. S. Dhillon, and H. Ren. On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic. *Electronic Trans. Num. Anal.*, 3:116–140, 1995. LAPACK working note 70, etna.mcs.kent.edu/vol.3.1995/pp116-149.dir/pp116-140.ps.

[6] J. W. Demmel and W. Kahan. accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, 1990.

[7] J. W. Demmel and X. S. Li. Faster numerical algorithms via exception handling. *IEEE Trans. Comp.*, 43(8):983–992, 1994. (Also: LAPACK Working Note 59).

[8] I. S. Dhillon. *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, University of California, Berkeley, California, 1997.

[9] I. C. F. Ipsen. Computing an eigenvector with inverse iteration. *SIAM Review*, 39(2):254–291, 1997.

[10] B. N. Parlett. *Acta Numerica*, chapter The new qd algorithms, pages 459–491. Cambridge University Press, 1995.

[11] B. N. Parlett and I. S. Dhillon. Fernando's solution to Wilkinson's problem: an application of double factorization. *Linear Algebra and Appl.*, 267:247–279, 1997.

[12] B. N. Parlett and I. S. Dhillon. Relatively robust representations of symmetric tridiagonals. *Linear Algebra and Appl.*, 309(1-3):121–151, 2000.

[13] B. N. Parlett and I. S. Dhillon. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices, 2004.

[14] B. N. Parlett and I. S. Dhillon. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25(3):858–899, 2004.

[15] B. N. Parlett and O. Marques. An implementation of the dqds algorithm (Positive case). *Linear Algebra and Appl.*, 309:217–259, 2000.