Optimization Methods and Software

ISSN: 1055-6788 (Print) 1029-4937 (Online) Journal homepage: http://www.tandfonline.com/loi/goms20

An interval-matrix branch-and-bound algorithm for bounding eigenvalues

Dimitrios Nerantzis & Claire S. Adjiman

To cite this article: Dimitrios Nerantzis & Claire S. Adjiman (2016): An interval-matrix branch-and-bound algorithm for bounding eigenvalues, Optimization Methods and Software, DOI: 10.1080/10556788.2016.1184663

To link to this article: http://dx.doi.org/10.1080/10556788.2016.1184663

Optimization Methods & Software, 2016
http://dx.doi.org/10.1080/10556788.2016.1184663

Taylor & Francis
Taylor & Francis Group

An interval-matrix branch-and-bound algorithm for bounding eigenvalues

Dimitrios Nerantzis and Claire S. Adjiman*

Department of Chemical Engineering, Centre for Process Systems Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ, UK

(*Received 1 December 2015; accepted 27 April 2016*)

We present and explore the behaviour of a branch-and-bound algorithm for calculating valid bounds on the *k*th largest eigenvalue of a symmetric interval matrix. Branching on the interval elements of the matrix takes place in conjunction with the application of Rohn's method (an interval extension of Weyl's theorem) in order to obtain valid outer bounds on the eigenvalues. Inner bounds are obtained with the use of two local search methods. The algorithm has the theoretical property that it provides bounds to any arbitrary precision $\epsilon > 0$ (assuming infinite precision arithmetic) within finite time. In contrast with existing methods, bounds for each individual eigenvalue can be obtained even if its range overlaps with the ranges of other eigenvalues. Performance analysis is carried out through nine examples. In the first example, a comparison of the efficiency of the two local search methods is reported using 4000 randomly generated matrices. The eigenvalue bounding algorithm is then applied to five randomly generated matrices with overlapping eigenvalue ranges. Valid and sharp bounds are indeed identified given a sufficient number of iterations. Furthermore, most of the range reduction takes place in the first few steps of the algorithm so that significant benefits can be derived without full convergence. Finally, in the last three examples, the potential of the algorithm for use in algorithms to identify index-1 saddle points of nonlinear functions is demonstrated.

Keywords: global optimization; branch-and-bound; interval matrix; eigenvalue bounds; index-1 saddle points

AMS Subject Classification: 65K; 65H

1. Introduction

In many practical applications requiring the computation of eigenvalues, the matrix of interest is known only as a function of some parameters and is therefore often expressed as an interval matrix [4,10,18]. As a result, there is a need for methods that allow the calculation or estimation of the ranges of the eigenvalues of interval matrices. However in general, problems associated with the eigenvalues of interval matrices are difficult problems. For example, checking positive-(semi)definiteness [18,22] or regularity (existence of singular matrix) [20] of interval matrices are known to be NP-hard problems. Moreover, computing approximate solutions for the minimum and maximum eigenvalues of symmetric interval matrices can be NP-hard [8].

*Corresponding author. Email: c.adjiman@imperial.ac.uk

Eigenvalue bounding methods also play an important role in deterministic global optimization algorithms. They are used in order to create valid convex underestimators [1] of general nonlinear functions. Furthermore, methods for bounding the lowest and second lowest eigenvalues of a symmetric interval matrix can be used as a test for identifying domains in which a twice-continuously differentiable function contains (or does not contain) index-1 saddle points. This can be used within a global deterministic algorithm to speed up the location of index-1 saddle points of potential energy functions [19], a challenging problem with applications in chemical engineering and other areas [5].

A number of methods have been proposed in the literature to obtain lower and upper bounds on the smallest and largest eigenvalues, respectively, of interval matrices [1,7,9,23–25]. Other methods have been devised to compute bounds for each individual eigenvalue [9,11,12,21]. An evolutionary method approach for inner bounds was presented by Yuan et al. [27]. Exact bounds for individual eigenvalues have been given by Deif [3] provided that the signs of the eigenvector entries remain constant over the interval matrix. This condition limits the applicability of this result.

The algorithms by Hladík et al. [10], Leng et al. [15], and Leng [14] can be used to calculate the real eigenvalue set of an interval matrix with any given precision. These algorithms begin with the calculation of an initial inclusion set and proceed by successive identification and removal of parts of the initial inclusion set which do not belong to the eigenvalue set. In particular, the algorithm by Hladík et al. has been shown to be fast and applicable to very large matrices (with small interval widths). However, when the ranges of individual eigenvalues overlap, the methods in [10,14,15] can only provide, at best, the bounds of the union of the overlapping ranges.

We present a branch-and-bound algorithm for the calculation of the bounds of any individual eigenvalue of symmetric interval matrices. The branching occurs on the interval entries of the input matrix. We use Ronh's theorem [9,24], which is an interval extension of Weyl's theorem [6] and local improvement methods in order to obtain valid bounds at each step. The algorithm can be used to calculate the bounds of a specific eigenvalue regardless of whether its range overlaps with that of other eigenvalues or not. Furthermore, the algorithm does not necessarily require the use of interval arithmetic.

The paper is organized as follows: In Section 2 we give a brief introduction to interval matrices and a few definitions. In Section 3 we present the pseudocode of the interval-matrix branch-and-bound algorithm. In Section 4 we present the general bounding approach used in the main algorithm. In Section 5 we present two local search algorithms. One from the existing literature and one given here. These algorithms are used in the main algorithm in order to improve the inner bounds and speed up convergence. In Section 6 we present results from the application of the method and finally and in Section 7, we draw our conclusions.

2. Preliminaries

We denote interval variables with lower case letters inside square brackets, $[\boldsymbol{x}]$, and the corresponding lower and upper bounds of $[\boldsymbol{x}]$ as $\underline{\boldsymbol{x}}$ and $\overline{\boldsymbol{x}}$, respectively. Symmetric interval matrices are denoted by capital letters inside square brackets. An interval matrix is simply a matrix with interval instead of scalar entries. For example, a 2×2 symmetric interval matrix is $[\mathbf{M}] = \begin{bmatrix} [-3,-2] & [-0.5,0.5] \\ [-0.5,0.5] & [-4,-3] \end{bmatrix}$. The symmetric interval matrix $[\mathbf{M}]$ can be interpreted as the infinite set of symmetric scalar matrices $\{\mathbf{M} : m_{ij} \in [m_{ij}] \text{with } m_{ij} = m_{ji}\}$. For example, if $\mathbf{M_1} = \begin{bmatrix} -3 & 0.1 \\ 0.1 & -3 \end{bmatrix}$ then $\mathbf{M_1} \in [\mathbf{M}]$. However if $\mathbf{M_2} = \begin{bmatrix} -3 & 0.2 \\ 0.1 & -3 \end{bmatrix}$ then $\mathbf{M_2} \notin [\mathbf{M}]$.

For an $n \times n$ symmetric scalar matrix $\mathbf{M}$, we denote by $\lambda_i(\mathbf{M})$ the ith largest eigenvalue of $\mathbf{M}$. Therefore, we order the eigenvalues as $\lambda_n(\mathbf{M}) \leq \lambda_{n-1}(\mathbf{M}) \leq \cdots \leq \lambda_1(\mathbf{M})$. We will make use of the following definitions:

DEFINITION 1 (eigenvalues of a symmetric interval matrix) *The ith largest eigenvalue of a symmetric matrix* $[\mathbf{M}]$ *is defined as* $\lambda_i([\mathbf{M}]) = \{\lambda_i(\mathbf{M}) : \mathbf{M} \in [\mathbf{M}]\}$.

Note that the set $\lambda_i([\mathbf{M}]) = \{\lambda_i(\mathbf{M}) : \mathbf{M} \in [\mathbf{M}]\}$ is a compact set in $\mathbb{R}$ ([9]). Thus we can write $\lambda_k([\mathbf{M}]) = [\underline{\lambda_k}, \overline{\lambda_k}]$. To avoid cumbersome notation, we omit the square brackets, which we use to denote a single interval, around $\lambda_k([\mathbf{M}])$.

DEFINITION 2 (spectral radius of a symmetric interval matrix) *The spectral radius,* $\rho([\mathbf{M}])$, *of an interval matrix* $[\mathbf{M}]$ *is defined as* $\rho([\mathbf{M}]) = \max_{\mathbf{M}\in[\mathbf{M}]} \rho(\mathbf{M})$, *where* $\rho(\mathbf{M})$ *is the spectral radius of* $\mathbf{M}$.

DEFINITION 3 (norm of a symmetric interval matrix) *For any norm* $\|\cdot\|$ *defined for scalar matrices, we define the corresponding norm for interval matrices as* $\|[\mathbf{M}]\| = \max\{\|\mathbf{M}\| : \mathbf{M} \in [\mathbf{M}]\}$.

DEFINITION 4 (interior and border of an interval matrix) *The interior of an interval matrix* $[\mathbf{M}]$ *is defined as* $\mathcal{I}([\mathbf{M}]) = \{\mathbf{M} \in [M] : m_{ij} = \underline{m_{ij}} \text{ or } m_{ij} = \overline{m_{ij}} \text{ iff } \underline{m_{ij}} = \overline{m_{ij}}\}$. *The border,* $\mathcal{B}([\mathbf{M}])$, *of* $[\mathbf{M}]$ *is simply the complement of* $\mathcal{I}([\mathbf{M}])$ *in* $\overline{[\mathbf{M}]}$.

3. The interval-matrix branch-and-bound algorithm

In this section we introduce the interval-matrix branch-and-bound algorithm which follows a 'classic' branch-and-bound scheme [13]. Given an $n \times n$ symmetric interval matrix $[\mathbf{M}]$, the algorithm returns lower and upper bounds for $\underline{\lambda_k([\mathbf{M}])} = \min_{\mathbf{M}\in[\mathbf{M}]} \lambda_k([\mathbf{M}])$. The inputs of the algorithm are the symmetric matrix $[\mathbf{M}]$, the order of the eigenvalue (e.g. kth largest eigenvalue), k, for which the bounds will be calculated, the maximum number of iterations, maxiters and the precision ϵ. We denote by L the list which contains sublists of the form $\{[\mathbf{M_i}], l_i, u_i\}$ where $[\mathbf{M_i}]$ is a symmetric matrix with l_i and u_i lower and upper bounds of $\underline{\lambda_k([\mathbf{M_i}])}$. Since the branching procedure can be represented by a binary tree, we will refer to the sublists in L as nodes. Furthermore, we denote the best lower and upper bounds by *BLB* and *BUB*, respectively. Finally, we denote by *RLB* the lowest lower bound of the nodes that have been removed due to the fact that the required precision has been achieved (e.g. the difference between the lower bound at a node and *BUB* is less than ϵ).

The choice of branching strategy (in our case which entry we choose to branch on in step 10 of Algorithm 1) can have a strong influence on the performance of branch-and-bound algorithms (see, for example, [2]). As will be evident from Proposition 4.3 in Section 4, in order to achieve theoretical convergence for a given precision, ϵ, it is necessary to branch on all (off-diagonal) interval entries. A straightforward branching scheme that meets this requirement would be to branch on the interval with the maximum width. However, since the lower bound is given by $\lambda_k(C) - \|[E]\|_\infty$, a more judicious choice might be to branch on the interval entry which reduces $\|[E]\|_\infty$ the most. However, from our experiments with the algorithm, we have not observed any significant effects on performance due to these different branching schemes. Finally, note that in Algorithm 1, the node which is visited at each step is the one with the current lowest lower bound.

Bounds on $\overline{\lambda_k([\mathbf{M}])} = \max_{\mathbf{M}\in[\mathbf{M}]} \lambda_k(\mathbf{M})$ can be calculated in an analogous way to Algorithm 1. In the following section, we discuss the bounding steps and branching of the algorithm in more detail. In the analysis which follows we will assume infinite precision arithmetic.

Algorithm 1 Interval-MatrixBB

(1) Inputs: $[\mathbf{M}]$, k, maxiters, ϵ.
(2) Calculate lower and upper bounds l and u for $\underline{\lambda_k([\mathbf{M}])}$.
(3) Set $BLB = l$, $BUB = u$ and $RLB = +\infty$.
(4) Initialize $L = \{\{[\mathbf{M}], l, u\}\}$, iter $= 0$.
(5) **while** iter $\leq$ maxiters **do**
(6) Choose the first entry, L_1, from list L.
(7) Set $[\mathbf{M}] = L_1[1]$, $l = L_1[2]$, and $u = L_1[3]$.
(8) Delete L_1 from L.
(9) **if** $l < BUB$ **then**
(10) Choose branching entry $[m_{ij}]$, $i \neq j$.
(11) Branch on $[m_{ij}]$ and create $[\mathbf{M_1}]$ and $[\mathbf{M_2}]$.
(12) Obtain bounds l_1 and u_1 for $\underline{\lambda_k([\mathbf{M_1}])}$.
(13) Obtain bounds l_2 and u_2 for $\underline{\lambda_k([\mathbf{M_2}])}$.
(14) $BUB = \min\{u_1, u_2, BUB\}$.
(15) **if** $BUB - l_1 < \epsilon$ **then**
(16) $RLB = \min\{l_1, RLB\}$
(17) **else**
(18) **if** $l_1 < BUB$: Insert $\{[\mathbf{M_1}], l_1, u_1\}$ in L so that the lower bounds in L are in increasing order.
(19) **end if**
(20) **if** $BUB - l_2 < \epsilon$ **then**
(21) $RLB = \min\{l_2, RLB\}$
(22) **else**
(23) **if** $l_2 < BUB$: Insert $\{[\mathbf{M_2}], l_2, u_2\}$ in L so that the lower bounds in L are in increasing order.
(24) **end if**
(25) $BLB = \min\{l_1, l_2, RLB, L_1[2]\}$ ($L_1[2]$ being the second entry of the first sublist in L.
(26) iter++.
(27) **if** L is empty or $BUB - BLB < \epsilon$ **then**
(28) Return BLB and BUB.
(29) **end if**
(30) **end if**
(31) **end while**
(32) Return BLB and BUB.

4. General bounding approach

We can write any given interval matrix $[\mathbf{M}]$ in the form $[\mathbf{M}] = \mathbf{C} + [\mathbf{E}]$ where by $\mathbf{C}$ we denote the centre matrix of $[\mathbf{M}]$, $c_{ij} = (\overline{m_{ij}} + m_{ij})/2$ and by $[\mathbf{E}]$ the radius matrix, $[e_{ij}] = [\underline{m_{ij}} - c_{ij}, \overline{m_{ij}} - c_{ij}] = [-e_{ij}, e_{ij}]$. We make use of the following theorem to calculate bounds, in steps 2, 12, and 13 in Algorithm 1.

THEOREM 4.1 (Rohn [9,24]) *Given a symmetric interval matrix* $[\mathbf{M}] = \mathbf{C} + [\mathbf{E}]$, *then*

$$\lambda_k(\mathbf{C}) + \underline{\lambda_n([\mathbf{E}])} \leq \lambda_k(\mathbf{C} + [\mathbf{E}]) \leq \lambda_k(\mathbf{C}) + \overline{\lambda_1([\mathbf{E}])}, \text{ for } k = 1, 2, \ldots, n. \tag{1}$$

We can write Equation (1) as

$$\lambda_k(\mathbf{C}) - \rho([\mathbf{E}]) \leq \lambda_k(\mathbf{C} + [\mathbf{E}]) \leq \lambda_k(\mathbf{C}) + \rho([\mathbf{E}]). \tag{2}$$

Thus, we can use the following for calculating bounds on $\underline{\lambda_k([\mathbf{M}])}$:

$$l = \lambda_k(\mathbf{C}) + b \leq \underline{\lambda_k([\mathbf{M}])} \leq \lambda_k(\mathbf{C}) = u, \tag{3}$$

where b is a lower bound of $-\rho([\mathbf{E}])$. Since the value $\lambda_k(\mathbf{C})$ is attained for $\mathbf{C} \in [\mathbf{M}]$ this means that $\underline{\lambda_k([\mathbf{M}])}$ cannot be greater than $\lambda_k(\mathbf{C})$, and thus $\lambda_k(\mathbf{C})$ is a valid upper bound. However, a better upper bound can be achieved with the use of local improvement step(s). We will see more on this matter later in this section. On the other hand, we can calculate a lower bound of $-\rho([\mathbf{E}])$ using a number of methods (see, e.g. [1]) or even the exact value of $\rho([\mathbf{E}])$ with the Hertz method [7] ($O(2^{n-1})$). We will use

$$b = \min_{i=1,\ldots,n} \sum_{j=1}^{n} \underline{e_{ij}} = -\|[\mathbf{E}]\|_\infty \leq -\rho([\mathbf{E}]). \tag{4}$$

(This bound is the same as the one we would get by the interval Gerschgorin method [1]). Note that in steps 12 and 13 the new lower bounds l_1, l_2 can actually be worse (lower) than l. In such case we simply replace them with l. The reason for this is that $\lambda_k(\mathbf{C_1})$ and/or $\lambda_k(\mathbf{C_2})$ could be less than $\lambda_k(\mathbf{C})$ and at the same time $b_1 = -\|[\mathbf{E_1}]\|_\infty$ and/or $b_2 = -\|[\mathbf{E_2}]\|_\infty$ might have not improved adequately or even at all. Nevertheless, after a certain number of iterations the lower bound must improve (see Proposition 4.3).

Next, the following lemma tells us that there is no need to branch on the diagonal entries.

LEMMA 4.2 ([9]) *Given an $n \times n$ symmetric interval matrix $[\mathbf{M}]$, define the symmetric interval matrices*

$$[\mathbf{L}] = \{l_{ii} = \underline{m_{ii}} \quad \text{and} \quad [l_{ij}] = [m_{ij}] \quad \text{for } i \neq j\} \tag{5}$$

and

$$[\mathbf{U}] = \{u_{ii} = \overline{m_{ii}} \quad \text{and} \quad [u_{ij}] = [m_{ij}] \quad \text{for } i \neq j\}. \tag{6}$$

Then $\forall \mathbf{M} \in [\mathbf{M}]$, $\exists L \in [\mathbf{L}]$ and $U \in [\mathbf{U}]$ such that

$$\lambda_i(\mathbf{L}) \leq \lambda_i(\mathbf{M}) \leq \lambda_i(\mathbf{U}) \quad \text{for } i = 1, 2, \ldots, n. \tag{7}$$

Lemma 4.2 tells us that we need to consider only the lower (upper) parts of the diagonal elements of an $n \times n$ symmetric matrix $[\mathbf{M}]$ in order to calculate bounds for $\underline{\lambda_k([\mathbf{M}])}$ ($\overline{\lambda_k([\mathbf{M}])}$) for any $k \in \{1, 2, \ldots, n\}$. Thus there is no need for branching on the diagonal elements.

PROPOSITION 4.3 *Consider an $n \times n$ symmetric interval matrix $[\mathbf{M}]$ with $\overline{m_{ij}} - \underline{m_{ij}} = w > 0$ for $i \neq j$ and $\overline{m_{ii}} - \underline{m_{ii}} = 0$ for $i = 1, 2, \ldots, n$. Then, given accuracy ϵ, by successive bisection every submatrix will have $u - l \leq \epsilon$ after a total of $\lceil ((n-1)w/2\epsilon)^{(n^2-n)/2} - 1 \rceil$ iterations (bisections).*

Proof It is helpful to imagine the bisection process as a binary tree. What we would like is to know how many bisections we need to perform in order to have a full binary tree where for each leaf of the tree the corresponding submatrix $[\mathbf{M_i}]$ will have $u_i - l_i \leq \epsilon$.

Initially, for the matrix $[\mathbf{M}] = \mathbf{C} + [\mathbf{E}]$, based on inequality (3), we have $u - l = \|[\mathbf{E}]\|_\infty = (n-1)w/2$. We want $\|[\mathbf{E}]\|_\infty$ to be halved k times so that we have

$$\frac{(n-1)w/2}{2^k} \leq \epsilon \Rightarrow k \geq \log_2 \left(\frac{(n-1)w}{2\epsilon} \right). \tag{8}$$

In order to halve $\|[\mathbf{E}]\|_\infty$ once, we need to branch on each of the $(n^2 - n)/2$ off-diagonal elements. Which means that the depth of the tree will grow to $d = (n^2 - n)/2$. Therefore, in

order to achieve the required accuracy, we will need to end up with a tree of depth $d \geq ((n^2 - n)/2)\log_2((n-1)w/2\epsilon)$. The number of leaves of a full binary tree is 2^d and the corresponding number of bisections is $2^d - 1$. Thus the required number of iterations would be

$$\left\lceil \left(\frac{(n-1)w}{2\epsilon} \right)^{(n^2-n)/2} - 1 \right\rceil. \tag{9}$$

■

At this point we make the following notes: first, although the number of iterations given by (9) is forbiddingly high, this is a worst-case scenario where no node fathoming takes place. As it can be seen in Section 6, the actual performance appears better than (9). This suggests and renders the algorithm practical for small-sized matrices with wide intervals. Furthermore, the actual dimension of the problem depends on the number of interval off-diagonal entries in the matrix and thus can be less than $(n^2 - n)/2$. Second, to the best of our knowledge, no other method exists that can solve the problem of calculating $\underline{\lambda_k}$ and/or $\overline{\lambda_k}$ for $k \in \{2, \ldots, n-1\}$ for general interval symmetric matrices, for a given accuracy ϵ. Thus, we do not know of any better upper bound on the algorithmic complexity of the problem.

5. Local search algorithms

As mentioned in the previous section, we can improve the trivial upper bound, $\lambda_k(\mathbf{M_c})$, of $\underline{\lambda_k}[\mathbf{M}]$. For this purpose we present two local search methods. The first is by Hladík et al. [11], and the second introduced here, based on Theorem 5.1 given in Section 5.2.

5.1 *Hladík et al. [11] local search*

In [11], Hladík et al. proposed a number of local search methods which they used along with eigenvalue bounding methods in order to obtain inner and outer approximations of eigenvalue ranges. Although in general the values obtained from the local search methods are approximate (conservative), they can be sometimes shown to be exact [11]. Here, we will make use of the method, from [11], shown in Algorithm 2 where with $\mathbf{C}$ we denote the centre matrix of $[\mathbf{M}]$ and with $\mathbf{E}$ being the radius matrix (e.g. $e_{ij} = (\overline{m_{ij}} - \underline{m_{ij}})/2$). By $\boldsymbol{v_k}(\mathbf{M})$ we denote the eigenvector of $\mathbf{M}$ which corresponds to the kth largest eigenvalue.

Algorithm 2 Hladík et al's. local improvement algorithm

(1) Inputs: $[\mathbf{M}]$, k
(2) Set $\mathbf{M} = \mathbf{C}$ and $\lambda_k = \infty$
(3) **while** $\lambda_k(\mathbf{M}) < \lambda_k$ **do**
(4) $\quad \lambda_k = \lambda_k(\mathbf{M})$
(5) $\quad \mathbf{D} = \mathrm{diag}(\mathrm{sign}(\boldsymbol{v_k}(\mathbf{M})))$
(6) $\quad \mathbf{M} = \mathbf{C} - \mathbf{DED}$
(7) **end while**
(8) Return λ_k.

For a local search of the maximum of λ_k (i.e. valid lower bound of $\overline{\lambda_k}$), we just replace $\lambda_k = -\infty$, $<$ with $>$ and $-$ with $+$ in Steps 2, 3, and 6, respectively.

5.2 *A new local search method*

It is known that the values $\underline{\lambda_n}$ and $\overline{\lambda_1}$ of a symmetric interval matrix $[\mathbf{M}]$ are attained at extreme matrices of $[\mathbf{M}]$ [7]. However, for the rest of the eigenvalue bounds ($\overline{\lambda_n}$, $\underline{\lambda_1}$, and $\underline{\lambda_k}$, $\overline{\lambda_k}$ for $k = 2, \ldots, n-1$) of a general symmetric interval matrix, the situation is more complicated since boundary values can be attained in $\mathcal{I}([\mathbf{M}])$ and/or $\mathcal{B}([\mathbf{M}])$. Nevertheless, for matrices with non-zero width off-diagonal elements, we can prove the following necessary conditions for $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \text{argmin}\lambda_k(\mathbf{M})\}$ (or $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \text{argmax}\lambda_k(\mathbf{M})\}$ for $k \in \{2, \ldots, n-1\}$) to belong in $\mathcal{I}([\mathbf{M}])$. Note that in the following theorem, the request for the diagonal elements to have zero widths is not an extra assumption and this stems from Lemma 4.2. Otherwise, it would be meaningless to refer to solutions in the interior, since that would never be true. Furthermore, for the remainder of this section, we assume eigenvectors have been normalized with respect to the Euclidean norm.

THEOREM 5.1 *Consider an $n \times n$ symmetric interval matrix $[\mathbf{M}]$ with $\underline{m_{ii}} = \overline{m_{ii}}$ for $i = 1, 2, \ldots, n$ and $m_{ij} \neq \overline{m_{ij}}$ for $i \neq j$. Let $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \text{argmin } \lambda_k(\mathbf{M})\}$ for some integer $k < n$. If $\mathbf{M}^* \in \mathcal{I}([\mathbf{M}])$ then $\lambda_k(\mathbf{M}^*) = \lambda_{k+1}(\mathbf{M}^*)$ or $\|\boldsymbol{v_k}\|_\infty = 1$.*

Proof Let $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \text{argmin } \lambda_k(\mathbf{M})\}$ with $(\boldsymbol{v_i}, \lambda_i)$, $i = 1, 2 \ldots, n$ being the eigenpairs of $\mathbf{M}^*$ ($\boldsymbol{v_i}$ normalized). Assume that $\lambda_k(\mathbf{M}^*) - \lambda_{k+1}(\mathbf{M}^*) > 0$ and $m = \|\boldsymbol{v_k}\|_\infty^2 < 1$. For $\rho \in (0, \lambda_k - \lambda_{k+1}]$ the eigenpairs of the matrix

$$\mathbf{M_1} = \mathbf{M}^* + \rho[m\mathbf{I} - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}] \tag{10}$$

are $(\boldsymbol{v_i}, \lambda_i + \rho m)$ for $i \neq k$ and $(\boldsymbol{v_k}, \lambda_k - \rho(1-m))$. However, in general, $\mathbf{M_1} \notin [\mathbf{M}]$. Consider the diagonal matrix $\mathbf{D}$ with entries $d_{ii} = \boldsymbol{v_{k,i}^2} - m \leq 0$ ($\boldsymbol{v_{k,i}^2}$ being the square of the ith entry of $\boldsymbol{v_k}$) and consider the matrix

$$\mathbf{M_2} = \mathbf{M_1} + \rho\mathbf{D} = \mathbf{M}^* + \rho[\text{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}) - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}], \tag{11}$$

where with $\text{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}})$ we denote the diagonal matrix with diagonal entries equal to the diagonal of $\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}$. Since $\mathbf{M}^* \in I([\mathbf{M}])$, for adequately small $\rho > 0$, $\mathbf{M_2} \in [\mathbf{M}]$ and because the diagonal entries of $\mathbf{M_2}$ are less or equal to the corresponding diagonal entries of $\mathbf{M_1}$ we have that $\lambda_i(\mathbf{M_2}) \leq \lambda_i(\mathbf{M_1})$ for $i = 1, 2, \ldots, n$ and thus $\lambda_k(\mathbf{M_2}) < \lambda_k(\mathbf{M}^*)$. This contradicts the assumption that $\mathbf{M}^* \in \{\text{argmin}\lambda_k(\mathbf{M}) : \mathbf{M} \in [\mathbf{M}]\}$. ■

Note that by 'or' in Theorem 5.1 we mean that at least one of the conditions must be true. The analogous argument of Theorem 5.1 can be made for $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \text{argmax}\lambda_k(\mathbf{M})\}$ for $k > 1$. Furthermore, we can employ the formula used in the proof of Theorem 5.1,

$$\mathbf{M}^* + \rho[\text{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}) - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}] \tag{12}$$

in an attempt to improve the upper bound, in the same way as with Algorithm 2. This is detailed in Algorithm 3.

Algorithm 3 A new local improvement algorithm.

(1) Inputs: $[\mathbf{M}]$, k $(k < n)$
(2) Set $\mathbf{M} = \mathbf{C}$
(3) **while** $\lambda_k(\mathbf{M}) > \lambda_{k+1}(\mathbf{M})$ and $\|\boldsymbol{v_k}(\mathbf{M})\|_\infty < 1$ **do**
(4) $\quad d = \lambda_k(\mathbf{M}) - \lambda_{k+1}(\mathbf{M})$
(5) $\quad \rho_{max} = \max\{\rho \in [0, d] : \mathbf{M} + \rho[\mathrm{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}) - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}] \in [\mathbf{M}]\}$
(6) $\quad$ **if** $\rho_{max} > 0$ **then**
(7) $\quad\quad \mathbf{M} = \mathbf{M} + \rho_{max}[\mathrm{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}) - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}]$
(8) $\quad$ **else**
(9) $\quad\quad$ Return $\lambda_k(\mathbf{M})$
(10) $\quad$ **end if**
(11) **end while**
(12) Return $\lambda_k(\mathbf{M})$.

For a local search of the maximum of λ_k (i.e. lower bound of $\overline{\lambda_k}$), we replace $\lambda_k(\mathbf{M}) < \lambda_{k-1}(\mathbf{M})$ in Step 3, $d = \lambda_{k-1}(\mathbf{M}) - \lambda_k(\mathbf{M})$ in Step 4 and $\mathbf{M} - \rho[\mathrm{diag}(\boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}) - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}}]$ in Steps 5 and 7.

Notice that Algorithm 2 can be applied to any kind of symmetric matrix while Algorithm 3 requires the off-diagonal entries to have non-zero widths. A common feature is that both algorithms, most of the times, terminate after one step and only rarely after two or three steps. Furthermore, Algorithm 2 always searches extreme matrices which makes it more suitable for obtaining an upper bound on $\underline{\lambda_n}$ and a lower bound on $\overline{\lambda_1}$. A comparison between the bounds obtained by the two algorithms is given in Example 1 in Section 6.

In practice we use both local search algorithms and we simply keep the best result. However, we do not apply them in every bounding step of the main algorithm, in order to reduce computational time. The methods are applied at the initial step and then we proceed at each iteration by using the bound obtained by the centre matrix eigenvalue calculation. If this bound happens to be better than the current best upper (or lower if we are bounding $\overline{\lambda_k}$) bound we then apply Algorithms 2 and 3 for further improvement.

We can make a stronger statement than the one of Theorem 5.1 by observing the following: If $\lambda_k = \lambda_{k+1}$, $\lambda_{k+1} \neq \lambda_{k+2}$, and $m = \|\boldsymbol{v_k}\|_\infty^2 + \|\boldsymbol{v_{k+1}}\|_\infty^2 < 1$ then, for adequately small ρ, the matrix $\mathbf{M_1} = \mathbf{M}^* + \rho[mI - \boldsymbol{v_k}\boldsymbol{v_k}^{\mathrm{T}} - \boldsymbol{v_{k+1}}\boldsymbol{v_{k+1}}^{\mathrm{T}}]$ would have 'conveniently placed' eigenvalues (but $\mathbf{M_1}$ not necessarily in $[\mathbf{M}]$) while the corresponding matrix $\mathbf{M_2} = \rho \sum_{i=0}^{1}[\mathrm{diag}(\boldsymbol{v_{k+i}}\boldsymbol{v_{k+i}}^{\mathrm{T}}) - \boldsymbol{v_{k+i}}\boldsymbol{v_{k+i}}^{\mathrm{T}}]$ would have $\lambda_k(\mathbf{M_2}) < \lambda_k(\mathbf{M}^*)$ with $\mathbf{M_2} \in [\mathbf{M}]$. More formally, we have

THEOREM 5.2 *Let* $\mathbf{M}^* \in \{\mathbf{M} \in [\mathbf{M}] : \mathit{argmin}\ \lambda_k(\mathbf{M})\}$. *If* $\mathbf{M}^* \in \mathcal{I}([\mathbf{M}])$ *then* $\exists s \in \{0, 1, \ldots, n-k\}$ *such that* $\lambda_k(\mathbf{M}^*) = \lambda_{k+1}(\mathbf{M}^*) = \cdots = \lambda_{k+s}(\mathbf{M}^*)$ *and* $\sum_{i=0}^{s} \|\boldsymbol{v_{k+i}}\|_\infty^2 \geq 1$.

Proof Assume $\mathbf{M}^* \in \mathcal{I}([\mathbf{M}])$ and that up to some integer $s \in \{0, 1, \ldots, n-k\}$ $\lambda_k = \cdots = \lambda_{k+s}$ (with $\lambda_{k+s} \neq \lambda_{k+s+1}$ otherwise $s = n-k$) and $m = \sum_{i=0}^{s} \|\boldsymbol{v_{k+i}}\|_\infty^2 < 1$. Consider the matrix

$$\mathbf{M_1} = \mathbf{M}^* + \rho\left[m\mathbf{I} - \sum_{i=0}^{s} \boldsymbol{v_{k+i}}\boldsymbol{v_{k+i}}^{\mathrm{T}}\right]. \tag{13}$$

The eigenpairs of $\mathbf{M_1}$ are $(\boldsymbol{v_i}, \lambda_i + \rho m)$ for $i < k$ and $(\boldsymbol{v_k}, \lambda_{k+i} - \rho(1-m))$ for $i = 0, 1, \ldots, s$. Following the same reasoning as in the proof of Theorem 5.1, for adequately small ρ the matrix

$$\mathbf{M_2} = \rho \sum_{i=0}^{s}[\mathrm{diag}(\boldsymbol{v_{k+i}}\boldsymbol{v_{k+i}}^{\mathrm{T}}) - \boldsymbol{v_{k+i}}\boldsymbol{v_{k+i}}^{\mathrm{T}}] \tag{14}$$

would have $\lambda_k(\mathbf{M_2}) < \lambda_k(\mathbf{M}^*)$ and at the same time $\mathbf{M_2} \in [\mathbf{M}]$. This contradicts the assumption that $\mathbf{M}^* \in \{\text{argmin}\lambda_k(\mathbf{M}) : \mathbf{M} \in [\mathbf{M}]\}$. ■

Remark: We could make use of Theorem 5.2 in order to expand the local search method given in Algorithm 3 but this would make the algorithm more complicated without providing any significant benefit in performance. This is due to the fact that the method usually terminates because a bound on one of the interval elements of the interval matrix is reached rather than because $\lambda_k = \lambda_{k+1}$. Nevertheless, we have stated Theorem 5.2 for completeness.

6. Results

In this section we present the results from the application of the algorithm to a number of randomly generated symmetric interval matrices. Given dimension n and radius R, we obtain an interval matrix $[\mathbf{M}] = \mathbf{C} + [\mathbf{E}]$ by generating the central matrix $\mathbf{C}$ with each entry chosen uniformly from $[-20, 20]$ and the $[\mathbf{E}]$ matrix with e_{ij} chosen uniformly from $[0, R]$. In Example 1, we compare the values of the bounds obtained with the two local improvement algorithms. In Examples 2–6 we run the overall bounding algorithm for a maximum of 10^4 iterations and with $\epsilon = 10^{-1}$. Note that $\underline{\lambda_n}$ and $\overline{\lambda_1}$ can be computed much faster, as mentioned previously, by the method from [7]. Nevertheless, we compute the extreme eigenvalue bounds for completeness. The algorithm is implemented in Python 2.7 and the calculations are performed with an Intel Core i7-3770 CPU @ 3.40GHz. Note that the calculations were not performed in a numerically verified way (e.g. use of interval arithmetic) which may lead to numerical errors in the bounds. If required, an implementation based on numerically validated bounds can be obtained with minor effort. The bounds in the results and the interval matrices used in each example are outwardly rounded to the third decimal place (this is why the bounds for $\underline{\lambda_1}$ in Table 1 appear to have width $> 10^{-1}$ yet are marked as converged).

6.1 *Example 1 – comparison of the two local improvement algorithms*

In this example we make a comparison between the two local improvement methods, Algorithms 2 and 3. We generate four groups of random 5×5 matrices consisting of a thousand matrices each and with radii $R = 5, 10, 20,$ and 30, respectively. We apply each algorithm in order to obtain upper bounds on $\underline{\lambda_k}$ for $k = 1,2,3,4$ and lower bounds on $\overline{\lambda_k}$ for $k = 2,3,4,5$. In Figure 1, for each group of random matrices, we plot a histogram of the difference between the bounds from the two algorithms. Positive values indicate that the bounds determined by Algorithm 3 were better than those obtained by Algorithm 2, while negative values indicate the reverse. More explicitly, we denote by $\underline{\lambda_k^{(2)}}^U$ and $\overline{\lambda_k^{(2)}}^L$ the upper and lower bounds on $\underline{\lambda_k}$

Table 1. Summary of results for Example 2.

Eigenvalue	$\underline{\lambda_k}$ bounds / time (CPU s)	$\overline{\lambda_k}$ bounds / time (CPU s)	$\underline{I_{10}}$, $\overline{I_{10}}$	*S* , *OBI*
$[\lambda_1]$	[2.462, 2.563]*/0.55	[30.560, 30.654]* / 0.01	55% , 74%	99% , 28%
$[\lambda_2]$	[-13.534,-13.411]/5.32	[11.267, 11.363]*/0.05	61% , 68%	99% , 35%
$[\lambda_3]$	$[-35.387, -35.304]$*/0.00	[-9.041,-8.900]/7.99	96% , 63%	99% , 30%

Notes: An asterisk indicates the bound widths have converged to 10^{-1}. $\underline{I_{10}}$ and $\overline{I_{10}}$ are the percentages of improvement, for the bounds of $\underline{\lambda_k}$ and $\overline{\lambda_k}$, respectively, after 10 steps (Equations (15) and (16)), *S* is the percentage of relative sharpness of the final bounds (Equation (17)) and *OBI* is the percentage of outer bounds improvement (Equation (18)).

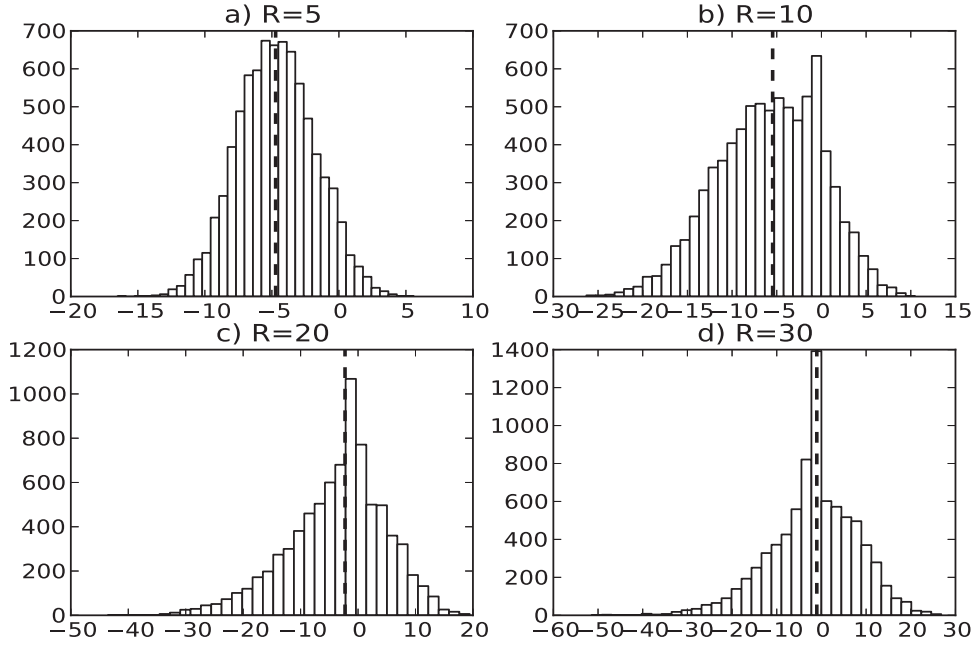

Figure 1. Comparison of bounds obtained by Algorithms 2 and 3. The histograms show the distributions of the values of $\underline{\lambda_k^{(2)^U}} - \underline{\lambda_k^{(3)^U}}$, $k = 1,2,3,4$ and $\overline{\lambda_k^{(3)^L}} - \overline{\lambda_k^{(2)^L}}$, $k = 2,3,4,5$ for 1000 randomly generated matrices, with different radii for each panel: (a) $R = 5$; (b) $R = 10$; (c) $R = 20$; (d) $R = 30$. The dashed vertical lines indicate the median in each case.

and $\overline{\lambda_k}$, respectively, obtained by Algorithm 2 and by $\underline{\lambda_k^{(3)^U}}$ and $\overline{\lambda_k^{(3)^L}}$ the corresponding bounds obtained by Algorithm 3. The values on the x-axis of each histogram represent the quantities $\underline{\lambda_k^{(2)^U}} - \underline{\lambda_k^{(3)^U}}$, $k = 1,2,3,4$ and $\overline{\lambda_k^{(3)^L}} - \overline{\lambda_k^{(2)^L}}$, $k = 2,3,4,5$.

As we see from Figure 1, for smaller radii, Algorithm 2 performs significantly better than Algorithm 3. However, as the radius increases, the relative performance of Algorithm 3 improves, as indicated by the increasing median value. Although on average Algorithm 2 performs better, an overall improvement can be achieved with the combination of the two methods (use of the best result). This comes at no significant cost since both methods terminate after one step in most cases and occasionally after two or three steps.

6.2 *Examples 2–6 – performance of the eigenvalue bounding algorithm*

The next set of examples is used to investigate the performance of Algorithm 1. For Examples 2–6, we plot the interval eigenvalue bounds as obtained by the algorithm and give a summary table of algorithmic performance. To explain the quantities in the table, we denote by $\underline{\lambda_k^{(i)^L}}$ and $\underline{\lambda_k^{(i)^U}}$ the lower and upper bounds, respectively, on $\underline{\lambda_k}$ at iteration i. We also denote by $\overline{\lambda_k^{(i)^L}}$ and $\overline{\lambda_k^{(i)^U}}$ the lower and upper bounds, respectively, on $\overline{\lambda_k}$ at iteration i. When the iteration superscript is omitted, the quantity refers to the corresponding value at the final iteration of the algorithm. In each table, we give the following information: the final bounds for $\underline{\lambda_k}$, $\underline{\lambda_k}^L$, and $\underline{\lambda_k}^U$; the final bounds for $\overline{\lambda_k}$, $\overline{\lambda_k}^L$, and $\overline{\lambda_k}^U$; the computational times required to obtain these bounds; the percentage improvement, denoted by $\underline{I_{10}}$ and $\overline{I_{10}}$, of the bounds on $\underline{\lambda_k}$ and $\overline{\lambda_k}$, respectively,

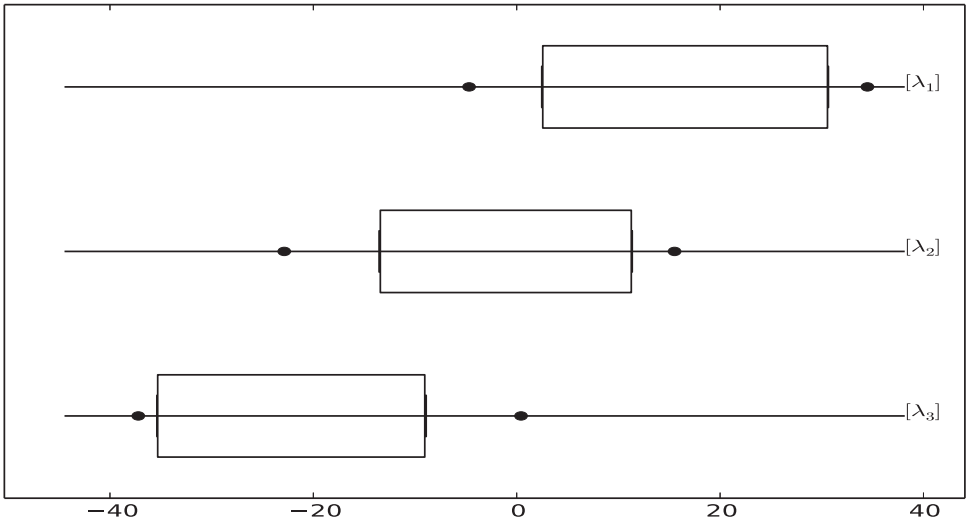

Figure 2. Eigenvalue ranges of Example 2. Initial bounds are shown in black dots. The enclosures provided by the inner bounds are indicated by large rectangles. The distance between the inner and outer bounds is shown via shorter filled rectangles. These are barely visible on this figure due to small width of these intervals.

between the initial and the tenth iterations:

$$\underline{I_{10}} = 100[1 - (\underline{\lambda_k^{(10)}}^U - \underline{\lambda_k^{(10)}}^L)/(\underline{\lambda_k^{(1)}}^U - \underline{\lambda_k^{(1)}}^L)] \tag{15}$$

and

$$\overline{I_{10}} = 100[1 - (\overline{\lambda_k^{(10)}}^U - \overline{\lambda_k^{(10)}}^L)/(\overline{\lambda_k^{(1)}}^U - \overline{\lambda_k^{(1)}}^L)]; \tag{16}$$

the relative sharpness, S, of the final bounds as a percentage:

$$S = 100(\overline{\lambda_k}^L - \underline{\lambda_k}^U)/(\overline{\lambda_k}^U - \underline{\lambda_k}^L); \tag{17}$$

the percentage improvement of the outer bounds between the initial bounds and final outer bounds, *OBI*:

$$OBI = 100(\overline{\lambda_k}^U - \underline{\lambda_k}^L)/(\overline{\lambda_k^{(1)}}^U - \underline{\lambda_k^{(1)}}^L). \tag{18}$$

6.3 *Example 2: n = 3, R = 10*

In this example we calculate eigenvalue bounds for the following randomly generated, 3×3, symmetric matrix with $R = 10$:

$$[\mathbf{M}] = \begin{bmatrix} [-6.852, 6.575] & [2.953, 21.876] & [-0.682, 9.799] \\ [2.953, 21.876] & [1.635, 6.707] & [-11.806, 0.069] \\ [-0.682, 9.799] & [-11.806, 0.069] & [-13.344, -9.041] \end{bmatrix}$$

Results are presented in Table 1 while in Figure 2, we plot the inner, outer and initial outer eigenvalue ranges. Note that all the intervals in Figure 2 overlap and thus other methods would return the more conservative range $[\underline{\lambda_3}, \overline{\lambda_1}] = [-35.387, 11.363]$, corresponding to an increase of 134% in the range of $[\lambda_1]$, 165% in the range of $[\lambda_2]$, and 149% in the range of $[\lambda_3]$.

Table 2. Result table for Example 3.

Eigenvalue	$\underline{\lambda_k}$ bounds/time (CPU s)	$\overline{\lambda}_k$ bounds/time (CPU s)	$\underline{I_{10}}$, $\overline{I_{10}}$	S , OBI
$[\lambda_1]$	[7.824, 7.884]*/0.90	[28.421, 28.488]*/0.01	47% , 58%	99% , 31%
$[\lambda_2]$	[2.011,2.197]/7.64	[18.497,18.880]/7.27	46% , 39%	97% , 43%
$[\lambda_3]$	[−21.369, −21.272]*/0.01	[-3.515,-3.310]/5.33	69% , 48%	98% , 35%
$[\lambda_4]$	[−57.637, −57.549]*/0.00	[−33.795, −33.707]*/0.00	85% , 84%	99% , 18%

Notes: An asterisk indicates the bound widths have converged to 10^{-1}. $\underline{I_{10}}$ and $\overline{I_{10}}$ are the percentages of improvement, for the bounds of $\underline{\lambda_k}$ and $\overline{\lambda}_k$, respectively, after 10 steps (Equations (15) and (16)), S is the percentage of relative sharpness of the final bounds (Equation (17)) and OBI is the percentage of outer bounds improvement (Equation (18)).

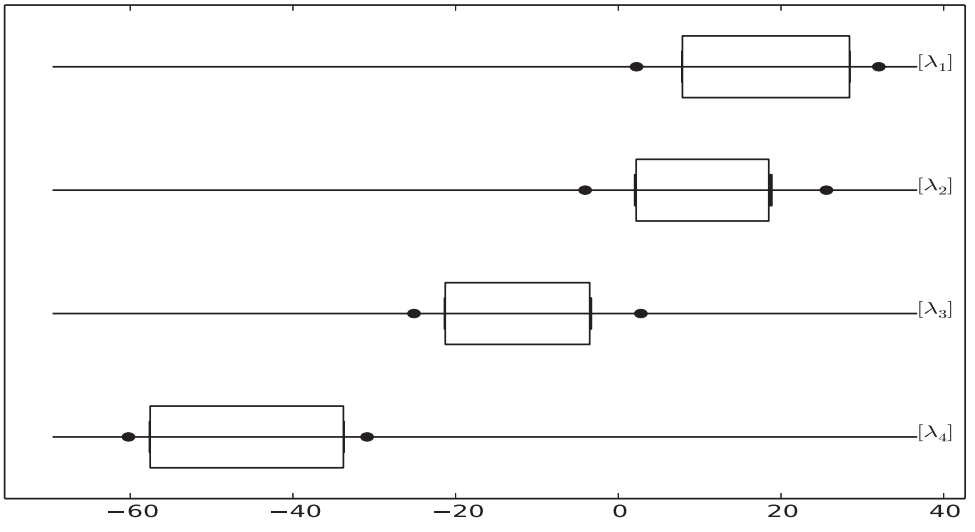

Figure 3. Eigenvalue ranges of Example 3. Initial bounds are shown in black dots. The enclosures provided by the inner bounds are indicated by large rectangles. The distance between the inner and outer bounds is shown via shorter filled rectangles.

6.4 *Example 3: $n=4$, $R=5$*

In this example we increase the dimension of the random test matrix to 4×4 and reduce the radius to $R=5$ and obtain the following matrix:

$$[\mathbf{M}] = \begin{bmatrix} [-4.340, 0.224] & [13.600, 15.001] & [-19.428, -10.118] & [13.756, 22.622] \\ [13.600, 15.001] & [-12.442, -9.068] & [13.962, 23.074] & [-9.825, -3.263] \\ [-19.428, -10.118] & [13.962, 23.074] & [-4.305, -1.296] & [-0.350, 7.571] \\ [13.756, 22.622] & [-9.825, -3.263] & [-0.350, 7.571] & [-13.402, -13.033] \end{bmatrix}$$

Results are shown in Table 2 while in Figure 3, we plot the corresponding eigenvalue ranges. Notice that $[\lambda_1]$ and $[\lambda_2]$ are found to be overlapping, indicating that the more conservative range [2.011, 28.488] would be obtained for both eigenvalues using other methods. This constitutes to increases of 28% and 57% in the ranges of $[\lambda_1]$ and $[\lambda_2]$, respectively.

Table 3. Result table for Example 4.

Eigenvalue	$\underline{\lambda}_k$ bounds/time (CPU s)	$\overline{\lambda}_k$ bounds/time (CPU s)	$\underline{I_{10}}$, $\overline{I_{10}}$	*S* , *OBI*
$[\lambda_1]$	[16.076,17.873]/7.65	[40.296, 40.340]*/0.05	26% , 62%	92% , 28%
$[\lambda_2]$	[4.131,5.041]/7.99	[25.990,26.296]/5.53	36% , 18%	95% , 30%
$[\lambda_3]$	[−14.581, −14.488]*/0.02	[11.064,12.872]/8.04	59% , 28%	93% , 20%
$[\lambda_4]$	[−31.857,−31.715]/2.56	[−12.550,−11.196]/7.86	34% , 28%	93% , 35%
$[\lambda_5]$	[−53.925, −53.834]*/0.01	[−24.441, −24.404]*/0.02	60% , 53%	99% , 11%

Notes: An asterisk indicates the bound widths have converged to 10^{-1}. $\underline{I_{10}}$ and $\overline{I_{10}}$ are the percentages of improvement, for the bounds of $\underline{\lambda}_k$ and $\overline{\lambda}_k$ respectively, after 10 steps (Equations (15) and (16)), *S* is the percentage of relative sharpness of the final bounds (Equation (17)) and *OBI* is the percentage of outer bounds improvement (Equation (18))

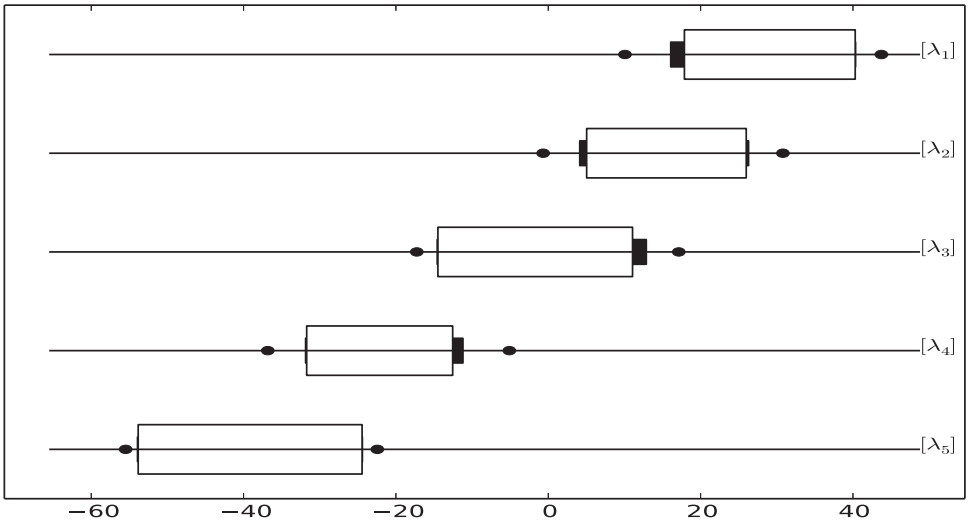

Figure 4. Eigenvalue ranges of Example 4. Initial bounds are shown in black dots. The enclosures provided by the inner bounds are indicated by large rectangles. The distance between the inner and outer bounds is shown via shorter filled rectangles.

6.5 *Example 4: n = 5, R = 5*

In this example we further increase the dimension of the random test matrix to 5×5 and maintain the radius to $R = 5$ to generate:

$$[\mathbf{M}] = \begin{bmatrix} [-13.381, -4.796] & [-20.029, -10.378] & [-6.984, -1.529] & [-18.732, -13.184] & [-1.132, 5.845] \\ [-20.029, -10.378] & [-6.463, -2.382] & [-18.754, -13.526] & [-3.619, 1.946] & [-9.496, -6.370] \\ [-6.984, -1.529] & [-18.754, -13.526] & [3.479, 11.218] & [-15.133, -6.965] & [9.598, 13.545] \\ [-18.732, -13.184] & [-3.619, 1.946] & [-15.133, -6.965] & [-3.744, -1.398] & [11.449, 19.341] \\ [-1.132, 5.845] & [-9.496, -6.370] & [9.598, 13.545] & [11.449, 19.341] & [-10.990, -7.583] \end{bmatrix}$$

Results and eigenvalue ranges for this example are shown in Table 3 and Figure 4, respectively. Figures 5–7 are indicative of the algorithm's behaviour. In the first case (Figure 5), nodes are being removed at a high rate and thus convergence is achieved very fast. In the second case (Figure 6), nodes are removed adequately fast and convergence to the required tolerance can still be achieved by increasing the maximum iteration number. In the third case (Figure 7), almost no nodes are removed and the algorithm does not converge. Nevertheless, a significant improvement with respect to the initial bounds is achieved at the termination of the algorithm. In Figure 8 we

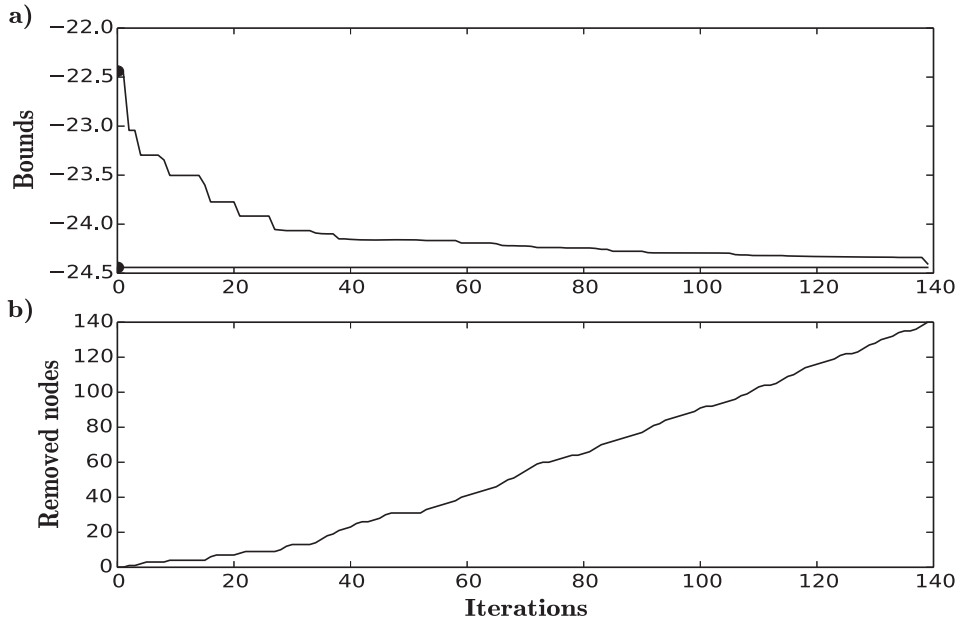

Figure 5. Bounding $\overline{\lambda_5}$: (a) Lower and upper bounds as functions of iterations. The two dots indicate the initial bounds. (b) Number of nodes fathomed as a function of iterations.

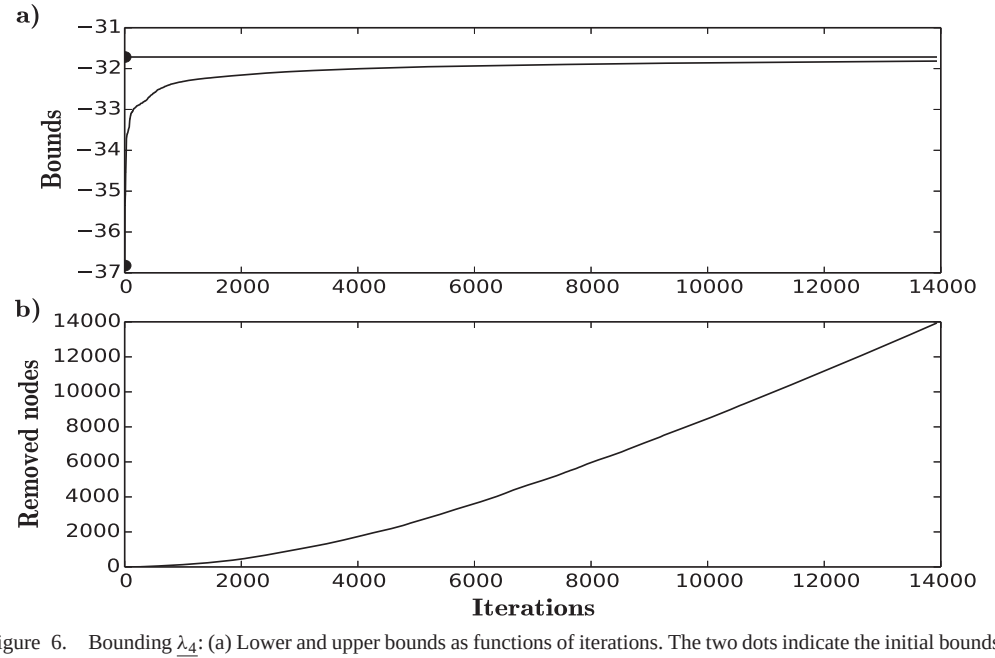

Figure 6. Bounding $\overline{\lambda_4}$: (a) Lower and upper bounds as functions of iterations. The two dots indicate the initial bounds. (b) Number of nodes fathomed as a function of iterations.

show the progress of the bounds on $\overline{\lambda_4}$when we do not make use of local search algorithms. A comparison of this figure with Figure 6 demonstrates that the use of the local search can significantly increase the convergence speed.

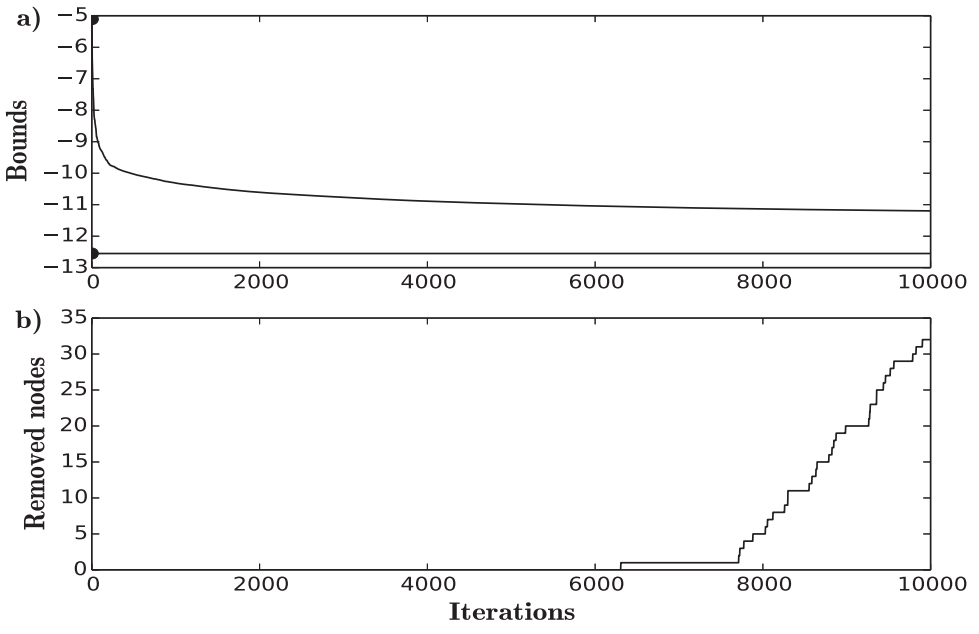

Figure 7. Bounding $\overline{\lambda_4}$: (a) Lower and upper bounds as functions of iterations. The two dots indicate the initial bounds. (b) Number of nodes fathomed as a function of iterations.

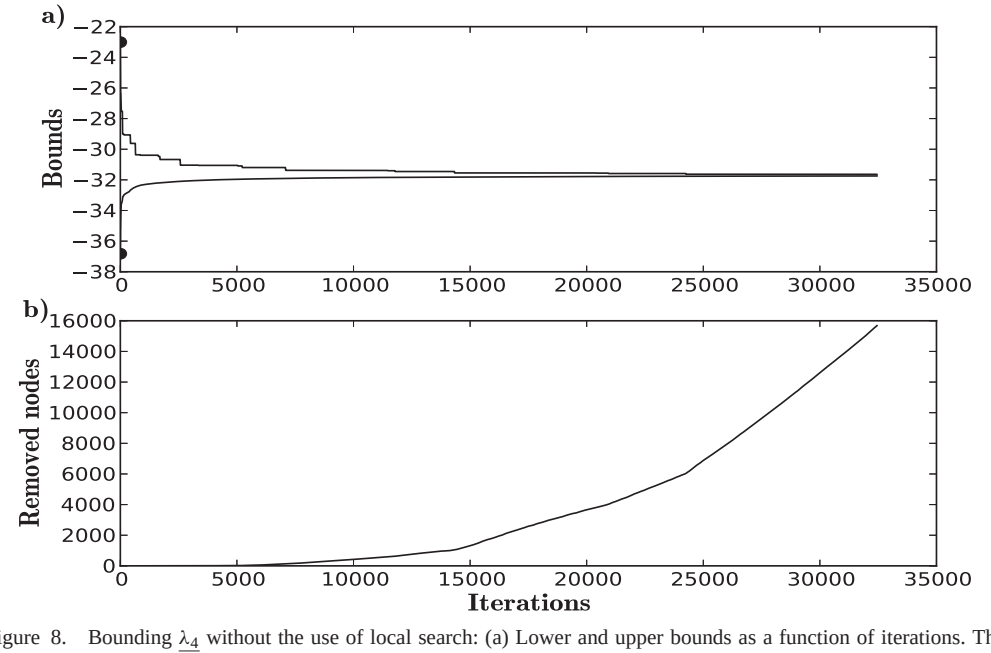

Figure 8. Bounding λ_4 without the use of local search: (a) Lower and upper bounds as a function of iterations. The two dots indicate the initial bounds. (b) Number of nodes fathomed as a function of iterations.

6.6 *Example 5: sparse interval entries, $n = 7$, $R = 5$*

In practical applications, the matrix might contain only a few interval entries. In this example, a random 7×7 symmetric matrix with a small number of interval entries (12 off-diagonal and 2

Table 4. Result table for Example 5.

Eigenvalue	$\underline{\lambda}_k$ bounds/time (CPU s)	$\overline{\lambda}_k$ bounds/time (CPU s)	$\underline{I_{10}}$, $\overline{I_{10}}$	*S* , *OBI*
$[\lambda_1]$	[36.059,36.297]/9.41	[46.023, 46.081]*/0.03	56% , 74%	97% , 53%
$[\lambda_2]$	[29.451, 29.552]*/0.05	[40.183, 40.272]*/6.36	64% , 62%	98% , 50%
$[\lambda_3]$	[4.713, 4.814]*/0.07	[14.494, 14.497]*/0.04	68% , 69%	99% , 53%
$[\lambda_4]$	[−15.814, −15.713]*/1.26	[−7.855, −7.850]*/0.08	54% , 54%	93% , 64%
$[\lambda_5]$	[−20.491, −20.389]*/0.76	[−12.244, −12.063]/6.18	52% , 51%	97% , 64%
$[\lambda_6]$	[−28.396, −28.231]/4.86	[−25.701, −25.157]/10.02	51% , 48%	78% , 84%
$[\lambda_7]$	[−51.939, −51.838]*/0.06	[−39.955, −39.859]*/0.70	66% , 60%	98% , 47%

Notes: An asterisk indicates the bound widths have converged to 10^{-1}. $\underline{I_{10}}$ and $\overline{I_{10}}$ are the percentages of improvement, for the bounds of $\underline{\lambda}_k$ and $\overline{\lambda}_k$, respectively, after 10 steps (Equations (15) and (16)), *S* is the percentage of relative sharpness of the final bounds (Equation (17)) and *OBI* is the percentage of outer bounds improvement (Equation (18))

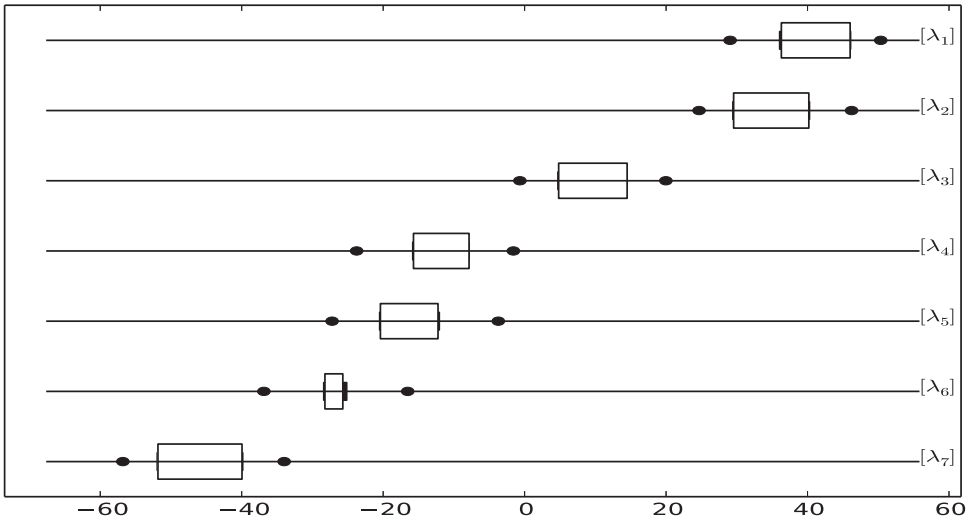

Figure 9. Eigenvalue ranges of Example 5. Initial bounds are shown in black dots. The enclosures provided by the inner bounds are indicated by large rectangles. The distance between the inner and outer bounds is shown via shorter filled rectangles.

diagonal entries randomly chosen with $R = 5$) is considered. The matrix is shown in Appendix 1. Results are given in Table 4 and eigenvalue bounds are plotted in Figure 9. Although this matrix is larger than that in Example 4, the CPU time required is similar. Despite overlap of the ranges of $[\lambda_1]$ and $[\lambda_2]$, and of those of $[\lambda_4]$, $[\lambda_5]$, and $[\lambda_6]$, the algorithm is able to resolve the bounds on these eigenvalues to good or even high accuracy.

6.7 *Example 6: tridiagonal matrix, n = 10, R = 5*

In this example we apply Algorithm 1 to a 10×10 randomly generated tridiagonal symmetric matrix, again with $R = 5$. The matrix is shown in Appendix 2. Results are given in Table 5 and eigenvalue bounds are plotted in Figure 10. It can be seen that computational performance decreases for this larger matrix. Nevertheless, tight bounds are obtained on most of the eigenvalues. The results show that all 10 eigenvalues overlap despite the use of a relatively small radius of 5, and the bounds obtained are much tighter than the worst-case range of $[-36.027, 34.841]$.

Table 5. Result table for Example 6.

Eigenvalue	$\underline{\lambda}_k$ bounds/time (CPU s)	$\overline{\lambda}_k$ bounds/time (CPU s)	$\underline{I_{10}}$, $\overline{I_{10}}$	*S* , *OBI*
$[\lambda_1]$	[19.584,19.883]/9.01	[34.603,34.841]/7.72	50% , 52%	96% , 15%
$[\lambda_2]$	[9.789,10.451]/9.79	[22.553,22.755]/7.00	41% , 44%	93% , 37%
$[\lambda_3]$	[7.707,8.682]/9.26	[18.238, 18.272]*/0.15	34% , 68%	90% , 38%
$[\lambda_4]$	[3.900, 4.001]*/0.25	[15.399,15.947]/9.82	69% , 47%	95% , 34%
$[\lambda_5]$	[2.662, 2.763]*/0.39	[14.837,15.426]/10.12	70% , 50%	95% , 30%
$[\lambda_6]$	[−5.772,−5.452]/8.55	[4.877,5.313]/10.46	61% , 58%	93% , 33%
$[\lambda_7]$	[− 12.201,−11.282]/9.47	[−3.034,−2.248]/10.61	40% , 45%	83% , 42%
$[\lambda_8]$	[−19.074,−18.253]/10.81	[−9.121,−8.691]/8.89	36% , 41%	88% , 45%
$[\lambda_9]$	[−29.158,−28.833]/8.76	[−12.945,−12.223]/8.98	25% , 31%	94% , 16%
$[\lambda_{10}]$	[−36.027,−35.600]/11.24	[−25.408,−24.931]/11.45	55% , 51%	92% , 36%

Notes: An asterisk indicates the bound widths have converged to 10^{-1}. $\underline{I_{10}}$ and $\overline{I_{10}}$ are the percentages of improvement, for the bounds of $\underline{\lambda}_k$ and $\overline{\lambda}_k$ respectively, after 10 steps (Equations (15) and (16)), *S* is the percentage of relative sharpness of the final bounds (Equation (17)) and *OBI* is the percentage of outer bounds improvement (Equation (18))

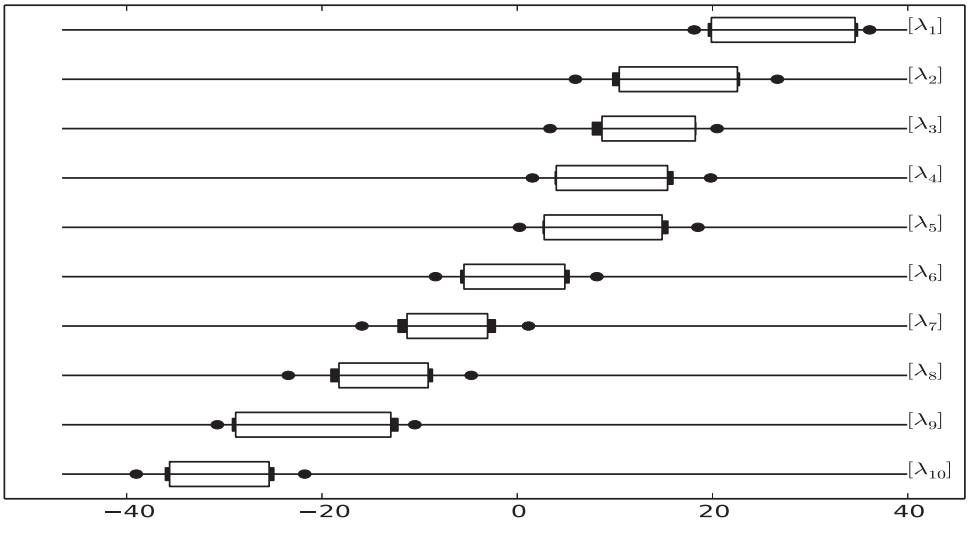

Figure 10. Eigenvalue ranges of Example 6. Initial bounds are shown in black dots. The enclosures provided by the inner bounds are indicated by large rectangles. The distance between the inner and outer bounds is shown via shorter filled rectangles.

6.8 *Examples 7–9: identification of non index-1 areas*

In many rate-based physical processes, such as chemical reactions or nucleation, the identification of transition states on potential energy surfaces is an important and challenging problem. Mathematically the problem is described as follows: Given a C^2 function $f : B \subset \mathbb{R}^n \to \mathbb{R}$ find $\boldsymbol{x}^* \in B$ such that $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ has 1 negative eigenvalue and $n-1$ positive eigenvalues. The point $\boldsymbol{x}^*$ is called a transition state or index-1 saddle point.

One emerging approach to this problem is based on the use of deterministic global search methods in which one aims first to locate all critical points and then to identify within those the physically relevant ones, namely index-1 saddle points (and possibly minima that are also of interest) [16,17,26]. However, in such an approach, it is necessary to obtain full convergence for all critical points, including those that are not of practical relevance. In order to focus the computational effort on the location of index-1 saddle points, a number of practical tests that allow the identification of non index-1 areas have recently been proposed [19]. These tests can

Table 6. Comparison of two approaches to identify areas that cannot contain an index-1 saddle point.

Example	Function	Dim.	L	Rohn's method	Alg. 1(max 5 steps)	Improvement (%)
7	Ackley	3	0.32	1072	1482	38
8	Levy	5	0.9	1025	1803	75
9	Himmelblau	5	2	1058	1189	12

Notes: 'Dim.' refers to the dimensionality of the example, L is the maximum edge length for the random hyper-rectangles, columns 5 and 6 indicate the number of hyper-rectangles, out of 10,000, that are found not to contain an index-1 saddle point, using Rohn's method and Algorithm 1, respectively, column 7 refers to the percentage increase in the number of hyper-rectangles found with the proposed approach

be used in each iteration of a global deterministic search method and allow the exclusion of such areas, thereby avoiding further investigation and speeding up the convergence process. One such test is based on the use of Rohn's method. For example, if we find that $\overline{\lambda}_{n-1} \leq 0$ for the interval Hessian matrix derived from the function f and calculated over a hyper-rectangular area B, then no index-1 saddle points exist in B and thus B can be excluded from the search.

Motivated by the fact that within only a few iterations the algorithm leads to a significant improvement of the initial eigenvalue bounds, as seen from the I_{10} values in Tables 1–5, it might be advantageous to use a few steps of the algorithm instead of a single step (Rohn's method) for the exclusion of non index-1 areas. In Table 6 we show the results for three selected test functions. For each test function, we randomly create 10,000 hyper-rectangles, within a given domain, with each edge having a length selected randomly in the interval $[0, L]$. For each hyper-rectangular area, we first calculate the corresponding interval Hessian matrix. We compare the application of two approaches to obtain an upper bound on $\overline{\lambda}_{n-1}$ and determine whether the region may contain an index-1 saddle point: Rohn's method (i.e. the initial step of Algorithm 1) or Algorithm 1 for a maximum of only five steps. For each approach, we report the number of hyper-rectangle found such that $\overline{\lambda}_{n-1} \leq 0$.

As test functions we use Ackley's function (Example 7, (19)), Levy's function (Example 8, (20)) and an extension to n dimensions of Himmelblau's function (Example 9, (21)):

$$f(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e, \tag{19}$$

with $n = 3$ and $x \in [0.5, 5]^3$.

$$f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2, \tag{20}$$

where $y_i = 1 + (x_i - 1)/4$, $n = 5$, and $x \in [-5, 5]^5$.

$$f(x) = \sum_{i<j}^{n}[(x_i^2 + x_j - 11)^2 + (x_i + x_j^2 - 7)^2], \tag{21}$$

where $n = 5$ and $x \in [-5, 5]^5$

The results indicate that a significant increase in the number of regions identified not to contain an index-1 saddle point is achieved when using Algorithm 1. Furthermore, the solution given by the local search method can be used in a stopping criterion since if the lower bound of $\overline{\lambda}_{n-1}$ is found to be strictly positive, there is no reason to proceed. In practice we would apply the local search only once and prior to branch-and-bound procedure for bounding $\overline{\lambda}_{n-1}$. Finally, in the same way described above, we can use the algorithm to identify convex areas ($\underline{\lambda_n} \geq 0$) and index-1 areas ($\overline{\lambda_n} < 0$ and $\underline{\lambda_{n-1}} > 0$).

7. Conclusions

We have presented a branch-and-bound algorithm for calculating bounds on all individual eigenvalues of a symmetric interval matrix. The algorithm is based on calculating successively tighter bounds by branching on the off-diagonal interval entries of the input matrix using Rohn's method for outer bounds and local search methods for inner bounds. In contrast to other methods, the algorithm provides valid and distinct bounds on each eigenvalue, regardless of whether the ranges of the eigenvalues overlap. Application to five examples, up to a 10×10 matrix, has shown that the algorithm can achieve significant reductions in the range of each eigenvalue compared to existing methods. The use of local search methods has been found to increase the convergence speed significantly. Two approaches have been used for local search: one developed previously [11] and one proposed here. The [11] method is found to perform best on average, but not systematically, making the combination of these two fast approaches desirable.

The algorithm is particularly effective for low-dimensional problems, where by dimension we mean the number of interval entries in the initial matrix. While the algorithm becomes more computationally demanding for larger problems, a few iterations always yield substantial reductions in the eigenvalue ranges and provide a low-cost approach to obtain good bounds. Furthermore, as shown in Examples 7–9, the proposed algorithm can be used as an effective improvement over Rohn's method as a test in deterministic global search methods for the location of index-1 saddle points.

Data statement

Supporting data for this work are available on request by writing to the corresponding author (c.adjiman@imperial.ac.uk).

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

Financial support from the Engineering and Physical Sciences Research Council (EPSRC) of the UK, via a Leadership Fellowship [EP/J003840/1], is gratefully acknowledged.

References

[1] C.S. Adjiman, S. Dallwig, C.A. Floudas, and A. Neumaier, *A global optimization method, αBB, for general twice-differentiable constrained NLPs I. Theoretical advances*, Comput. Chem. Eng. 22 (1998), pp. 1137–1158.
[2] T. Csendes and D. Ratz, *Subdivision direction selection in interval methods for global optimization*, SIAM J. Numer. Anal. 34 (1997), pp. 922–938.
[3] A. Deif, *The interval eigenvalue problem*, ZAMM – J. Appl. Math. Mech./Z. Angew. Math. Mech. 71 (1991), pp. 61–64.
[4] A.D. Dimarogonas, *Interval analysis of vibrating systems*, J. Sound Vib. 183 (1995), pp. 739–749.
[5] M.M.H. Ellabaan, Y.S. Ong, M.H. Lim, and K. Jer-Lai, *Finding Multiple First Order Saddle Points Using a Valley Adaptive Clearing Genetic Algorithm*, in IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2009, pp. 457–462.
[6] G.H. Golub and C.F. Van Loan, *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*, 3rd ed., The Johns Hopkins University Press, Baltimore, 1996.
[7] D. Hertz, *The extreme eigenvalues and stability of real symmetric interval matrices*, IEEE Trans. Autom. Control 37 (1992), pp. 532–535.
[8] M. Hladík, *Complexity issues for the symmetric interval eigenvalue problem*, Open Math, 13 (2015).
[9] M. Hladík, D. Daney, and E. Tsigaridas, *Bounds on real eigenvalues and singular values of interval matrices*, SIAM J. Matrix Anal. Appl. 31 (2010), pp. 2116–2129.

[10] M. Hladík, D. Daney, and E. Tsigaridas, *An algorithm for addressing the real interval eigenvalue problem*, J. Comput. Appl. Math. 235 (2011), pp. 2715–2730.
[11] M. Hladík, D. Daney, and E. Tsigaridas, *Characterizing and approximating eigenvalue sets of symmetric interval matrices*, Comput. Math. Appl. 62 (2011), pp. 3152–3163.
[12] M. Hladík, D. Daney, and E. Tsigaridas, *A filtering method for the interval eigenvalue problem*, Appl. Math. Comput. 217 (2011), pp. 5236–5242.
[13] R. Horst, P.M. Pardalos, and N. Van Thoai, *Introduction to Global Optimization*, Springer Science & Business Media, Dordrecht, 2000.
[14] H. Leng, *Real eigenvalue bounds of standard and generalized real interval eigenvalue problems*, Appl. Math. Comput. 232 (2014), pp. 164–171.
[15] H. Leng, Z. He, and Q. Yuan, *Computing bounds to real eigenvalues of real-interval matrices*, Int. J. Numer. Methods Eng. 74 (2008), pp. 523–530.
[16] Y. Lin and M.A. Stadtherr, *Advances in interval methods for deterministic global optimization in chemical engineering*, J. Global Optim. 29 (2004), pp. 281–296.
[17] Y. Lin and M.A. Stadtherr, *Locating stationary points of sorbate-zeolite potential energy surfaces using interval analysis*, J. Chem. Phys. 121 (2004), pp. 10159–10166.
[18] A. Nemirovskii, *Several NP-hard problems arising in robust stability analysis*, Math. Control, Signals Syst. 6 (1993), pp. 99–105.
[19] D. Nerantzis and C.S. Adjiman, *Deterministic Global Optimization and Transition States*, in 12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering 37 (2015), pp. 851–856.
[20] S. Poljak and J. Rohn, *Checking robust nonsingularity is NP-hard*, Math. Control, Signals Syst. 6 (1993), pp. 1–9.
[21] Z. Qiu, P.C. Müller, and A. Frommer, *An approximate method for the standard interval eigenvalue problem of real non-symmetric interval matrices*, Commun. Numer. Methods Eng. 17 (2001), pp. 239–251.
[22] J. Rohn, *Checking positive definiteness or stability of symmetric interval matrices is NP-hard*, Comment. Math. Univ. Carolin. 35 (1994), pp. 795–797.
[23] J. Rohn, *An algorithm for checking stability of symmetric interval matrices*, IEEE Trans. Automat. Control 41 (1996), pp. 133–136.
[24] J. Rohn, *Bounds on eigenvalues of interval matrices*, ZAMM – J. Appl. Math. Mech./Zeitschrift für Angewandte Mathematik und Mechanik 78 (1998), pp. 1049–1050.
[25] A. Skjäl and T. Westerlund, *New methods for calculating αBB-type underestimators*, J. Global Optim. 58 (2014), pp. 411–427.
[26] K.M. Westerberg and C.A. Floudas, *Locating all transition states and studying the reaction pathways of potential energy surfaces*, J. Chem. Phys. 110 (1999), pp. 9259–9295.
[27] Q. Yuan, Z. He, and H. Leng, *An evolution strategy method for computing eigenvalue bounds of interval matrices*, Appl. Math. Comput. 196 (2008), pp. 257–265.

Appendix 1. Matrix of Example 5 (sparse interval entries)

$$[\mathbf{M}] = \begin{pmatrix} [-17.448, -8.931] & & & & & & \\ -17.933 & -3.0769 & & & & & \\ 0.828 & [-16.605, -8.919] & [-0.766, 3.614] & & & & \\ -2.682 & [13.550, 23.498] & [12.048, 21.553] & 13.237 & & & \\ -16.277 & 5.913 & -19.363 & 3.559 & -4.189 & & \\ [-0.920, 1.502] & [-13.596, -11.311] & 1.160 & -2.585 & 2.626 & -3.293 & \\ -9.448 & -15.174 & -13.762 & -5.352 & 15.225 & 8.797 & -6.353 \end{pmatrix}$$

Appendix 2. Matrix of Example 6 (tridiagonal)

Diagonal: [[−14.964, −11.750], [−16.049, −9.979] , [−4.917, 1.523], [9.020, 19.007], [5.340, 9.723], [10.435, 12.681], [2.355, 10.660], [−10.4775, −2.044], [2.194, 4.969], [−10.137, −1.903]]

Lower/upper diagonal: [[12.707, 15.697], [9.853, 15.241], [−0.266, 2.815], [2.583, 4.573], [5.422, 10.488], [−13.950, −8.734], [14.656, 21.373], [−1.715, 3.770], [10.856, 12.916]]