Determined Global Optimisation and Location of Transition States

Dimitrios Nerantzis

September 2016
**Declaration of Originality**

I, Dimitrios Nerantzis, pledge that the contents of this thesis is my own work and that any work of others has been fully acknowledged as appropriate.

---

**Copyright Declaration**

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution-Non Commercial-No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or distribution, researchers must make clear to others the licence terms of this work.
Abstract

Transition states (index-1 saddle points) play a crucial role in determining the rates of chemical transformations but their reliable identification remains challenging in many applications. Deterministic global optimization methods have previously been employed for the location of transition states (TSs) by initially finding all stationary points and then identifying the TSs among the set of solutions. We propose several regional tests, applicable to general nonlinear, twice continuously differentiable functions, to accelerate the convergence of such approaches by identifying areas that do not contain any TS or that may contain a unique TS. The tests are based on the application of the interval extension of theorems from linear algebra to an interval Hessian matrix. They can be used within the framework of global optimization methods with the potential of reducing the computational time for TS location. We present the theory behind the tests, discuss their algorithmic complexity and show via a few examples that significant gains in computational time can be achieved by using these tests.

Next, 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 various examples.

Finally, we present a refinement method in order to improve (reduce) the $\alpha$ values given by the scaled Gershgorin method and thus create tighter convex underestimators. We apply the new method and compare it with the scaled Gershgorin method on a number of test interval symmetric matrices. Although in the experiments we use the scaled Gershgorin method the refinement algorithm can be utilized to improve the $\alpha$ values of any other method as well.
Acknowledgments

I would like to thank my advisor, Professor Claire S. Adjiman, for always being very supportive and never being stressful.
To anyone who pursues knowledge.
# Contents

1 Introduction ................................................. 17
   1.1 Context of the thesis .................................. 17
   1.2 Goals of the thesis .................................... 18
   1.3 Outline of the thesis .................................. 18

2 Background .................................................. 20
   2.1 Why global optimisation? – Classification of optimisation methods ... 20
   2.2 Branch-and-Bound methods and the αBB algorithm ................. 21
   2.3 Transition States ....................................... 23
   2.4 Interval Matrix Branch and Bound .......................... 24
   2.5 Calculation of α values for the αBB underestimator ............ 26

3 Enclosure of all index-1 saddle points of general nonlinear functions ...... 27
   3.1 Introduction ........................................... 27
   3.2 Preliminaries .......................................... 27
   3.3 Proposed approach .................................... 29
   3.4 Regional tests for the existence of index-1 saddle points ......... 31
      3.4.1 The Gerschgorin test ................................ 32
      3.4.2 Gerschgorin test example ........................... 36
      3.4.3 Recursive Inertia (RecIn) test ....................... 36
      3.4.4 RecIn test example .................................. 40
      3.4.5 Extended RecIn test ................................ 40
      3.4.6 xRecIn test example ................................ 44
3.4.7 2 × 2 Inertia test .............................................. 45
3.4.8 2 × 2 Inertia test example .................................. 47
3.4.9 Rohn test ......................................................... 48
3.4.10 Rohn test example ............................................. 49
3.5 Index-1 areas ....................................................... 50
3.6 Completeness of the tests ....................................... 51
3.7 Algorithmic complexity .......................................... 55
3.8 Results .................................................................. 57
3.8.1 Problem 1: Ackley’s function ................................ 58
3.8.2 Problem 2: Levy function .................................... 60
3.8.3 Problem 3: Himmelblau’s function ......................... 62
3.8.4 Problem 4: 2D-XY lattice model ............................ 64
3.8.5 Overall performance of the tests ......................... 68
3.9 Summary ................................................................ 69

4 An interval-matrix branch-and-bound algorithm for bounding eigenvalues 71
4.1 Introduction ......................................................... 71
4.2 Preliminaries ......................................................... 72
4.3 The interval-matrix branch-and-bound algorithm .................. 73
4.4 General bounding approach ...................................... 76
4.5 Local search algorithms .......................................... 79
  4.5.1 Hladik et al. ([30]) local search ............................. 80
  4.5.2 A new local search method ................................ 80
4.6 Results .................................................................. 85
  4.6.1 Example 1 – Comparison of the two local improvement algorithms .............................................. 86
  4.6.2 Examples 2-6 – Performance of the eigenvalue bounding algorithm ............................................. 88
4.6.3 Example 2: \( n = 3, R = 10 \) ........................................ 89
4.6.4 Example 3: \( n = 4, R = 5 \) ........................................... 91
4.6.5 Example 4: \( n = 5, R = 5 \) .......................................... 93
4.6.6 Example 5: Sparse interval entries, \( n = 7, R = 5 \) ............... 99
4.6.7 Example 6: Tridiagonal matrix, \( n = 10, R = 5 \) ................. 101
4.6.8 Examples 7-9: Identification of non index-1 areas. ............... 103
4.7 Summary ........................................................................... 106

5 Tighter \( \alpha \)BB relaxations through a refinement scheme for the scaled
Gershgorin theorem.

5.1 Introduction ................................................................. 107
5.2 The \( \alpha \)BB underestimator and the scaled Gershgorin method ... 109
5.3 Haynsworth’s theorem .................................................... 112
5.4 The refinement algorithm ............................................... 114
5.5 Results on random symmetric interval matrices ...................... 120
5.6 Results on random interval Hessian matrices ....................... 126
5.7 Summary ...................................................................... 132

6 Conclusions and future work ................................................ 134

6.1 TS location ................................................................... 134
6.2 Interval Matrix Branch and Bound ................................. 136
6.3 Refinement method ....................................................... 137
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Summary of the tests.</td>
<td>54</td>
</tr>
<tr>
<td>3.2</td>
<td>CPU times and number of solutions of each type found for each run for the Ackley function.</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>CPU times and number of solutions of each type found for each run for the Levy function.</td>
<td>61</td>
</tr>
<tr>
<td>3.4</td>
<td>CPU times and number of solutions of each type found for each run for the Himmelblau function.</td>
<td>63</td>
</tr>
<tr>
<td>3.5</td>
<td>CPU times and number of solutions of each type found for each run for the 2D-XY lattice model function.</td>
<td>66</td>
</tr>
<tr>
<td>3.6</td>
<td>Success rate, in percentage, for each test for each problem.</td>
<td>68</td>
</tr>
<tr>
<td>4.1</td>
<td>Summary of results for Example 2. An asterisk indicates the bound widths have converged to $10^{-4}$. $I_{10}$ and $I_{10}$ are the percentages of improvement, for the bounds of $\Lambda_k$ and $\lambda_k$ respectively, after 10 steps (eq. 4.15 and 4.16), $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).</td>
<td>90</td>
</tr>
</tbody>
</table>
4.2 Result table for Example 3. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $I_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\bar{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16). $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

92

4.3 Result table for Example 4. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $I_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\bar{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16). $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

94

4.4 Result table for Example 5. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $I_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\bar{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16). $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

100

4.5 Result table for Example 6. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $I_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\bar{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16). $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

102
4.6 Comparison of two approaches to identify areas that cannot contain an index-1 saddle point. “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 7, respectively; column 7 refers to the percentage increase in the number of hyper-rectangles found with the proposed approach.

5.1 Mean \( I \) values for each reduction option in each group of random matrices.

5.2 Mean \( I \) values for each test function for each \( L \) value.
List of Figures

3.1 Algorithm flowchart .................................................. 31

3.2 Number of unfathomed nodes at each iteration for each run for the
Ackley function. Dashed curves correspond to the same test but with
local search ................................................................. 60

3.3 Number of unfathomed nodes at each iteration for each run for the
Levy function. Dashed curves correspond to the same test but with
local search ................................................................. 62

3.4 Number of unfathomed nodes at each iteration for each run for the
Himmelblau function. Dashed curves correspond to the same test but
with local search ............................................................. 64

3.5 Number of unfathomed nodes at each iteration for each run for the
2D-XY lattice model function. Dashed curves correspond to the same
test but with local search .................................................. 67

4.1 Comparison of bounds obtained by Algorithms 8 and 9. The his-
tograms show the distributions of the values of $\lambda_k^{(2)L} - \lambda_k^{(3)L}$, $k = 1, 2, 3, 4$ and $\lambda_k^{(3)L} - \lambda_k^{(2)L}$, $k = 2, 3, 4, 5$ 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 ................................................................. 87
4.2 Eigenvalue ranges of Example 2. Initial bounds are shown with 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. 

4.3 Eigenvalue ranges of Example 3. Initial bounds are shown with 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. 

4.4 Eigenvalue ranges of Example 4. Initial bounds are shown with 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. 

4.5 Bounding $\lambda_5$: 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. 

4.6 Bounding $\lambda_4$: 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. 

4.7 Bounding $\lambda_4$: 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. 

4.8 Bounding $\lambda_4$ 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. 

4.9 Eigenvalue ranges of Example 5. Initial bounds are shown with 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.
4.10 Eigenvalue ranges of Example 6. Initial bounds are shown with 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.

5.1 Histogram of the \( I \) values (5.18) for the 1000 \( 3 \times 3 \) random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).

5.2 Histogram of the \( I \) values (5.18) for the 1000 \( 4 \times 4 \) random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).

5.3 Histogram of the \( I \) values (5.18) for the 1000 \( 5 \times 5 \) random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).

5.4 Histogram of the \( I \) values (5.18) for the 1000 \( 7 \times 7 \) random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).

5.5 Comparison of the Shared (5.11) and Extra-weighted (5.12) reduction options: histogram of the values \( I_2 - I_1 \) where \( I_1 \) (Shared) and \( I_2 \) (Extra-weighted) are the values shown in Figures 5.4 a) and b) respectively.

5.6 Histogram of the \( I \) values (5.22) for interval Hessians of the Griewank function on randomly selected hyper-rectangular areas with a) \( L = 2 \), b) \( L = 1 \) and c) \( L = 0.2 \).

5.7 Histogram of the \( I \) values (5.22) for interval Hessians of the Levy function on randomly selected hyper-rectangular areas with a) \( L = 2 \), b) \( L = 1 \) and c) \( L = 0.2 \).
5.8 Histogram of the $I$ values (5.22) for interval Hessians of the extended Himmelblau function on randomly selected hyper-rectangular areas with a) $L = 2$, b) $L = 1$ and c) $L = 0.2$.

5.9 Comparison of the Shared (5.11) and Extra-weighted (5.12) reduction options on the random Hessians corresponding to Figure 5.6 (Griewank, $L = 2$). Histogram of the values $I_2 - I_1$ where $I_1$ corresponds to the Shared option $I$ values and $I_2$ to the Extra-weighted option $I$ values.
1 Introduction

1.1 Context of the thesis

The topics in this thesis fall in the general category of deterministic global optimisation and specifically the development of branch-and-bound methods. These methods proceed by dividing the domain space and calculating successively improving upper and lower bounds for the global minimum while identifying and removing areas that do not contain the global solution. Although in this work we use the αBB algorithm, many of the methods developed can be integrated into other algorithms.

One of the problems we address is the location of transition states (index-1 saddle points). Transition states play an important role in chemical engineering and their location remains a challenging problem. We also develop an interval matrix branch and bound algorithm for bounding individual eigenvalues. Furthermore we develop a refinement method for improving the tightness of the bounds given by existing methods used by the αBB algorithm.

At the “core” of this thesis is interval analysis and more specifically interval linear algebra. All parts of the thesis are based upon “extracting” information from interval symmetric matrices. This information revolves around bounding eigenvalues, number of positive and negative eigenvalues and positive (semi)definiteness of an interval symmetric matrix. In most cases, problems related to interval matrices are NP-hard problems.
1.2 Goals of the thesis

We summarise the aims of this thesis as the following three broad goals.

1. Development of a Deterministic Global Optimisation technique for the location of index-1 saddle points.

2. Further development of Deterministic Global Optimisation methods.

3. Development of algorithms that can be used as an important part of deterministic global methods and which provide improvements on similar algorithms already used and presented in the literature.

1.3 Outline of the thesis

The thesis is organized as follows:

In Chapter 1 we begin with the background of this thesis. We discuss deterministic global optimization and why there is a need for developing such methods while giving a brief mention of the relevant literature. We then discuss the problem of locating transition states and address its use in chemical engineering and possibly in other fields, and the state of the art in the current literature, including both local and global methods.

In Chapter 2 we deal with the use of Deterministic Global Optimisation for the location of index-1 saddle points (Transition States). We develop five regional tests for the identification of areas which do not contain any index-1 saddles (or contain a unique index-1 saddle). These tests are used within the αBB algorithm in order to speed-up the process of locating index-1 saddles. We apply the method on a number of test problems and we draw conclusions about the efficiency and the limitations of our approach.

In Chapter 3 we present an Interval-Matrix Branch-and-Bound Algorithm (MBB). This algorithm calculates bounds of any individual (k-th largest) eigenvalue of a symmetric interval matrix even if overlapping of the eigenvalue ranges takes place.
The algorithm is applied to a number of randomly generated matrices as well as interval Hessian matrices calculated over areas of actual multivariable functions.

In Chapter 4 we develop a “refinement” method for the improvement of the \( \alpha \) values calculated by the scaled Gerschgorin method aiming in creating tighter \( \alpha \)BB underestimators. Again, the algorithm is applied to a number of randomly generated matrices as well as interval Hessian matrices calculated over areas of actual multivariable functions.

Finally, in Chapter 5 we summarize the results of this thesis and we discuss future work.
2 Background

2.1 Why global optimisation? – Classification of optimisation methods

In [68], Neumaier gives an extensive review on deterministic global optimisation methods. As mentioned by the author, while on a number of optimisation problems we are satisfied with a good approximation of the optimal solution the need for global methods which guarantee that the problem will be solved to global optimality arises from problems in many fields of science and engineering where only the true global optimum is of interest. Some examples of such cases include phase equilibria [52, 53], protein folding [67], robotics [44, 45] and computer assisted proofs [20].

In the same work ([68]) Neumaier gives a useful classification of optimisation methods based on the guarantees they provide with respect to finding the global minimum for a given problem:

- **Incomplete** methods are methods which usually locate a local minimum.

- **Asymptotically complete** methods guarantee that the global minimum will be found given infinite time but provide no guarantee otherwise that the solution found is the global minimum.

- **Complete** methods find the global minimum with certainty, assuming infinite precision arithmetic, within given tolerance after finite time.
- **Rigorous** methods, find the global minimum with certainty and within given tolerances even in the presence of rounding errors, except in near-degenerate cases, where the tolerances may be exceeded.

Incomplete methods are basically local methods such as any quasi-Newton method (see for example [69]). Stochastic methods such as genetic algorithms [11, 59], particle swarm optimisation [37], simulated annealing [38] and many other (for example [14, 82]), can be considered as asymptotically complete methods assuming they can sample the entire domain, which possibly depends on the implementation. Deterministic methods that fall in the asymptotically complete category include methods such as the DIRECT algorithm [36, 42]. In the last two categories, which the work in this thesis belongs to, we have interval methods such as the Skelboe-Moore algorithm [61, 79] and the interval Newton method [21, 22] and branch-and-bound methods such as αBB [3, 2, 1]. The boundaries between the last two categories can be considered somewhat “thin” and whether a method falls into one category or the other might depend on the implementation.

### 2.2 Branch-and-Bound methods and the αBB algorithm

Branch and bound methods proceed by dividing the domain space and calculating successively improving upper and lower bounds for the global minimum while identifying and removing areas that do not contain the global solution. In the 60’s Land and Doig ([12]) and Little et al. ([13]) introduced the branch and bound scheme for discrete optimisation. Piyavskii’s algorithm ([21]), introduced in 1972, is an example of a Lipschitzian global optimisation method. In 1974, Skelboe introduced a branch and bound method using interval arithmetic ([79]) based on the work of Moore [61]. Perhaps the most well-known interval based method now is the interval Newton method [21, 22] for the solution of nonlinear systems of equations. It is worth mentioning that deterministic global methods have been presented for the
solution of the very challenging class of bilevel problems. For example, Mitsos et al. ([60]) and later Kleniati and Adjiman ([39, 40, 41]) developed branch and bound methods for solving bilevel optimisation problems. Also, Zuhe and Neumaier ([88]) presented an interval algorithm for solving minimax problems (a special case in the broader class of bilevel problems).

Of special interest to this thesis is the \( \alpha \)BB \([51, 52, 53, 54]\) algorithm. The \( \alpha \)BB algorithm is a branch and bound method that can be applied to general nonlinear, twice-continuously differential, constrained optimisation problems:

\[
\min_x \quad f(x) \\
\text{subject to} \quad g_j(x) \leq 0, \ j = 1, \ldots, m \\
\quad h_k(x) = 0, \ k = 1, \ldots, s \\
\quad x_i \in [\underline{x}_i, \overline{x}_i], \ i = 1, \ldots, n.
\]

In each iteration the \( \alpha \)BB algorithm solves a convex relaxation of the above problem,

\[
\min_x \quad \tilde{f}(x) \\
\text{subject to} \quad \tilde{g}_j(x) \leq 0, \ j = 1, \ldots, m \\
\quad \tilde{h}^+_k(x) \leq 0, \ k = 1, \ldots, s \\
\quad \tilde{h}^-_k(x) \leq 0, \ k = 1, \ldots, s \\
\quad x_i \in [\underline{x}_i, \overline{x}_i], \ i = 1, \ldots, n,
\]

where \( \tilde{f}(x), \tilde{g}_j(x), \tilde{h}^+_k(x) \) and \( \tilde{h}^-_k(x) \) are convex underestimators of \( f(x), g_j(x), h_k(x) \) and \(-h_k(x)\) respectively. The solution of the relaxed problem provides a valid lower bound of the global minimum over the specified hyper-rectangular area while any local search can provide a valid upper bound. The \( \alpha \)BB algorithm can create the convex underestimators by exploiting certain terms such as bilinear terms using the McCormick relaxation ([51]). However, one of the most important features of \( \alpha \)BB
is that it can create convex underestimators for general nonlinear twice-continuously differentiable terms. We will discuss this feature of $\alpha$BB in more detail in Chapter 5.

Finally, for an introduction to global deterministic optimisation the reader is referred to the books of Horst et al. [34, 33, 35] while state-of-the-art available software packages (with respect to deterministic global optimisation techniques) include ANTIGONE ([58]) BARON ([78]) and others, a performance comparison of which can be found in [58].

2.3 Transition States

Transition States (TSs) or index-1 saddle points play a crucial role in determining rates of chemical transformations [83] and are also of interest in other areas such as robotics for path planning ([17]) and in game theory where the optimal strategy is located on a saddle point of the cost function ([88]). However, their reliable location remains a challenging problem.

A number of local methods have been proposed in the literature for the identification of transition states. For example, in the Rational Function Optimization (RFO) method [6] and the Dimer method [24], a local search for a single TS is performed, while in the Nudged Elastic Band method [25], an approximation of the minimum energy path between two minima is constructed and a TS is found as the point with the maximum energy on this path. In [16], an alternative approach is based on the transformation of the initial potential energy surface so that TSs correspond to local minima on the new surface. Stochastic methods such as simulated annealing [8] and genetic algorithms [17] have also been employed for locating TSs. While computationally more expensive, such methods do not require any starting points to locate a TS and may find multiple TSs.

Our focus is on deterministic global methods, that can guarantee the identification of all TSs within a specified domain. In the existing literature, the use of such
methods for TS location includes the work of Westerberg and Floudas [85] using the $\alpha$BB algorithm [3, 4] and the work of Lin and Stadtherr [48] using an interval Newton method [22, 12]. In [85] and [48] the authors locate all critical points of a potential energy function and then classify the solutions based on the signs of the eigenvalues of the corresponding Hessian matrices. This approach has been found to be reliable but a drawback in the context of TS location is that computational time is spent locating, to a high accuracy, critical points with index greater than 1 (i.e., with a number of negative eigenvalues greater than 1), and index-0 (i.e. minima). Because of the computational cost associated with deterministic global optimization, it may be beneficial to focus the search on regions that contain TSs only. In Chapter 3 we propose several tests that allow the elimination of certain regions. We apply this approach to a number of test functions. Through these examples, we explore the trade-off between the cost of the tests and the number of iterations and CPU time required to identify all TSs.

2.4 Interval Matrix Branch and Bound

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 [13, 63, 29]. 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 [74, 63] or regularity (existence of singular matrix) [72] 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 ([28]).

Eigenvalue bounding methods also play an important role in deterministic global optimization algorithms. They are used in order to create valid convex underesti-
mators \cite{2} 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 \cite{64}, a challenging problem with applications in chemical engineering and other areas \cite{17}.

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 \cite{26,2,32,80,76,77}. Other methods have been devised to compute bounds for each individual eigenvalue \cite{73,32,30,31}. An evolutionary method approach for inner bounds was presented by Yuan et al. \cite{87}. Exact bounds for individual eigenvalues have been given by Deif \cite{12} 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. \cite{29}, Leng et al. \cite{47}, and Leng \cite{46} 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 \cite{29,47,46} can only provide, at best, the bounds of the union of the overlapping ranges.

In Chapter 4 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 \cite{32,77}, which is an interval extension of Weyl’s theorem \cite{18} 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.

2.5 Calculation of $\alpha$ values for the $\alpha$BB underestimator

As we have briefly mentioned already, the $\alpha$BB algorithm can construct a convex underestimator of a general nonlinear function, $f \in C^2$ over a hyper-rectangular area $X = [x_1, x_1], \ldots, [x_n, x_n]^T$. This is accomplished by adding to $f$ a function,

$$q(x) = \sum_{i=1}^{n} \alpha_i(x_i - \underline{x}_i)(\overline{x}_i - x_i), \alpha_i \geq 0.$$

The $\alpha_i$ values have to be determined in order for the resulting underestimating function, $F(x) = f(x) + q(x)$, to be convex over the area $X$. This implies that the Hessian matrix of $F$, $H_F = H_f + H_q$ has to be positive semi-definite for every $x \in X$. Note that the Hessian $H_q$ of $q$ is diagonal with diagonal entries equal to $2\alpha_i$. Thus by making the $\alpha$ values large enough we can make the $H_F(x)$ positive semi-definite $\forall x \in X$.

The $\alpha$BB algorithm calculates the interval Hessian of $f$ over the area of interest and through the use of a number of possible methods (see [2]) which provide eigenvalue bounds the $\alpha$ values are determined so as to guarantee the positive semi-definiteness of $H_F$ and thus the convexity of the underestimator $F$ over a certain sub-domain. For example, with the Hertz method ([26]) we can calculate the exact lowest eigenvalue of a given symmetric interval matrix. However this comes at the cost of $O(2^{n-1})$. Another method, the scaled Gerschgorin, use an extended version of Gerschgorin’s theorem, allows for more flexibility and the cost is only $O(n^2)$.

The effective calculation of the $\alpha$ values plays a critical role in the performance of the $\alpha$BB algorithm. We will discuss this subject in more detail in Chapter 5 where we introduce a refinement method which aims to further improve the $\alpha$ values obtained after the application of the scaled Gerschgorin method.
3 Enclosure of all index-1 saddle points of general nonlinear functions

3.1 Introduction

We consider the following problem: Given a function \( f : B \subseteq \mathbb{R}^n \to \mathbb{R}, f \in C^2 \) we want to find all the critical points, \( x^* \in B : \nabla f(x^*) = 0 \), of \( f \) for which the Hessian matrix \( \nabla^2 f(x^*) \) has eigenvalues \( \lambda_n < 0 < \lambda_{n-1} \leq \ldots \leq \lambda_1 \). As already mentioned in Chapter 2 such points are called Transition States (TSs) or index-1 saddle points and play an important role in determining rates of chemical transformations [83] and are also of interest in other areas including robotics and game theory.

The chapter is organized as follows: In Section 3.2, we give some basic notions and definitions related to interval matrices. In section 3.3, we introduce the general algorithmic framework. The regional tests are presented in Section 3.4. Local search over index-1 areas is discussed in Section 3.5. In Section 3.6 we characterize the tests in terms of their completeness. In Section 3.7, we address the algorithmic complexity of the problems that we aim to solve with the tests. The algorithm is applied to a number of examples in Section 3.8 and a brief summary is given in Section 3.9.

3.2 Preliminaries

We make extensive use of concepts from interval arithmetic throughout this paper. We introduce the necessary concepts in this section and the reader is referred to [D2] for further details.
We denote interval variables with lower case letters inside square brackets, \([x]\), and the corresponding lower and upper bounds as \(\underline{x}\) and \(\overline{x}\) respectively. Interval matrices are denoted with capital letters inside square brackets. An interval matrix is simply a matrix with interval entries instead of scalar entries. For example, a symmetric interval matrix is 
\[
[M] = \begin{bmatrix}
-3 & -2 \\
-2 & -0.5 & 0.5 \\
-0.5 & 0.5 & -4 & -3
\end{bmatrix}
\] 
The interval matrix \([M]\) can be interpreted as the infinite set of symmetric scalar matrices \(\{M : m_{ij} \in [m_{ij}] \text{ with } m_{ij} = m_{ji}\}\). For example, if \(M_1 = \begin{bmatrix}
-3 & 0.1 \\
0.1 & -3
\end{bmatrix}\) then \(M_1 \in [M]\). However if \(M_2 = \begin{bmatrix}
-3 & 0.2 \\
0.1 & -3
\end{bmatrix}\) then \(M_2 \notin [M]\).

Properties of scalar matrices, such as positive-definiteness and non-singularity are defined for interval matrices by requiring the property to hold for each scalar matrix belonging to the interval matrix. In this paper we are interested in symmetric interval matrices since we will calculate interval Hessian matrices over a given hyper-rectangular area, \([X] = ([x_1], [x_2], ..., [x_n])^T\). Therefore we deal only with real eigenvalues.

**Definition 3.2.0.1** (Positive definite interval matrix) An interval matrix \([M]\) is positive definite iff every \(M \in [M]\) is positive definite.

**Definition 3.2.0.2** (Non-singular interval matrix) An interval matrix \([M]\) is non-singular iff every \(M \in [M]\) is non-singular.

For a \(n \times n\) symmetric matrix \(M\) we denote with \(\lambda_i(M)\) the \(i\)-th largest eigenvalue of \(M\), with \(\lambda_n(M) \leq \lambda_{n-1}(M) \leq ... \leq \lambda_1(M)\). The eigenvalues of a symmetric interval matrix are defined as follows.

**Definition 3.2.0.3** (Eigenvalues of an interval matrix) The \(i\)th largest eigenvalue of a symmetric matrix \([M]\) is defined as the set \(\lambda_i([M]) = \{\lambda_i(M) : M \in [M]\}\).
Definition 3.2.0.4 (Index and coindex of scalar matrix) The index (coindex), \(\text{index}(M)\) (\(\text{coindex}(M)\)), of a symmetric matrix \(M\) is the number of strictly negative (positive) eigenvalues of \(M\).

Definition 3.2.0.5 (Index of symmetric interval matrix) The index of a symmetric interval matrix \([M]\) is defined as \(\min\{\text{index}(M) : M \in [M]\}\).

Similarly we define the coindex for symmetric interval matrices.

Definition 3.2.0.6 (Inertia of a symmetric scalar matrix) Given a symmetric matrix \(M\), the inertia of \(M\), \(\text{In}(M)\), is the triplet \((\pi(M), \nu(M), \delta(M))\) of the numbers of positive, negative and zero eigenvalues of \(M\) respectively.

Note that \(\pi\) and \(\nu\) are the same as the index and coindex respectively.

Definition 3.2.0.7 (Inertia of a symmetric interval matrix) Given a symmetric matrix \([M]\), the inertia of \([M]\), \(\text{In}([M])\), is defined as \(\min\{\text{In}(M) : M \in [M]\}\).

That is, \(\text{In}([M]) = \left( \min_{M \in [M]} \pi(M), \min_{M \in [M]} \nu(M), \min_{M \in [M]} \delta(M) \right)\).

Definition 3.2.0.8 (Norm of an interval matrix) We define the \(p\)-norm of an interval matrix, \([M]\), as \(\|\| [M] \|_p = \max\{\|M\|_p : M \in [M]\}\).

It is easy to verify that by this definition all the conditions required to hold for a norm of a scalar matrix also hold for the norm of an interval matrix.

3.3 Proposed approach

We use a branch-and-bound (B&B) algorithm and the formulation proposed in [85] (problem \(P\) below) in order to search for critical points:
\[
(P) \begin{align*}
\text{minimize} \quad & s \\
\text{subject to} \quad & \frac{\partial f(x)}{\partial x_i} - s \leq 0, \ i = 1, \ldots, n \\
& -\frac{\partial f(x)}{\partial x_i} - s \leq 0, \ i = 1, \ldots, n \\
& x_i \in [x_i, \bar{x}_i], \ i = 1, \ldots, n.
\end{align*}
\]

The interpretation behind the formulation of (P) (see also [85] for more details) is that if we perturb \( \frac{\partial f}{\partial x_i} \) by a positive value \( s \) then around a critical point \( x^* \) we would have \( \frac{\partial f}{\partial x_i}(x^*) - s \leq 0 \leq \frac{\partial f}{\partial x_i}(x^*) + s \), for \( i = 1, ..., n \). Thus the constraints in (P) describe regions which contain the critical points of \( f \). By gradually lowering the value of \( s \) these regions shrink until \( s = 0 \) where only the critical points remain. In case \( s \) becomes negative it means that the problem is infeasible (no critical points exist in the given domain).

However, aiming to focus the computational effort on the location of TSs, we introduce a number of tests which can be used to bound the number of negative and positive eigenvalues of an interval matrix. In a branch-and-bound algorithm, at any given iteration, valid lower and upper bounds on the global minimum are calculated over hyper-rectangular subsets \( R \) of the initial domain \( B \). By dividing each subset area improving lower and upper bounds are obtained. Whenever the lower bound of a given area is found to be greater than the best upper bound so far, the area is fathomed. We modify the approach by applying, prior to each bounding step, a test on the interval Hessian matrix, \( [\nabla^2 f(R)] \), calculated over \( R \) by the natural interval extension [22] of the second derivatives \( \frac{\partial^2 f}{\partial x_i \partial x_j} \). The interval Hessian can be seen as a superset of \( \{ \nabla^2 f(x) : x \in R \} \). If the test reveals that every matrix in \( [\nabla^2 f(R)] \) has index > 1 then we fathom the area \( R \). If the test reveals that every matrix in \( [\nabla^2 f(R)] \) is index-1 and coindex-\( n - 1 \) then we can choose to perform a local search, since it can be shown (cf. Section [22]) that this implies that there can be at most one TS in \( R \). If a TS is found during the local search, we fathom the area. Otherwise the test is inconclusive and we proceed to the next step of the
modified B&B algorithm. A flowchart of the proposed procedure is given in figure 3.1. A check to determine if zero is contained in the interval gradient is also applied at every iteration; if it is not the area is discarded. Note that index-1 saddle points, as defined, can only appear as isolated critical points of \( f \) (i.e. if \( x^* \) is a TS then \( \exists \epsilon > 0 \) s.t. \( \forall x \neq x^* \text{ with } |x - x^*| \leq \epsilon \nabla f(x) \neq 0 \)), otherwise the corresponding Hessian would be singular. Thus a surface can only have a finite number of TSs over a given hyper-rectangular domain.

**Figure 3.1: Algorithm flowchart.**

3.4 Regional tests for the existence of index-1 saddle points

In this section, we introduce five regional tests related to the presence of TSs. The tests can be used to identify regions that do not contain any TS, or regions that contain at most one TS. The computational complexity of each test is reported in
each case. If the tests are embedded within a branch-and-bound algorithm for the solution of Problem \( P \), the computational complexity of the solution of the convex lower bounding problem, which is NP-hard, dominates the overall cost. Furthermore, if the \( \alpha \text{BB} \) algorithm \([51, 9, 2, 1]\) is used, the interval Hessian matrix information required in the tests is readily available from the construction of the lower bounding problem and an efficient implementation can be developed with minimal effort devoted to the application of tests. Examples of the application of each test can be found in the Appendix.

3.4.1 The Gerschgorin test

We begin by developing a regional test based on the well-known theorem by Gerschgorin \([86]\).

**Theorem 3.4.1.1 (Gerschgorin)** Given a matrix \( M \in \mathbb{C}^{n \times n} \), define the radii

\[
r_i = \sum_{i \neq j} |m_{ij}|
\]

and the discs \( D_i(M) = \{ z \in \mathbb{C} : |z - m_{ii}| \leq r_i \} \). Then all the eigenvalues of \( M \) belong to the union \( G(M) = \bigcup D_i(M) \). Furthermore, if the union of \( k \) of the discs \( D_i(M) \) forms a disjoint set from the rest \( n - k \) discs, then it contains exactly \( k \) eigenvalues.

An interval extension for the first part of the above theorem was given in \([2]\) and used for the calculation of lower bounds for the eigenvalues of symmetric interval matrices. Here we are interested in the second part of Gerschgorin’s theorem, on counting the eigenvalues in disjoint sets. The extension in \([2]\) is also valid for the second part of the theorem.

**Theorem 3.4.1.2 (Interval extension)** Given a \( n \times n \) symmetric interval matrix \([M]\), define the radii

\[
r_i([M]) = \sum_{j=1}^{n} \max\{|m_{ij}|, |m_{ij}|\}
\]

and the intervals \( D_i([M]) = \ldots \)

32
\[ [m_{ii} - r_i([M]), m_{ii} + r_i([M])] \text{ for } i = 1, 2, \ldots n. \] Then all the eigenvalues of every \( M \in [M] \) belong to the union \( G([M]) = \bigcup_i D_i([M]) \). Furthermore, if the union of \( k \) of the intervals \( D_i([M]) \) forms a disjoint set from the other \( n - k \) intervals, then it contains exactly \( k \) eigenvalues of every \( M \in [M] \).

**Proof.** Based on the definition of the intervals \( D_i([M]) \), we have that \( \forall M \in [M], D_i(M) \subseteq D_i([M]) \) for \( i = 1, 2, \ldots, n \). Thus \( \forall M \in [M], G(M) \subseteq G([M]) \Rightarrow \forall M \in [M], \sigma(M) \in G([M]) \) where \( \sigma(M) \) is the spectrum of \( M \). To prove the second part of the theorem, assume, without loss of generality, that the union \( U_k = \bigcup_{i=1}^k D_i([M]) \), for some \( k \in \{1, 2, \ldots, n\} \), is disjoint from \( U_{n-k} = \bigcup_{i=k+1}^n D_i([M]) \). Then, \( \forall M \in [M], \bigcup_{i=1}^k D_i(M) \subseteq U_k \) and \( \bigcup_{i=k+1}^n D_i(M) \subseteq U_{n-k} \) and therefore by Theorem 3.4.1.1 exactly \( k \) eigenvalues of \( M \) belong to \( U_k \).

We give a pseudocode for a test based on Theorem 3.4.1.2, which we call the Gerschgorin test, in Algorithm III. Regions for which the interval Hessian contains no negative disks (convex areas), or where a set of more than one discs lie on the negative side and are disjoint from the rest, are removed (lines 16-17 and 21-29 in Alg. III). By “discs” here we mean the intervals \( D_i([M]) \). Regions with one negative eigenvalue and all the other positive may also be identified (lines 18-19). Notice that the Gerschgorin test may be inconclusive even for a scalar matrix.
Algorithm 1 Pseudocode for Gerschgorin test: $O(n^2)$

1: Set $n = \dim([M])$, $L = +\infty$, List of neg. discs: $\text{nd} = \emptyset$, Total number of neg. discs: $\text{nnd} = 0$.

2: for $i = 1 : n$ do

3: $[d_i] = [m_{ii}]$.

4: for $j = 1 : n$, $j \neq i$ do

5: $m = \max\{|m_{ij}|, |\overline{m_{ij}}|\}$

6: $[d_i] = [d_i] + [-m, m]$.

7: end for

8: if $d_i < 0$ then

9: $\text{nd} = \text{nd} \cup \{d_i\}$ and $\text{nnd}++$

10: else if $d_i < L$ then

11: Set $L = d_i$.

12: end if

13: end for
14: if  \( \text{nnd} \leq 1 \) and \( L < 0 \) then

15:  Stop. Test is inconclusive.

16: else if ( \( \text{nnd} == 0 \) or \( \text{nnd} > 1 \) ) and \( L \geq 0 \) then

17:  Fathom area.

18: else if \( \text{nnd} == 1 \) and \( L > 0 \) then

19:  Optional: Local search. (see Section 3.5)

20: else

21:  Sort \( \text{nd} \) w.r.t. the upper bounds in decreasing order.

22:  for  \( i = 1 : \text{nnd} \) do

23:     if \( \text{nd}[i] \) is \( \geq L \) then

24:         \( \text{nnd}-- \).

25:         \( L = \min\{L, \text{nd}[i]\} \)

26:     end if

27:  end for

28:  if \( \text{nnd} > 1 \) then

29:      Fathom area.

30:  else

31:      Stop. Test is inconclusive.

32:  end if

33: end if

Note that the complexity given for Algorithm 11 (and any other regional test algorithm) is given without taking in account the optional choice of local search.
3.4.2 Gerschgorin test example

Consider the matrix,

\[
[M] = \begin{bmatrix}
-3, & -2 \\
-0.5, & 0.5 \\
-1, & 1
\end{bmatrix}
\begin{bmatrix}
-0.5 & 0.5 \\
-4 & -3 \\
0.5 & 1
\end{bmatrix}
\begin{bmatrix}
-1, & 1 \\
0.5 & 1 \\
2 & 5
\end{bmatrix}
\begin{bmatrix}
-3, & 2 \\
-0.5, & 0.5 \\
-1, & 1
\end{bmatrix}
\begin{bmatrix}
-0.5 & 0.5 \\
-4 & -3 \\
0.5 & 1
\end{bmatrix}
\begin{bmatrix}
-1, & 1 \\
0.5 & 1 \\
2 & 5
\end{bmatrix}
\] .

Then we have that \( r_1 = r_2 = r_3 = 1.5 \) and \( D_1 = [-4.5, -0.5] \), \( D_2 = [-5.5, -1.5] \), \( D_3 = [1, 4.5] \). \( D_1 \) and \( D_2 \) lie on the negative side while \( D_3 \) on the positive. By Theorem 3.4.1.2 we know that every \( M \in [M] \) has exactly two negative eigenvalues \( \in D_1 \cup D_2 \) and one positive \( \in D_3 \).

3.4.3 Recursive Inertia (RecIn) test

Based on Haynsworth’s theorem \[23, \text{ f} \] we can construct algorithms for obtaining bounds on the number of negative and positive eigenvalues of interval matrices.

**Theorem 3.4.3.1 (Haynsworth)** Given a symmetric matrix \( M \) partitioned in the form, \( M = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \) and assuming \( A \) is non-singular, then, \( \text{In}(M) = \text{In}(A) + \text{In}(C - B^T A^{-1} B) \).

Haynsworth’s Theorem can be extended in the interval case as follows:

**Theorem 3.4.3.2 (Interval extension)** Given a symmetric interval matrix \([M]\) partitioned in the form, \([M] = \begin{bmatrix} [A] & [B] \\ [B]^T & [C] \end{bmatrix} \) and assuming \([A]\) is non-singular, then, \( \text{In}([M]) \geq \text{In}([A]) + \text{In}([C] - [B]^T [A]^{-1} [B]) \).
Before we proceed with the proof we note that when we multiply two interval matrices, \([A]\) and \([B]\), we have that \([C] = [A][B] \supseteq \{AB : A \in [A] \text{ and } B \in [B]\}\).

The proof of Theorem 3.4.3.2 is straightforward:

**Proof.** Let \([S] = [C] - [B]^T[A]^{-1}[B]\) and \(S_x = \{C - B^T A^{-1} B : A \in [A], B \in [B], C \in [C]\}\) with \([S] \supseteq S_x\). Then

\[
\text{In}([C]) = \min_{A \in [A], B \in [B], C \in [C]} \text{In}(A) + \text{In}(C - B^T A^{-1} B) \tag{3.1}
\]

\[
\geq \min_{A \in [A]} \text{In}(A) + \min_{S \in S_x} \text{In}(S) \tag{3.2}
\]

\[
\geq \min_{A \in [A]} \text{In}(A) + \min_{S \in [S]} \text{In}(S) = \text{In}([A]) + \text{In}([S]). \tag{3.3}
\]

\(\square\)

We can make use of Haynsworth’s theorem recursively, as shown by Cottle [9]. Cottle considers scalar matrices and chooses \(A\) to be a single non-zero entry in the diagonal. By interchanging corresponding rows and columns simultaneously, thus not affecting the eigenvalues, we bring the selected entry \(A\) to the top left position of the matrix. We note the sign of \(A\), we then calculate \(C - B^T A^{-1} B\) (the Schur complement of \(A\) in \(M\)), and repeat. If all the elements in the diagonal are zero, we are either left with a zero matrix or we can choose \(A\) to be of the form \(\begin{pmatrix} 0 & a \\ a & 0 \end{pmatrix}\). In this way, we can always calculate the complete inertia of a scalar matrix.

A straightforward adaptation of this recursive scheme for interval matrices \([M]\) is simply to scan the diagonal for an interval that does not contain zero and re-arrange \([M]\) as appropriate, calculate the interval Schur complement \([C] - [B]^T[A]^{-1}[B]\)
and repeat. We should give priority to negative intervals. If at any point all the
diagonal interval elements contain zero, then we cannot proceed further with the
analysis and stop. Note that in the interval case, each time we find a negative (resp.
positive) interval in the diagonal of a subsequent Schur complement, this means that
all the scalar matrices contained in the initial interval matrix have a further negative
(resp. positive) eigenvalue. In a similar manner, Meyer and Swartz [57] used Schur’s
formula, \( \det(M) = \det(A)\det(C - B^T A^{-1} B) \), for a convexity test applied to interval
matrices (such a test was mentioned in [9] for scalar matrices) along with a branch-
and-bound method. In Algorithm 2 we give a pseudocode for the proposed recursive
inertia test, RecIn.
Algorithm 2 Pseudocode for RecIn algorithm: $O(n^3)$

1: Set $n = \text{dim}([M])$. Initialize neg=0, pos=0 (number of negative and positive (interval) eigenvalues).

2: Search for a diagonal interval $[m_{aa}]$ with $0 \not\in [m_{aa}]$. Give priority to negative intervals.

3: if none found then

4: Stop, test is inconclusive.

5: else

6: $pos = pos + 1$, if $[m_{aa}] > 0$ or $neg = neg + 1$, if $[m_{aa}] < 0$

7: if $neg > 1$ or $pos == n$ then

8: Fathom area.

9: else if $neg == 1$ and $pos == n - 1.$ then

10: Optional: Local search.

11: else if $\text{dim}([M]) > 1$ then

12: Calculate the interval Schur complement of $[A] = [m_{aa}]$ in $[M]$, set $[M]$ to the Schur complement and repeat from step 2.

13: else

14: Test is inconclusive.

15: end if

16: end if
3.4.4 RecIn test example

Consider the matrix,

\[
[M] = \begin{bmatrix}
[-0.08, 0.88] & [0.30, 0.55] & [0.65, 0.84] \\
[0.30, 0.55] & [-0.89, -0.69] & [-0.14, 0.23] \\
[0.65, 0.84] & [-0.14, 0.23] & [-0.86, -0.74]
\end{bmatrix}.
\]

At the first step we have, \([A(0)] = [-0.89, -0.69] < 0, [B(0)] = [0.30, 0.55] [-0.14, 0.23] \]
and \([C(0)] = \begin{bmatrix}
[-0.08, 0.38] & [0.65, 0.34] \\
[0.65, 0.34] & [-0.86, -0.74]
\end{bmatrix}.

For the second step, the (interval) Schur complement is

\[
[M(1)] = [C(0)] - \frac{1}{[A(0)]} [B(0)]^T [B(0)] = \begin{bmatrix}
[0.02, 0.81] & [0.23, 0.33] \\
[0.23, 0.83] & [-0.86, -0.66]
\end{bmatrix}
\]

and we have \([A(1)] = [-0.86, -0.66] < 0 \) which is the second negative interval we find. Thus we conclude that every \(M \in [M]\) has at least two negative eigenvalues.

3.4.5 Extended RecIn test

The RecIn test cannot proceed if all diagonal elements of the initial input matrix or of a subsequent Schur complement contain zero. We extend the RecIn algorithm to overcome this issue.

The following Lemma was given in [32].

**Lemma 3.4.5.1** Given a \(n \times n\) symmetric interval matrix \([M]\) define the symmetric interval matrices

\[
[L] = \{l_{ii} = m_{ii} \text{ and } [l_{ij}] = [m_{ij}] \text{ for } i \neq j\} \tag{3.4}
\]
and

\[ [U] = \{ u_{ii} = \overline{m}_{ii} \text{ and } [u_{ij}] = [m_{ij}] \text{ for } i \neq j \}. \quad (3.5) \]

Then \( \forall M \in [M] \), there are \( L \in [L] \) and \( U \in [U] \) such that,

\[ \lambda_i(L) \leq \lambda_i(M) \leq \lambda_i(U) \text{ for } i = 1, 2, \ldots, n. \quad (3.6) \]

**Corollary 3.4.5.2** Given a \( n \times n \) symmetric interval matrix \([M]\) and defining the matrices \([L]\) and \([U]\) as above, then \( \forall M \in [M] \),

\[ \min_{U \in [U]} \nu(U) \leq \nu(M) \leq n - \min_{L \in [L]} \pi(L). \quad (3.7) \]

**Proof.** Lemma 3.4.5.1 implies that \( \forall M \in [M] \), there are \( L \in [L] \) and \( U \in [U] \) such that

\[ \nu(U) \leq \nu(M) \leq \nu(L). \quad (3.8) \]

Therefore \( \forall M \in [M] \) we have,

\[ \min_{U \in [U]} \nu(U) \leq \nu(M) \leq \max_{L \in [L]} \nu(L). \quad (3.9) \]

Also, \( n - \min_{L \in [L]} \pi(L) \geq \max_{L \in [L]} \nu(L) \) (the inequality stems from the fact that the matrix
might have zero eigenvalues) and hence finally,

\[
\min_{U \in [U]} \nu(U) \leq \nu(M) \leq n - \min_{L \in [L]} \pi(L). \square \tag{3.10}
\]

\[
\min_{L \in [L]} \pi(L) \leq \pi(M) \leq n - \min_{U \in [U]} \nu(U), \forall M \in [M]. \tag{3.11}
\]

In a similar way, we can show that,

Based on corollary 3.4.5.2, we introduce algorithms RecIn\_U and RecIn\_L. RecIn\_U makes use of the [U] part of the initial input matrix [M] and of each subsequent Schur complement and is used to calculate a lower bound of \( \min_{U \in [U]} \nu(U) \). In analogy, RecIn\_L makes use of the [L] part and is used to calculate a lower bound of \( \min_{L \in [L]} \pi(L) \). Thus, by (3.10) and (3.11), we obtain bounds for \( \nu([M]) \) and \( \pi([M]) \). We give the pseudocode for the RecIn\_U in Algorithm 3 and then the extended recursive inertia test, xRecIn in Algorithm 4. We omit the pseudocode for RecIn\_L since it is easy to derive it from RecIn\_U.
Algorithm 3 Pseudocode for the RecIn_U algorithm: $O(n^3)$

1: Set $n = \text{dim}(\text{[}M\text{])}$. Initialize $neg = 0$, $pos = 0$ (number of negative and positive (interval) eigenvalues).


3: Search for a diagonal element $m_{aa} \neq 0$. Give priority to negative elements.

4: if none found then

5: Search for an off-diagonal element $[m_{ij}] \neq 0$.

6: if none found then

7: Stop, test is inconclusive.

8: else

9: $neg = neg + 1$. If $neg > 1$, return $neg$.

10: Calculate the interval Schur complement of $[A] = \begin{bmatrix} 0 & [m_{ij}] \\ [m_{ji}] & 0 \end{bmatrix}$ in $[M]$, set $[M]$ equal to the Schur complement and repeat from step 2.

11: end if

12: else

13: $neg = neg + 1$, if $[m_{aa}] < 0$

14: if $neg > 1$ then

15: return $neg$.

16: else if $\text{dim}([M]) > 1$ then

17: Calculate the interval Schur complement of $[A] = [m_{aa}]$ in $[M]$, set $[M]$ equal to the Schur complement and repeat from step 2.

18: else

19: return $neg$.

20: end if

21: end if
Note that for the calculation of the Schur complement in step 10 of the RecIn_U algorithm, the inverse of $[A]$ is simply $[A]^{-1} = \begin{bmatrix} 0 & 1/m_{ij} \\ 1/m_{ji} & 0 \end{bmatrix}$.

**Algorithm 4** Pseudocode for the xRecIn test: $O(n^3)$

1: Set $n = \dim([M])$.
2: $\nu_U = \text{RecIn}_U([M])$.
3: if $\nu_U > 1$ then
   4: Fathom area.
5: else if $\nu_U == 1$ then
   6: $\pi_L = \text{RecIn}_\pi([M])$.
7: if $\pi_L == n - 1$ then
   8: Local search.
9: end if
10: else
11: Stop. Test is inconclusive.
12: end if

### 3.4.6 xRecIn test example

Consider this simple example,

$$[M] = \begin{bmatrix} [-2, -1] & 0 & 0 \\ 0 & [-1, 1] & [2, 3] \\ 0 & [2, 3] & [-1, 1] \end{bmatrix}.$$

If we were to apply RecIn, we would have at the first step: $[A^{(0)}] = [-2, -1] < 0$. 

44
However, at the next step, the (interval) Schur complement would be $[M^{(1)}] = 
abla$

$$
\begin{bmatrix}
-1, 1 & 2, 3 \\
2, 3 & -1, 1
\end{bmatrix}
$$
and RecIn would not be able to proceed. Considering Corollary 3.4.5.2, we can set $[M^{(1)}] = 
abla$

$$
\begin{bmatrix}
1, 1 & 2, 3 \\
2, 3 & 1, 1
\end{bmatrix}
$$
and now we have $[A^{(1)}] = [1, 1] > 0$ and the next Schur complement, $[M^{(2)}] = [1, 1] - [4, 9]/[1, 1] = [-8, -3] < 0$. Thus every $M \in [M]$ has at least two negative eigenvalues.

### 3.4.7 2 × 2 Inertia test

Another possible way to make use of Theorem 3.4.3.1 for our purpose is to choose $[A]$, in $[M] = 
abla$

$$
\begin{bmatrix}
[A] & [B] \\
[B]^T & [C]
\end{bmatrix},
$$
to be any of the 2 × 2 diagonal sub-matrices of $[M]$, $[A_{ij}] = 
abla$

$$
\begin{bmatrix}
m_{ii} & m_{ij} \\
m_{ji} & m_{jj}
\end{bmatrix}.
$$
The maximum eigenvalue, $\lambda_{ij} = \max_{A_{ij} \in [A_{ij}]} \lambda_1(A_{ij})$, of each of these matrices is

$$
\lambda_{ij} = \frac{m_{ii} + m_{jj} + \sqrt{(m_{ii} - m_{jj})^2 + 4 \max\{m_{ij}^2, m_{ji}^2\}}}{2}. \tag{3.12}
$$

If $\lambda_{ij} < 0$ for any of the sub-matrices then by Theorem 3.4.3.1 we know that every $M \in [M]$ has at least two negative eigenvalues and thus we can fathom the corresponding area. In Algorithm 5 we give a pseudocode for this test to which we refer as the 2 × 2 inertia test.
Algorithm 5 Pseudocode for the $2 \times 2$ inertia test: $O(n^2)$

1: Set $n = \dim([M])$.

2: for $i=1:n$ do

3:     for $j>i:n$ do

4:         $\lambda = \frac{m_{ii} + m_{jj} + \sqrt{(m_{ii} - m_{jj})^2 + 4 \max\{m_{ij}^2, m_{ji}^2\}}}{2}$

5:         if $\lambda < 0$ then

6:             Fathom area.

7:         end if

8:     end for

9: end for

Note that the $2 \times 2$ inertia test does not remove TSs and minima and that it may be inconclusive even for a scalar matrix. However, it is computationally cheap and it is easy to implement. Furthermore, it is straightforward to show that this test is more effective than the Gerschgorin test in identifying non-TS areas. More formally we have the following:

**Proposition 3.4.7.1** Given a $n \times n$ symmetric interval matrix $[M]$, if the Gerschgorin test reveals that $\text{index}([M]) > 1$ then so does the $2 \times 2$ inertia test. The reverse is not always true.

**Proof.** Since the Gerschgorin test reveals that $\text{index}([M]) > 1$, this implies that there are at least two rows of $[M]$, $i$ and $j$, for which

$$\bar{m}_{ii} + \sum_{k=1, k \neq i}^{n} \max\{|m_{ik}|, |\bar{m}_{ik}|\} < 0 \text{ and } \bar{m}_{jj} + \sum_{k=1, k \neq j}^{n} \max\{|m_{jk}|, |\bar{m}_{jk}|\} < 0.$$
This implies that

\[ m_{ii} + \max\{|m_{ij}|, |m_{ji}|\} < 0 \quad \text{and} \quad m_{jj} + \max\{|m_{ji}|, |m_{ij}|\} < 0. \quad (3.13) \]

From (3.13) and Theorem 3.4.1.2 we have that for \( [M_{ij}] = \begin{bmatrix} m_{ii} & m_{ij} \\ m_{ji} & m_{jj} \end{bmatrix} \), \( \lambda([M_{ij}]) < 0 \) and since the \( 2 \times 2 \) inertia test provides the exact upper bound of \( \lambda([M_{ij}]) \), it also reveals that index\( ([M]) > 1 \).

Finding a counter-example to show that the reverse is not always true is easy. For example:

\[
[M] = \begin{bmatrix}
-2 & -1 & 0 & 100 \\
0 & -2 & -1 & 0 \\
0 & 100 & 0 & 0 \\
0 & 30 & 0 & 55 \\
0 & 89 & 0 & 69 \\
0 & 14 & 0 & 23 \\
0 & 30 & 0 & 55 \\
0 & 14 & 0 & 23 \\
0 & 86 & 0 & 74
\end{bmatrix},
\]

The sub-matrix \([M_{12}] = \begin{bmatrix} -2 & -1 \\ -1 & -2 \end{bmatrix} \) clearly has two negative interval eigenvalues. The \( 2 \times 2 \) inertia test would identify this. However, because of the large entry, \( m_{13} = 100 \), the Gerschgorin discs would form one joint set with negative lower bound and positive upper bound and thus the Gerschgorin test would be inconclusive even for this very simple case.

**3.4.8 2 × 2 inertia test example**

Consider the matrix,

\[
[M] = \begin{bmatrix}
-0.08 & 0.88 & 0.30 & 0.55 & 0.65 & 0.84 \\
0.30 & 0.55 & -0.89 & -0.69 & -0.14 & 0.23 \\
0.65 & 0.84 & -0.14 & 0.23 & -0.86 & -0.74
\end{bmatrix}.
\]
At step 1:
\[
M_{12} = \begin{bmatrix}
-0.08, 0.88 & [0.30, 0.55] \\
[0.30, 0.55] & -0.89, -0.69
\end{bmatrix}, \lambda_{12} = 1.05.
\]

At step 2:
\[
M_{13} = \begin{bmatrix}
-0.08, 0.88 & [0.65, 0.84] \\
[0.65, 0.84] & -0.86, -0.74
\end{bmatrix}, \lambda_{13} = 1.24.
\]

Finally, at step 3:
\[
M = \begin{bmatrix}
-0.89, -0.69 & [-0.14, 0.23] \\
[-0.14, 0.23] & -0.86, -0.74
\end{bmatrix}, \lambda_{23} = -0.48 < 0 \text{ and therefore we conclude that every } M \in [M] \text{ has at least two negative eigenvalues.}
\]

3.4.9 Rohn test

The last test we present is based on Rohn’s method [32] which is derived from the interval extension of Weyl’s inequality [18].

**Theorem 3.4.9.1 (Weyl)** Given \( n \times n \) symmetric (scalar) matrices \( C \) and \( E \), then

\[
\lambda_k(C) + \lambda_n(E) \leq \lambda_k(C + E) \leq \lambda_k(C) + \lambda_1(E), \text{ for } k = 1, 2, \ldots, n. \tag{3.14}
\]

where for any matrix \( M \), \( \lambda_n(M) \leq \ldots \leq \lambda_1(M) \). Any given interval matrix \([M]\) can be written as \( C + [E] \) where \( c_{ij} = (\overline{m}_{ij} + \underline{m}_{ij})/2 \) and \([e_{ij}] = [\overline{m}_{ij} - c_{ij}, \overline{m}_{ij} - c_{ij}]\). Calculating lower and upper bounds, \( \underline{\lambda}_n \) and \( \overline{\lambda}_1 \), for \( \lambda_n([E]) = \{ \lambda_n(E) : E \in [E] \} \) and \( \lambda_1([E]) = \{ \lambda_1(E) : E \in [E] \} \), respectively, leads to the theorem by Rohn:

**Theorem 3.4.9.2** (Rohn) Given a symmetric interval matrix \([M] = C + [E] \), then

\[
\lambda_k(C) + \underline{\lambda}_n \leq \lambda_k(C + [E]) \leq \lambda_k(C) + \overline{\lambda}_1, \text{ for } k = 1, 2, \ldots, n. \tag{3.15}
\]
Note that because \( C \) has been defined as the center matrix of \([M]\), \( \lambda_2 = -\lambda_1 \) and also that the widths of the intervals \( \lambda_k([M]) \) are all the same. We can calculate \( \lambda_n \) (and \( \lambda_{n-1} \)) using a number of methods (see \([2, 80])\), the simplest being the interval extension of Gerschgorin’s theorem \( (O(n^2)) \) and the most expensive being the Hertz-Rohn method \( (O(2^{n-1})) \) \([26, 75, 27]\). The Rohn test is summarized in Algorithm 6.

**Algorithm 6** Pseudocode for Rohn’s test: \( O(n^2) - O(2^{n-1}) \)

1: Set \( n = \text{dim}([M]) \).

2: Calculate \([\lambda_n]\) and \([\lambda_{n-1}]\) for \([M]\) using Theorem 4.4.0.1 and an eigenvalue bounding method.

3: if \( \lambda_n \geq 0 \) then

4: Fathom area (convex area).

5: else if \( \lambda_n < 0 \) and \( \lambda_{n-1} > 0 \) then

6: Optional: Local search.

7: else if \( \lambda_{n-1} \leq 0 \) then

8: Fathom area.

9: else

10: Test is inconclusive.

11: end if

### 3.4.10 Rohn test example

As an example for the Rohn test consider again the matrix,
The center matrix is

\[
C = \begin{bmatrix}
0.4 & 0.425 & 0.745 \\
0.425 & -0.79 & 0.045 \\
0.745 & 0.045 & -0.8
\end{bmatrix},
\]

and

\[
[E] = \begin{bmatrix}
[-0.44, 0.44] & [-0.125, 0.125] & [-0.095, 0.095] \\
[-0.125, 0.125] & [-0.10, 0.10] & [-0.185, 0.185] \\
[-0.095, 0.095] & [-0.185, 0.185] & [-0.06, 0.06]
\end{bmatrix}.
\]

Since, \(\lambda_2([M]) \leq \lambda_2(C) + \lambda_1([E]) = -0.83 + 0.66 < 0\), all matrices in \([M]\) have at least two negative eigenvalues.

### 3.5 Index-1 areas

In section 3.3 we stated that hyper-rectangular areas where every matrix is index-1 and coindex-\(n-1\) has at most one TS. We give a proof of this statement here. The proof is straightforward and we state it for completeness.

**Theorem 3.5.0.1** Assume we have a function \(f \in C^2, f : B \subseteq \mathbb{R}^n \rightarrow \mathbb{R}\) where B is an open hyper-rectangular box. If \(\nabla^2 f(x)\) has index 1 and coindex \(n-1\) \(\forall x \in B\) then there is at most one TS in B.

**Proof.** If \(f\) has any critical points in \(B\) then by the assumption that \(\nabla^2 f(x), x \in B\) is index-1 and coindex-\(n-1\), they would be TSs. Now assume that \(x_1, x_2 \in B\) with
$x_1 \neq x_2$ are critical points of $f$. Then, by the mean value theorem

$$\nabla f(x_2) = \nabla f(x_1) + \nabla^2 f(\xi)(x_2 - x_1), \quad (3.16)$$

for some $\xi$ between $x_1$ and $x_2$ and since $B$ is a hyper-rectangle $\Rightarrow \xi \in B$. However,

$$\nabla f(x_1) = \nabla f(x_2) = 0$$

and therefore

$$\nabla^2 f(\xi)(x_2 - x_1) = 0 \Rightarrow \nabla^2 f(\xi) \text{ singular}, \quad (3.17)$$

which contradicts our assumption.

In practice the interval Hessian, $[\nabla^2 f(B)]$, over $B$ would be an overestimation of $\{\nabla^2 f(x) : x \in B\}$. Hence, if the assumptions of Theorem 3.5.0.1 are true for $[\nabla^2 f(B)]$, they are also true for $\{\nabla^2 f(x) : x \in B\}$. At such cases we can perform a local search using Newton’s method for the unique critical point and if we locate a solution we can save this solution and fathom the corresponding area.

### 3.6 Completeness of the tests

The proposed tests take as input a symmetric interval matrix $[M]$ and aim to verify if

$\exists M \in [M]$ such that, index($M$) = 1 and coindex($M$) = $n - 1$. Moreover, we might, optionally, try to verify if $\forall M \in [M]$, index($M$) = 1 and coindex($M$) = $n - 1$. The $2 \times 2$ inertia test is an exception since it attempts to verify if index($M$) > 1 $\forall M \in [M]$. In any case a test might fail to provide a definitive answer and thus be inconclusive. By considering under what circumstances a test may be inconclusive, we can classify the proposed tests using the following definitions.

**Definition 3.6.0.2** (Complete test) A test is called complete if it is never incon-
Definition 3.6.0.3 (ε-complete test) A test is called ε-complete if ∀ n × n non-singular, scalar matrix C, ∃ ε > 0 such that ∀ [E] with ||[E]|| < ε the test is not inconclusive for C + [E] as input.

Definition 3.6.0.4 (Incomplete test) A test is called incomplete if it is not ε-complete.

We note that in the above definitions, for any test, we assume infinite-precision arithmetic and also that we know the maximum number of steps a priori.

The Gerschgorin and 2 × 2 inertia tests are incomplete since they can be inconclusive even for scalar matrices. The recursive inertia test is also incomplete since it cannot deal with matrices where all the diagonal elements contain zero. The extended recursive inertia test and Rohn test are ε-complete. We do not know of any method that can result in a complete test or if a complete test is even possible. In the next section we prove that this is an NP-hard problem.

We could attempt to construct a complete test with the following reasoning. The Hertz-Rohn method [26] gives the exact lower and upper bounds of the smallest and largest eigenvalue, respectively, of any symmetric interval matrix [M]. It does so by calculating the smallest and largest eigenvalues over a finite number (2^{n-1}) of scalar matrices M ∈ [M]. The entries of these scalar matrices are either m_{ij} or m_{ij}. Based on this, we might ask whether it is possible to have an a priori way of identifying a finite number of matrices in any given symmetric interval matrix [M], so that we can find the exact lower bound of index([M]). We can show that, unlike the case of calculating the extreme eigenvalues, this is not possible if each element m_{ij} is chosen as a function only of [m_{ij}]. This is expressed more formally in the following proposition.
Proposition 3.6.0.5 Define a set \( S = \{S_1, S_2, ..., S_s\} \) where each \( S_k, k = 1, 2, ..., s, \)
is a set of functions \( m^{(k)}_{ij} : \mathbb{R}^2 \to \mathbb{R} \) for \( i, j = 1, 2, ..., n \) with \( i \leq j \) such that
\( l \leq m^{(k)}_{ij}(l, u) \leq u \) for any \( l, u \in \mathbb{R} \) with \( l \leq u \). Given a \( n \times n \) symmetric interval matrix \([M]\), the set \( S \) defines a set, \( S([M]) \), of scalar matrices \( M_1, M_2, ..., M_s \in [M] \).

For any choice of \( S \) there is always a matrix \([M]\) for which the set \( S([M]) \) fails to identify correctly the lower bound of \( \text{index}([M]) \). That is, \( \exists M^* \in [M] \) such that
\( \text{index}(M^*) < \min\{\text{index}(M) : M \in S(M)\} \).

Proof. Consider a matrix of the form \([M] = \begin{bmatrix} 1 & 1 & b \\ 1 & 2 & c \\ b & c & d_3 \end{bmatrix}\). From theorem 3.4.3.1.

\( \forall M \in [M] \) we have

\[
\ln(M) = \ln(1) + \ln(1) + \ln(-c^2 + 2cb - 2b^2 + d_3) \text{ with } c \in [c]. \tag{3.18}
\]

The roots of \( h(c) = -c^2 + 2cb - 2b^2 + d_3 \) are given by \( c_1^*, c_2^* = b \pm \sqrt{d_3 - b^2} \). The distance between the roots is \( d(c_1^*, c_2^*) = 2\sqrt{d_3 - b^2} \) and the midpoint is \( b \). The function \( h \) is concave and thus positive in \((c_1^*, c_2^*)\) and negative outside of \([c_1^*, c_2^*]\).

For a given set \( S, |S| = s \) the scalar matrices \( M_1, M_2, ..., M_s \in S([M]) \) will have a corresponding entry \( c_1, c_2, ..., c_s \in [c] \). By appropriately choosing values for \( b \) and \( d_3 \), for example, \( b = (c_k + c_{k+1})/2 \) and \( b^2 < d_3 < d(c_k, c_{k+1})^2/4 + b^2 \) (such that \( 0 < 2\sqrt{d_3 - b^2} < d(c_k, c_{k+1}) \)), we would have that \( \forall M_i \in S[M] \) \( \text{index}(M_i) = 1 \). However, the matrix \( M^* \in [M] \) with \( c = b \) would have \( \text{index}(M^*) = 0 \).
Corollary 3.6.0.6 There is no choice of $S$ such that for any $n \times n$ symmetric interval matrix $[M]$, $S([M])$ provides correct bounds for $\lambda_i([M])$, $i = 1, 2, \ldots, n$.

Proof. If such a choice of $S$ existed then it would also allow the correct calculation of the bounds for the index of any symmetric interval matrix, which contradicts Proposition 3.6.0.5. \qed

A summary with the characteristics of each test is given in Table 3.1.

<table>
<thead>
<tr>
<th>Test</th>
<th>Completeness</th>
<th>Complexity</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gerschgorin</td>
<td>Incomplete</td>
<td>$O(n^2)$</td>
<td>Effective when diagonal entries are large with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>respect to off diagonal</td>
</tr>
<tr>
<td>2x2 Inertia</td>
<td>Incomplete</td>
<td>$O(n^2)$</td>
<td>Does not remove minima.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Simple to implement.</td>
</tr>
<tr>
<td>Rohn</td>
<td>$\epsilon$-complete</td>
<td>$O(n^2) - O(2^{n-1})$</td>
<td>Requires direct calculation of eigenvalues.</td>
</tr>
<tr>
<td>RecIn</td>
<td>Incomplete</td>
<td>$O(n^3)$</td>
<td>Not applicable when all diagonal entries contain zero.</td>
</tr>
<tr>
<td>xRecIn</td>
<td>$\epsilon$-complete</td>
<td>$O(n^3)$</td>
<td>Can handle cases where all diagonal entries contain zero.</td>
</tr>
</tbody>
</table>
3.7 Algorithmic complexity

In this section we investigate the algorithmic complexity of the problems that we aim to solve with the algorithms given in section 3.4 that is, identifying a TS matrix or a non-TS matrix. By TS and non-TS we mean, given a symmetric interval matrix $[M]$, identifying if $\forall M \in [M]$, $\text{index}(M) = 1$ and $\text{coindex}(M)=n-1$ or if $\nexists M \in [M]$ with $\text{index}(M) = 1$ and $\text{coindex}(M) = n-1$ respectively. Rohn [74] proved that checking positive definiteness of an interval matrix is an NP-hard problem.

**Theorem 3.7.0.7** The decision problem:

- **Instance:** A $n \times n$ symmetric interval matrix $[M]$.
- **Question:** Is $[M]$ positive definite?

is NP-hard.

The problem of positive definiteness can be trivially reduced in polynomial time to the following problem.

**Corollary 3.7.0.8** The decision problem:

- **Instance:** A $n \times n$ symmetric interval matrix $[M]$ and integer $k \in \{1, 2, ..., n\}$
- **Question:** Is $\text{index}([M]) = k$ and $\text{coindex}([M]) = n - k$?

is NP-hard.
Proof. Simply consider the block interval matrix

\[
[M] = \begin{bmatrix}
D & 0 \\
0 & [A]
\end{bmatrix}
\]

(3.19)

where \(D\) can be any diagonal \(k \times k\) matrix with all the diagonal entries being negative and \([A]\) a symmetric interval matrix. Checking if \(\text{index}([M]) = k\) and \(\text{coindex}([M]) = n - k\) is equivalent to checking if \([A]\) is positive definite.

Therefore identifying a TS matrix is NP-hard. With the help of Haynsworth’s theorem and using the same reduction as in [63], used for proving that checking the positive semi-definiteness of an interval matrix is NP-hard, we can prove the NP-hardness of identifying a non-TS matrix. First we give the following lemma from [63].

**Lemma 3.7.0.9** The decision problem:

**Instance:** A positive integer \(m\) and an \(m\)-dimensional vector \(a\), \(\|a\|_2 \leq 0.1\) with rational positive entries.

**Question:** Determine whether \(\max \{ z^T (I_m - aa^T)z : z \in \mathbb{R}^m, \|z\|_\infty \leq 1\} \leq m - 1/d^2(a) \) where \(d(a)\) is the smallest common denominator of the entries of \(a\).

is NP-complete.

**Theorem 3.7.0.10** The decision problem:
Instance: A $n \times n$ symmetric interval matrix $[M]$. 

Question: $\exists M \in [M]$ with index($M$) = 1 and coindex($[M]$) = $n - k$? is NP-hard.

Proof. Given integer $m$ and vector $a$, set $A = (I_m - aa^T)^{-1}$, $\mu = m - 1/d^2(a)$ and define the matrix

$$
[M] = \begin{bmatrix}
A & [z] \\
[z]^T & \mu
\end{bmatrix},
$$

$[z] = [-1, 1]^m$. (3.20)

Note that $I_m - aa^T$ is positive definite and thus $A$ exists and is also positive definite. From Theorem 3.4.3.1 we have that $\forall M \in [M]$,

$$In(M) = In(A) + In(\mu - z^T(I_m - aa^T)z)$$

(3.21)

Since $In(A) = (m, 0, 0)$, $[M]$ contains an index-1 matrix iff $\exists z^*$ such that $\mu - z^*T(I_m - aa^T)z^* < 0$ which would imply a "no" answer to problem 3.7.0.9. 

3.8 Results

The proposed tests have been implemented in the $\alpha$BB algorithm [1]. The use of the $\alpha$BB algorithm for solving problem (P) requires the calculation of the second derivatives of the constraints, which include first derivatives of the function $f$. Therefore, function $f$ must be three-times continuously differentiable in the specific
implementation we have developed. The tests presented here, however, are applicable to $C^2$ functions and can readily be integrated within algorithms that do not require the constraints to be in $C^2$, e.g. [3]. As mentioned previously, an efficient implementation of the tests can be constructed by using the interval values of the second-order derivatives of $f$ that can be computed when calculating $\alpha$ values for the underestimators. A more basic implementation has been used here, so that the computational performance provides a worst-case analysis of the cost of the tests.

We investigate the performance of the proposed tests on a number of problems. For each problem we perform one run using no test and separate runs using each test without local search. For the Gerschgorin, RecIn and Rohn tests we also perform runs with local search in order to evaluate whether there would be any improvement regarding the CPU time. For bounding the eigenvalues in Rohn’s test we used the interval extension of Gerschgorin’s theorem [4]. For each problem we give a table containing the CPU times for each run and the corresponding number of (non-degenerate) minima, TSs and other solutions found and a graph which shows the number of unfathomed nodes at each iteration for each run. We also give a summary of the success rates (no. of nodes fathomed by test / no. of times test applied) for each test in each problem (no local search applied). The computations were performed on an Intel CPU @ 3060 MHz using an absolute convergence tolerance of $10^{-6}$ and a minimum box size of $10^{-6}$.

3.8.1 Problem 1: Ackley’s function

For the first example, we apply the algorithm to Ackley’s function:

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

with $n = 3$ and $x \in [0.5, 3]^3$. This low-dimensional example has 81 first-order saddle points, which are found with all configurations of the algorithm (with or without tests). We can observe from Table 3.2 that, with the application of the regional tests, the CPU time can be reduced by more than 50% in comparison to the “no test” case (location of all critical points), which has a CPU time of 64 seconds. A
further reduction in CPU time of 15 to 30% is achieved with the application of the local search over areas that are found to have index-1. The RecIn test has the best performance, with a CPU time of only 19 seconds when the local search is also applied, with the Rohn test also exhibiting very strong performance. Furthermore, the Rohn and RecIn tests only return the TSs as solutions while the Gerschgorin test and the $2 \times 2$ test return a number of non-TS critical points too: 11 of the 84 higher-order saddle points or maxima in the case of the Gerschgorin test and all 27 minima in the case of the $2 \times 2$ test. In Figure 3.2, the number of open nodes in the branch-and-bound tree is reported as a function of iteration number for every test. The scales used in the five panels are the same to make comparison easier. The significant reduction in the number of iterations when the tests are applied is evident and the branch-and-bound tree is found to be much smaller.

Table 3.2: CPU times and number of solutions of each type found for each run for the Ackley function.

<table>
<thead>
<tr>
<th>Test</th>
<th>CPU Time</th>
<th>CPU Time</th>
<th>#Mins</th>
<th>#TSs</th>
<th>#Other solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With local search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No test</td>
<td>64 sec</td>
<td>-</td>
<td>27</td>
<td>81</td>
<td>84</td>
</tr>
<tr>
<td>Gersch.</td>
<td>38 sec</td>
<td>32 sec</td>
<td>0</td>
<td>81</td>
<td>11</td>
</tr>
<tr>
<td>$2 \times 2$ Inertia</td>
<td>33 sec</td>
<td>-</td>
<td>27</td>
<td>81</td>
<td>0</td>
</tr>
<tr>
<td>Rohn</td>
<td>30 sec</td>
<td>21 sec</td>
<td>0</td>
<td>81</td>
<td>0</td>
</tr>
<tr>
<td>RecIn</td>
<td>28 sec</td>
<td>19 sec</td>
<td>0</td>
<td>81</td>
<td>0</td>
</tr>
</tbody>
</table>
3.8.2 Problem 2: Levy function

In this example we use a Levy function: 
\[ f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2[1 + 10 \sin^2(\pi y_{i+1})] + (y_n - 1)^2, \]
where \( y_i = 1 + (x_i - 1)/4 \). In our case, \( n = 5 \) and \( x \in [-5,5]^5 \). This more challenging example has a total of 349 stationary points of which 142 are transition states and 63 are minima, as can be seen in Table 3.3. Notice that the Hessian of \( f \) is tridiagonal.

Again, without local search, we see a significant reduction in CPU time, of between 9 and 38% (Table 3.3), and in iteration number, of up to 41% (Figure 3.3). The maximum overall CPU time reduction achieved with the use of a test combined with local search is of 50%. The RecIn test has the best performance with a CPU time of 108 seconds in contrast to the 218 seconds required when no regional test is applied. As in the first example, the Rohn test
provides the second-best performance when accompanied by local search. However, without local search, the second-best performance is achieved with the $2 \times 2$ inertia test. Both Rohn and RecIn tests return only the TSs as solutions, whereas the $2 \times 2$ inertia test leads to the identification of all 63 minima and the Gerschgorin test to the identification of 58 other stationary points.

Table 3.3: CPU times and number of solutions of each type found for each run for the Levy function.

<table>
<thead>
<tr>
<th>Test</th>
<th>CPU Time</th>
<th>CPU Time</th>
<th>#Mins</th>
<th>#TSs</th>
<th>#Other solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With local search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No test</td>
<td>218 sec</td>
<td>-</td>
<td>63</td>
<td>142</td>
<td>144</td>
</tr>
<tr>
<td>Gersch.</td>
<td>197 sec</td>
<td>174 sec</td>
<td>0</td>
<td>142</td>
<td>58</td>
</tr>
<tr>
<td>2x2 Inertia</td>
<td>152 sec</td>
<td>-</td>
<td>63</td>
<td>142</td>
<td>0</td>
</tr>
<tr>
<td>Rohn</td>
<td>169 sec</td>
<td>140 sec</td>
<td>0</td>
<td>142</td>
<td>0</td>
</tr>
<tr>
<td>RecIn</td>
<td>134 sec</td>
<td>108 sec</td>
<td>0</td>
<td>142</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 3.3: Number of unfathomed nodes at each iteration for each run for the Levy function. Dashed curves correspond to the same test but with local search.

3.8.3 Problem 3: Himmelblau’s function

In this example we use an extension of Himmelblau’s function to multiple dimensions: 

\[ f(x) = \sum_{i<j} \left[ (x_i^2 + x_j - 11)^2 + (x_i + x_j^2 - 7)^2 \right], \]

where \( n = 6 \) and \( x \in [-5, 5]^6 \). The results are presented in Table 3.4 and Figure 3.4. Although this example has only one variable more than Problem 2, the number of stationary points is much greater, with 729 points in total, of which 192 are transition states and 64 are minima. There is therefore a considerable computational cost to searching for all stationary points. The basic algorithm, without any regional tests, identifies all 729 points in 520 CPU seconds, compared to 218 CPU seconds in Problem 2. In contrast, the use of tests without local search leads to a reduction in CPU time of between 36 and 52% and the use of tests with local search to a reduction of between 38 and 54%
overall. It is clear from these numbers that the application of one test provides most of the performance improvement in this example, and that the local search, albeit beneficial, has a modest impact on the overall CPU times. Once more the RecIn test is the most effective test, reducing the CPU time by a factor greater than 2 with respect to the case when no test is applied. In this particular case, the Gerschgorin test does not lead to the identification of additional stationary points. The $2 \times 2$ test offers second-best performance, and identifies all minima as well as all transition states.

Table 3.4: CPU times and number of solutions of each type found for each run for the Himmelblau function.

<table>
<thead>
<tr>
<th>Test</th>
<th>CPU Time</th>
<th>CPU Time With local search</th>
<th>#Mins</th>
<th>#TSs</th>
<th>#Other solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>No test</td>
<td>520 sec</td>
<td>-</td>
<td>64</td>
<td>192</td>
<td>473</td>
</tr>
<tr>
<td>Gersch.</td>
<td>332 sec</td>
<td>319 sec</td>
<td>0</td>
<td>192</td>
<td>0</td>
</tr>
<tr>
<td>2x2 Inertia</td>
<td>272 sec</td>
<td>-</td>
<td>64</td>
<td>192</td>
<td>0</td>
</tr>
<tr>
<td>Rohn</td>
<td>333 sec</td>
<td>320 sec</td>
<td>0</td>
<td>192</td>
<td>0</td>
</tr>
<tr>
<td>RecIn</td>
<td>248 sec</td>
<td>237 sec</td>
<td>0</td>
<td>192</td>
<td>0</td>
</tr>
</tbody>
</table>
3.8.4 Problem 4: 2D-XY lattice model

For the last example we use the 2D-XY lattice model \[ H = \frac{1}{2} \sum_{k \in \Lambda} \sum_{l \in N(k)} [1 - \cos(\theta_k - \theta_l)] \]

where \( \Lambda = \{1, 2, ..., 9\} \) and \( N\{k\} \) is the set of indices of the neighbouring lattice points to the lattice point with index \( k \).

The 2-dimensional XY lattice model has been studied, amongst others, in [54, 55]. The model exhibits exponential growth of the number of stationary points as the number of lattice points grows. Here, we consider a 3 x 3 lattice where \( \theta_7 = \theta_8 = \ldots \)
\[ \theta_9 = 0, \theta_3 = \theta_6 = \pi/2 \text{ and } \theta_i \in [-\pi, \pi] \text{ for } i = 1, 2, 4, 5. \] Thus, this is a 4-dimensional problem. This example has a relatively small number of stationary points (33), with only 5 transition states and one minimum, and the algorithm without tests identifies all these points within 86 CPU seconds. However, the performance of the tests, as presented in Table 3.5 and Figure 3.5, is more disparate than in previous examples. The frequent appearance of interval Hessian matrices where some or all diagonal elements include zero makes this example more challenging for some of the tests. Thus, the Gerschgorin test leads to a reduction in the total number of iterations of less than 4%, and no reduction in the CPU time, which remains at 86 CPU seconds. This is due to the fact that some Gerschgorin discs overlap when zero is present in the diagonal elements and this may result in the test being inconclusive. We note that the computational cost could be reduced with a more efficient implementation that permits the re-use of the calculations of the interval Hessian matrix elements carried out while constructing the \( \alpha \text{BB} \) underestimators for the purpose of the test. Nevertheless, based on the implementation used here, the Gerschgorin test does not lead to a change in CPU time and identifies 26 “other” solutions in addition to the 5 transition states. Secondly, in this case the Rohn test performs better than the \( \text{RecIn} \) test: this latter test leads to a larger CPU time than the Rohn test and fails to remove a number of non-TS solutions. The reason for this is the presence of zeros in the diagonal entries of the interval Hessian matrices that prevent application of the \( \text{RecIn} \) test. However, the use of the \( x\text{RecIn} \) test can overcome this problem and, as can be seen in Table 3.5, it performs slightly better than the Rohn test.
Table 3.5: CPU times and number of solutions of each type found for each run for the 2D-XY lattice model function.

<table>
<thead>
<tr>
<th>Test</th>
<th>CPU Time</th>
<th>CPU Time</th>
<th>#Mins</th>
<th>#TSs</th>
<th>#Other solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>With local search</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No test</td>
<td>86 sec</td>
<td>-</td>
<td>1</td>
<td>5</td>
<td>27</td>
</tr>
<tr>
<td>Gersch.</td>
<td>86 sec</td>
<td>86 sec</td>
<td>0</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>2x2 Inertia</td>
<td>46 sec</td>
<td>-</td>
<td>1</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>Rohn</td>
<td>33 sec</td>
<td>28 sec</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>RecIn</td>
<td>45 sec</td>
<td>40 sec</td>
<td>0</td>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>xRecIn</td>
<td>32 sec</td>
<td>27 sec</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 3.5: Number of unfathomed nodes at each iteration for each run for the 2D-XY lattice model function. Dashed curves correspond to the same test but with local search.
### 3.8.5 Overall performance of the tests

Table 3.6: Success rate, in percentage, for each test for each problem.

<table>
<thead>
<tr>
<th>Test/Problem</th>
<th>Ackley</th>
<th>Levy</th>
<th>Himmelblau</th>
<th>2D-XY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gersch.</td>
<td>1.93</td>
<td>0.37</td>
<td>1.40</td>
<td>0.02</td>
</tr>
<tr>
<td>2x2 Inertia</td>
<td>3.31</td>
<td>1.75</td>
<td>4.35</td>
<td>1.40</td>
</tr>
<tr>
<td>Rohn</td>
<td>4.06</td>
<td>0.73</td>
<td>1.23</td>
<td>2.79</td>
</tr>
<tr>
<td>RecIn</td>
<td>4.35</td>
<td>2.37</td>
<td>5.35</td>
<td>1.45</td>
</tr>
<tr>
<td>xRecIn</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.91</td>
</tr>
</tbody>
</table>

Overall, the application of the proposed tests leads to a reduction in the number of iterations and this is usually accompanied by a significant reduction in CPU time, by up to 50%. The application of the local search always leads to a reduction in both CPU time and iteration number. The most appropriate version of the recursive inertia test (RecIn or xRecIn) test, as indicated by the presence or not of zeros in the diagonal elements of the interval Hessian matrix, is found to provide the best performance in every case. The Rohn test usually performs well too, while the CPU time reduction is not as large with the Gerschgorin and $2 \times 2$ inertia tests. The worst performance was observed in applying the Gerschgorin test to Problem 4, where the presence of zeros in the Hessian matrix results in overlap of the Gerschgorin discs and the inability to eliminate most nodes. This provides a useful insight into the types of problems for which this test is most appropriate.

It is instructive to consider the success rates of the tests. In the proposed approach, the interval gradient test was applied at every node of the branch-and-bound tree and the chosen test was then applied at every node at which the interval gradient test was passed. The success rate of each test is calculated as the ratio of the number of
nodes fathomed by a test to the number of times this test was applied, and is reported in Table 3.6. The success rates obtained are of the order of a few percent, with a maximum value of 5.35%. As discussed, the lowest overall success rate is exhibited by the Gerschgorin test, while the RecIn test is most consistently successful. As can be expected, the tests tend to become more effective as the nodes become smaller for two reasons. First, in the test cases considered here, there are many stationary points and a large portion of the domain contains points at which the Hessian matrix is index-1 (whether they are index-1 critical points or not). Second, the larger the volume of the node the larger the overestimation inherent in the evaluation of the interval Hessian matrix, so that large nodes cannot be eliminated easily. Despite the relative inefficiency of the tests, the CPU-times for the problems presented are halved, indicating that the tests play a useful role. Further gains in CPU time may be derived by imposing a maximum threshold on the size of the node so that tests are only applied to “small-enough” nodes.

A strategy to reduce the number of iterations is to apply multiple tests. The RecIn/xRecIn tests generally lead to the elimination of regions that are eliminated by other tests. However, the reverse is not true. If the tests are applied in series, it is therefore advantageous to apply the least computationally demanding tests first, specifically Gerschgorin and $2 \times 2$ inertia and to follow this with RecIn/xRecIn tests. This strategy was deployed on the test problems, but due to the relatively low dimensionality of the examples (up to 6 variables), it did not lead to an improvement in CPU time compared to applying RecIn/xRecIn only. It would be interesting to explore this strategy further by deploying the tests in parallel on larger problems.

3.9 Summary

In this Chapter we deal with the problem of locating all index-1 saddle points of general $C^2$ functions. We have presented five methods, Gerschgorin, Rohn, $2 \times 2$ inertia, recursive inertia (RecIn) and extended recursive inertia (xRecIn), for the
identification of non-index-1 symmetric interval matrices. As shown, this is an NP-hard problem. We have used the αBB algorithm for locating all critical points of a given function and utilised the presented methods as tests in order to identify and remove areas that do not contain any index-1 solutions. From the presented test problems we see that this lead to approximately a 50% reduction in computational time. Finally, among the developed tests, the xRecIn method had the best performance.
4 An interval-matrix branch-and-bound algorithm for bounding eigenvalues

4.1 Introduction

In this chapter we develop an interval-matrix branch and bound method (MBB) for calculating bounds on any individual eigenvalue of symmetric interval matrices. Furthermore we prove two theorems with respect to the necessary conditions for a minimum or maximum eigenvalue to reside in the interior (a term we define in the coming section) of the interval matrix. Based on this, we develop a local search algorithm which we use alongside with another local search algorithm from the literature for upper bounding (or lower bounding if we are calculating a maximum). We test the MBB algorithm on a number of randomly generated matrices and present detailed results on its performance.

The chapter is organized as follows: In Section 4.2 we give a brief introduction to interval matrices and a few definitions. In Section 4.3 we present the pseudocode of the Interval-Matrix Branch-and-Bound algorithm. In Section 4.4 we present the general bounding approach used in the main algorithm. In Section 4.5 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 4.6 we present results from the application of the method and finally and in Section 4.7, we close with a brief summary.

4.2 Preliminaries

We denote interval variables with lower case letters inside square brackets, $[x]$, and the corresponding lower and upper bounds of $[x]$ as $\underline{x}$ and $\overline{x}$ respectively. Symmetric interval matrices are denoted by capital letters inside square brackets.

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

**Definition 4.2.0.1 (Eigenvalues of a symmetric interval matrix)** The $i$-th largest eigenvalue of a symmetric matrix $[M]$ is defined as $\lambda_i([M]) = \{ \lambda_i(M) : M \in [M] \}$.

Note that the set $\lambda_i([M]) = \{ \lambda_i(M) : M \in [M] \}$ is a compact set in $\mathbb{R}$ (32). Thus we can write $\lambda_k([M]) = [\underline{\lambda}_k, \overline{\lambda}_k]$. To avoid cumbersome notation we omit the square brackets, which we use to denote a single interval, around $\lambda_k([M])$.

**Definition 4.2.0.2 (Spectral radius of a symmetric interval matrix)** The spectral radius, $\rho([M])$, of an interval matrix $[M]$ is defined as $\rho([M]) = \max_{M \in [M]} \rho(M)$, where $\rho(M)$ is the spectral radius of $M$.

**Definition 4.2.0.3 (Norm of a symmetric interval matrix)** For any norm $\| \cdot \|$ defined for scalar matrices, we define the corresponding norm for interval matrices as $\| [M] \| = \max \{ \| M \| : M \in [M] \}$.

**Definition 4.2.0.4 (Interior and border of an interval matrix)** The interior of an interval matrix $[M]$ is defined as $\mathcal{I}([M]) = \{ M \in [M] : m_{ij} = m_{ij} \text{ or } m_{ij} =$
\( m_{ij} \text{ iff } m_{ij} = \overline{m_{ij}} \). The border, \( B([M]) \), of \([M]\) is simply the complement of \( I([M]) \) in \([M]\).

### 4.3 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 (34). Given an \( n \times n \) symmetric interval matrix \([M]\), the algorithm returns lower and upper bounds for \( \lambda_k([M]) \). The inputs of the algorithm are the symmetric matrix \([M]\), the order of the eigenvalue (e.g. \( k \)-th largest eigenvalue), \( k \), for which the bounds will be calculated, the maximum number of iterations, maxiters and the precision \( \epsilon \). We denote by \( L \) the list which contains sublists of the form \( \{[M_i], l_i, u_i\} \) where \([M_i]\) is a symmetric matrix with \( l_i \) and \( u_i \) lower and upper bounds of \( \lambda_k([M_i]) \). 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 \( \epsilon \)).
Algorithm 7 Interval-MatrixBB

1: Inputs: \([M], k, \text{maxiters}, \epsilon\).

2: Calculate lower and upper bounds \(l\) and \(u\) for \(\lambda_k([M])\).

3: Set \(BLB = l, BUB = u\) and \(RLB = +\infty\).

4: Initialize \(L = \{ [[M], l, u] \}\), \(iter = 0\).

5: while \(iter \leq \text{maxiters}\) do

6: Choose the first entry, \(L_1\), from list \(L\).

7: Set \([M] = L_1[1], l = L_1[2], \text{and } u = L_1[3]\).

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 \([M_1]\) and \([M_2]\).

12: Obtain bounds \(l_1\) and \(u_1\) for \(\lambda_k([M_1])\).

13: Obtain bounds \(l_2\) and \(u_2\) for \(\lambda_k([M_2])\).

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 \([M_1], l_1, u_1\) in \(L\) so that the lower bounds in \(L\) are in increasing order.

19: end if

20: end if

21: end while
if $BUB - l_2 < \epsilon$ then

$RLB = \min \{l_2, RLB\}$

else

if $l_2 < BUB$: Insert $\{[M_2], l_2, u_2\}$ in $L$ so that the lower bounds in $L$ are in increasing order.

end if

$BLB = \min \{l_1, l_2, RLB, L_1[2]\}$ ($L_1[2]$ being the second entry of the first sublist in $L$.

$iter++.$

if $L$ is empty or $BUB - BLB < \epsilon$ then

Return $BLB$ and $BUB$.

end if

end if

end while

Return $BLB$ and $BUB$.

The choice of branching strategy (in our case which entry we choose to branch on in step 10 of Algorithm 7) can have a strong influence on the performance of Branch-and-Bound algorithms (see for example [10]). As will be evident from Proposition 4.4.0.6 in Section 4.4, in order to achieve theoretical convergence for a given precision, $\epsilon$, 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 6, the node which is visited at each step is the one with the current lowest lower bound.

Bounds on \(\lambda_k([M]) = \max_{M \in [M]} \lambda_k(M)\) can be calculated in an analogous way to Algorithm 6. 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.

### 4.4 General bounding approach

We begin by recalling Rohn’s theorem from Chapter 3, Subsection 3.4.9. We can write any given interval matrix \([M]\) in the form \([M] = C + [E]\) where by \(C\) we denote the centre matrix of \([M]\), \(c_{ij} = (\overline{m_{ij}} + \underline{m_{ij}})/2\) and by \([E]\) 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 6.

**Theorem 4.4.0.1** (Rohn[32, 77]) *Given a symmetric interval matrix \([M] = C + [E],\) then*

\[
\lambda_k(C) + \lambda_n([E]) \leq \lambda_k(C + [E]) \leq \lambda_k(C) + \lambda_1([E]), \text{ for } k = 1, 2, \ldots, n. \tag{4.1}
\]

We can write eq. (4.1) as

\[
\lambda_k(C) - \rho([E]) \leq \lambda_k(C + [E]) \leq \lambda_k(C) + \rho([E]). \tag{4.2}
\]

Thus, we can use the following for calculating bounds on \(\lambda_k([M])\):

\[
l = \lambda_k(C) + b \leq \lambda_k([M]) \leq \lambda_k(C) = u. \tag{4.3}
\]
where $b$ is a lower bound of $-\rho([E])$. Since the value $\lambda_k(C)$ is attained for $C \in [M]$ this means that $\lambda_k([M])$ cannot be greater than $\lambda_k(C)$, and thus $\lambda_k(C)$ 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 $-\rho([E])$ using a number of methods (see for example [2]) or even the exact value of $\rho([E])$ with the Hertz method [26] ($O(2^{n-1})$). We will use

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

(4.4)

(This bound is the same as the one we would get by the interval Gershgorin method [2].) 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 $\lambda_k(C_1)$ and/or $\lambda_k(C_2)$ could be less than $\lambda_k(C)$ and at the same time $b_1 = -\|E_1\|_\infty$ and/or $b_2 = -\|E_2\|_\infty$ might have not improved adequately or even at all. Nevertheless, after a certain number of iterations the lower bound must improve (see proposition 4.3.0.6).

Next, the following Lemma (the same with Lemma 3.4.5.1 in Chapter 3) tells us that there is no need to branch on the diagonal entries.

**Lemma 4.4.0.5** ([32]) *Given an $n \times n$ symmetric interval matrix $[M]$, define the symmetric interval matrices*

$$[L] = \{l_{ii} = m_{ii} \text{ and } [l_{ij}] = [m_{ij}] \text{ for } i \neq j\}$$

(4.5)

*and*

$$[U] = \{u_{ii} = m_{ii} \text{ and } [u_{ij}] = [m_{ij}] \text{ for } i \neq j\}.$$
Then $\forall M \in [M], \exists L \in [L]$ and $U \in [U]$ such that

$$\lambda_i(L) \leq \lambda_i(M) \leq \lambda_i(U) \text{ for } i = 1, 2, \ldots, n.$$  \hspace{1cm} (4.7)

Lemma 4.4.0.5 tells us that we need to consider only the lower (upper) parts of the diagonal elements of an $n \times n$ symmetric matrix $[M]$ in order to calculate bounds for $\lambda_k([M])$ (\overline{\lambda_k([M])}) for any $k \in \{1, 2, \ldots, n\}$. Thus there is no need for branching on the diagonal elements.

**Proposition 4.4.0.6** Consider an $n \times n$ symmetric interval matrix $[M]$ with $m_{ij} - m_{ij} = w > 0$ for $i \neq j$ and $\overline{m_{ii}} - m_{ii} = 0$ for $i = 1, 2, \ldots, n$. Then, given accuracy $\epsilon$, by successive bisection every submatrix will have $u_i - l_i \leq \epsilon$ after a total of

$$\left[ \left( \frac{(n-1)w}{2\epsilon} \right)^{\frac{n^2-n}{2}} - 1 \right] \text{ iterations (bisectons).}$$

**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 $[M_i]$ will have $u_i - l_i \leq \epsilon$.

Initially, for the matrix $[M] = C + [E]$, based on ineq. (4.3), we have $u - l = ||[E]||_\infty = (n-1)w/2$. We want $||[E]||_\infty$ to be halved $k$ times so that we have

$$\frac{(n-1)w/2}{2^k} \leq \epsilon \Rightarrow k \geq \log_2 \left( \frac{(n-1)w}{2\epsilon} \right).$$ \hspace{1cm} (4.8)

In order to halve $||[E]||_\infty$ once, we need to branch on each of the $\frac{n^2-n}{2}$ off-diagonal elements. Which means that the depth of the tree will grow to $d = \frac{n^2-n}{2}$. Therefore, in order to achieve the required accuracy we will need to end up with a tree of depth

78
\[ d \geq \frac{n^2-n}{2} \log_2 \left( \frac{(n-1)w}{2\epsilon} \right). \] 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

\[
\left\lceil \left( \frac{(n - 1)w^2}{2\epsilon} \right)^{\frac{n^2-n}{2}} - 1 \right\rceil. \tag{4.9}
\]

At this point we make the following notes: first, although the number of iterations given by (4.9) is forbiddingly high, this is a worst case scenario where no node fathoming takes place. As it can be seen in the Results Section, the actual performance appears better than (4.9). 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 \(\frac{n^2-n}{2}\). Second, to the best of our knowledge, no other method exists that can solve the problem of calculating \(\lambda_k\) and/or \(\lambda_k^\pm\) for \(k \in \{2, ..., n-1\}\) for general interval symmetric matrices, for a given accuracy \(\epsilon\). Thus, we do not know of any better upper bound on the algorithmic complexity of the problem.

### 4.5 Local search algorithms

As mentioned in the previous section, we can improve the trivial upper bound, \(\lambda_k(M_c)\), of \(\lambda_k[M]\). For this purpose we present two local search methods. The first is by Hladik et al. [30], and the second introduced here, based on Theorem 4.5.2.1 given in Subsection 4.5.2.
4.5.1 Hladík et al. ([30]) local search

In [30], 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 (see [30]). Here, we will make use of the method, from [30], shown in Algorithm 8 where with $C$ we denote the centre matrix of $[M]$ and with $E$ the radius matrix (e.g. $e_{ij} = (\overline{m}_{ij} - \underline{m}_{ij})/2$). By $v_k(M)$ we denote the eigenvector of $M$ which corresponds to the $k$-th largest eigenvalue.

**Algorithm 8** Hladík et al.’s. local improvement algorithm

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

For a local search of the maximum of $\lambda_k$ (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.

4.5.2 A new local search method.

It is known that the values $\overline{\lambda}_n$ and $\underline{\lambda}_1$ of a symmetric interval matrix $[M]$ are attained at extreme matrices of $[M]$ ([26]). However, for the rest of the eigenvalue bounds ($\overline{\lambda}_n$, $\underline{\lambda}_1$, and $\overline{\lambda}_k$, $\underline{\lambda}_k$ for $k = 2, ..., n-1$) of a general symmetric interval matrix the situation...
is more complicated since boundary values can be attained in $\mathcal{I}([M])$ and/or $\mathcal{B}([M])$. Nevertheless, for matrices with non-zero width off-diagonal elements we can prove the following necessary conditions for $M^* \in \{ M \in [M] : \text{argmin} \, \lambda_k(M) \}$ (or $M^* \in \{ M \in [M] : \text{argmax} \, \lambda_k(M) \}$ for $k \in \{2, \ldots, n-1\}$) to belong in $\mathcal{I}([M])$. 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 4.4.0.5. 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 normalised with respect to the Euclidean norm.

**Theorem 4.5.2.1** Consider an $n \times n$ symmetric interval matrix $[M]$ with $\underline{m}_{ii} = \overline{m}_{ii}$ for $i = 1, 2, \ldots, n$ and $\underline{m}_{ij} \neq \overline{m}_{ij}$ for $i \neq j$. Let $M^* \in \{ M \in [M] : \text{argmin} \, \lambda_k(M) \}$ for some integer $k < n$. If $M^* \in \mathcal{I}([M])$ then $\lambda_k(M^*) = \lambda_{k+1}(M^*)$ or $\|v_k\|_\infty = 1$.

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

$$M_1 = M^* + \rho[mI - v_kv_k^T]$$

(4.10)

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

$$M_2 = M_1 + \rho D = M^* + \rho[\text{diag}(v_kv_k^T) - v_kv_k^T],$$

(4.11)

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

Note that by “or” in Theorem 4.5.2.1 we mean that at least one of the conditions must be true. The analogous argument of Theorem 4.5.2.1 can be made for $M^* \in \{M \in [M] : \text{argmax } \lambda_k(M)\}$ for $k > 1$. Furthermore, we can employ the formula used in the proof of Theorem 4.5.2.1,

$$M^* + \rho[\text{diag}(v_k v_k^T) - v_k v_k^T]$$

(4.12)

in an attempt to improve the upper bound, in the same way as with Algorithm 3. This is detailed in Algorithm 4.
Algorithm 9 A new local improvement algorithm.

1: Inputs: $[M], k \ (k < n)$

2: Set $M = C$

3: while $\lambda_k(M) > \lambda_{k+1}(M)$ and $\|v_k(M)\|_{\infty} < 1$ do

4: $d = \lambda_k(M) - \lambda_{k+1}(M)$

5: $\rho_{\text{max}} = \max\{\rho \in [0, d] : M + \rho[\text{diag}(v_kv_k^T) - v_kv_k^T] \in [M]\}$

6: if $\rho_{\text{max}} > 0$ then

7: $M = M + \rho_{\text{max}}[\text{diag}(v_kv_k^T) - v_kv_k^T]$

8: else

9: Return $\lambda_k(M)$

10: end if

11: end while

12: Return $\lambda_k(M)$.

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

Notice that Algorithm 8 can be applied to any kind of symmetric matrix while Algorithm 9 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 8 always searches extreme matrices which makes it more suitable for obtaining an upper bound on $\lambda_n$ and a lower bound on $\bar{\lambda}_1$. A comparison between the bounds obtained by the two algorithms is given in example 1 in the Results Section.

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 $\lambda_k$) bound we then apply Algorithms 8 and
9 for further improvement.

We can make a stronger statement than the one of Theorem 4.5.2.1 by observing
the following: If $\lambda_k = \lambda_{k+1}, \lambda_{k+1} \neq \lambda_{k+2},$ and $m = \|v_k\|_2^2 + \|v_{k+1}\|_\infty^2 < 1$ then, for
adequately small $\rho,$ the matrix $M_1 = M^* + \rho[mI - v_kv_k^T - v_{k+1}v_{k+1}^T]$ would have “con-
veniently placed” eigenvalues (but $M_1$ not necessarily in $[M]$) while the correspond-
ing matrix $M_2 = \rho\sum_{i=0}^s [\text{diag}(v_{k+i}v_{k+i}^T)-v_{k+i}v_{k+i}^T]$ would have $\lambda_k(M_2) < \lambda_k(M^*)$ with $M_2 \in [M].$ More formally we have:

**Theorem 4.5.2.2** Let $M^* \in \{M \in [M] : \text{argmin} \lambda_k(M)\}.$ If $M^* \in \mathcal{I}([M])$ then $\exists s \in \{0, 1, ..., n - k\}$ such that $\lambda_k(M^*) = \lambda_{k+1}(M^*) = ... = \lambda_{k+s}(M^*)$ and
\[\sum_{i=0}^s \|v_{k+i}\|_\infty^2 \geq 1.\]

**Proof.** Assume $M^* \in \mathcal{I}([M])$ and that up to some integer $s \in \{0, 1, ..., n - k\}$ $\lambda_k = ...
= \lambda_{k+s}$ (with $\lambda_{k+s} \neq \lambda_{k+s+1}$ otherwise $s = n - k$) and $m = \sum_{i=0}^s \|v_{k+i}\|_\infty^2 < 1.$
Consider the matrix
\[M_1 = M^* + \rho \left[ mI - \sum_{i=0}^s v_{k+i}v_{k+i}^T \right]. \tag{4.13}\]
The eigenpairs of $M_1$ are $(v_i, \lambda_i + \rho m)$ for $i < k$ and $(v_k, \lambda_{k+1} - \rho(1 - m))$ for
$i = 0, 1, ..., s.$ Following the same reasoning as in the proof of Theorem 4.5.2.1 for

84
adequately small $\rho$ the matrix

$$M_2 = \rho \sum_{i=0}^{s} \left[ \text{diag}(v_{k+i} v_{k+i}^T) - v_{k+i} v_{k+i}^T \right]$$  \hspace{1cm} (4.14)$$

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

\[ \square \]

Note that we could make use of Theorem 4.5.2.2 in order to expand the local search method we give in Algorithm 4 however this would make the algorithm more complicated while at the same time would not provide any significant effect in performance since the method usually terminates because we have reached a bound from one of the interval elements of the interval matrix rather than because $\lambda_k = \lambda_{k+1}$. Nevertheless, we have stated Theorem 4.5.2.2 for completeness.

### 4.6 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 $[M] = C + [E]$ by generating the central matrix $C$ with each entry chosen uniformly from $[-20, 20]$ and the $[E]$ 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 $\lambda_n$ and $\lambda_1$ can be computed much faster, as mentioned previously, by the method from [20]. 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 $\Delta_1$ in Table 1 appear to have width $> 10^{-1}$ yet are marked as converged).

### 4.6.1 Example 1 – Comparison of the two local improvement algorithms

In this example we make a comparison between the two local improvement methods, Algorithms 8 and 9. We generate four groups of random $5 \times 5$ 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 $\lambda_k$ for $k = 1, 2, 3, 4$ and lower bounds on $\overline{\lambda}_k$ for $k = 2, 3, 4, 5$. In Figure 4.1, 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 9 were better than those obtained by Algorithm 8, while negative values indicate the reverse. More explicitly, we denote by $\lambda_k^{(2)U}$ and $\lambda_k^{(2)L}$ the upper and lower bounds on $\lambda_k$ and $\overline{\lambda}_k$ respectively, obtained by Algorithm 8 and by $\lambda_k^{(3)U}$ and $\lambda_k^{(3)L}$ the corresponding bounds obtained by Algorithm 9. The values on the $x$-axis of each histogram represent the quantities $\lambda_k^{(2)U} - \lambda_k^{(3)U}$, $k = 1, 2, 3, 4$ and $\lambda_k^{(3)L} - \lambda_k^{(2)L}$, $k = 2, 3, 4, 5$. 

86
Figure 4.1: Comparison of bounds obtained by Algorithms $\mathfrak{A}$ and $\mathfrak{B}$. The histograms show the distributions of the values of $\lambda_{k}^{(2)} U - \lambda_{k}^{(3)} U$, $k = 1, 2, 3, 4$ and $\lambda_{k}^{(3)} L - \lambda_{k}^{(2)} L$, $k = 2, 3, 4, 5$ 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.

As we see from Figure 4.1, for smaller radii, Algorithm $\mathfrak{A}$ performs significantly better than Algorithm $\mathfrak{B}$. However, as the radius increases, the relative performance of Algorithm $\mathfrak{B}$ improves, as indicated by the increasing median value. Although on average Algorithm $\mathfrak{B}$ 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.

4.6.2 Examples 2-6 – Performance of the eigenvalue bounding algorithm

The next set of examples is used to investigate the performance of Algorithm 7. 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 $\lambda_k^{(i)L}$ and $\lambda_k^{(i)U}$ the lower and upper bounds, respectively, on $\lambda_k$ 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 $\lambda_k$, $\lambda_k^L$, and $\lambda_k^U$; 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 $I_{10}$ and $\overline{T}_{10}$, of the bounds on $\lambda_k$ and $\overline{\lambda_k}$, respectively, between the initial and the tenth iterations:

$$ I_{10} = 100 \left[ 1 - \frac{(\lambda_k^{(10)U} - \lambda_k^{(10)L})}{(\lambda_k^{(1)U} - \lambda_k^{(1)L})} \right] $$

and

$$ \overline{T}_{10} = 100 \left[ 1 - \frac{(\overline{\lambda_k^{(10)U}} - \overline{\lambda_k^{(10)L}})}{(\overline{\lambda_k^{(1)U}} - \overline{\lambda_k^{(1)L}})} \right] $$

where $I_{10}$ is given by (4.15) and $\overline{T}_{10}$ is given by (4.16).

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

$$ S = 100(\lambda_k^L - \lambda_k^U)/(\lambda_k^U - \lambda_k^L); $$

(4.17)
the percentage improvement of the outer bounds between the initial bounds and final outer bounds, $OBI$:

$$OBI = 100\left(\frac{\lambda_k^U - \lambda_k^L}{\lambda_k^{(1)}_U - \lambda_k^{(1)}_L}\right).$$ (4.18)

4.6.3 Example 2: $n = 3, R = 10$

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


Results are presented in Table 4.1 while in Figure 4.2 we plot the inner, outer and initial outer eigenvalue ranges. Note that all the intervals in Figure 4.2 overlap and thus other methods would return the more conservative range $[\lambda_3, \lambda_1] = [-35.387, 11.363]$, 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 4.1: Summary of results for Example 2. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $T_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\overline{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16), $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17) and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>$\lambda_k$ bounds / time (CPU s)</th>
<th>$\overline{\lambda}_k$ bounds / time (CPU s)</th>
<th>$I_{10}$, $T_{10}$</th>
<th>$S$, $OBI$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\lambda_1]$</td>
<td>[2.462, 2.563]$^*$ / 0.55</td>
<td>[30.560, 30.654]$^*$ / 0.01</td>
<td>55%, 74%</td>
<td>99%, 28%</td>
</tr>
<tr>
<td>$[\lambda_2]$</td>
<td>[-13.534,-13.411] / 5.32</td>
<td>[11.267,11.363]$^*$ / 0.05</td>
<td>61%, 68%</td>
<td>99%, 35%</td>
</tr>
<tr>
<td>$[\lambda_3]$</td>
<td>[−35.387, –35.304]$^*$ / 0.00</td>
<td>[-9.041,-8.900] / 7.99</td>
<td>96%, 63%</td>
<td>99%, 30%</td>
</tr>
</tbody>
</table>
Figure 4.2: Eigenvalue ranges of Example 2. Initial bounds are shown with 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.

4.6.4 Example 3: \( n = 4, \ R = 5 \)

In this example we increase the dimension of the random test matrix to \( 4 \times 4 \) and reduce the radius to \( R = 5 \) and obtain the following matrix:
\[
[M] = 
\begin{bmatrix}
\end{bmatrix}
\]

Results are shown in Table 4.2 while in Figure 4.3 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 4.2: Result table for Example 3. An asterisk indicates the bound widths have converged to \(10^{-1}\). \(I_{10}\) and \(I_{OBI}\) are the percentages of improvement, for the bounds of \(\lambda_k\) and \(\ovl\lambda_k\) respectively, after 10 steps (eq. 4.15 and 4.16), \(S\) is the percentage of relative sharpness of the final bounds (eq. 4.17) and \(OBI\) is the percentage of outer bounds improvement (eq. 4.18).

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

92
Figure 4.3: Eigenvalue ranges of Example 3. Initial bounds are shown with 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.

4.6.5 Example 4: $n = 5, R = 5$

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

Table 4.3: Result table for Example 4. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $\overline{I}_{10}$ are the percentages of improvement, for the bounds of $\lambda_k$ and $\overline{\lambda}_k$ respectively, after 10 steps (eq. 4.13 and 4.14), $S$ is the percentage of relative sharpness of the final bounds (eq. 4.16) and $OBI$ is the percentage of outer bounds improvement (eq. 4.17).

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>$\underline{\lambda}_k$ bounds / time (CPU s)</th>
<th>$\overline{\lambda}_k$ bounds / time (CPU s)</th>
<th>$I_{10}$, $\overline{I}_{10}$</th>
<th>$S$, $OBI$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\lambda_1]$</td>
<td>$[16.076,17.873]$ / 7.65</td>
<td>$[40.296,40.340]^*$ / 0.05</td>
<td>26% , 62%</td>
<td>92% , 28%</td>
</tr>
<tr>
<td>$[\lambda_2]$</td>
<td>$[4.131,5.041]$ / 7.99</td>
<td>$[25.990,26.296]$ / 5.53</td>
<td>36% , 18%</td>
<td>95% , 30%</td>
</tr>
<tr>
<td>$[\lambda_4]$</td>
<td>$[-31.857,-31.715]$ / 2.56</td>
<td>$[-12.550,-11.196]$ / 7.86</td>
<td>34% , 28%</td>
<td>93% , 35%</td>
</tr>
<tr>
<td>$[\lambda_5]$</td>
<td>$[-53.925,-53.834]^*$ / 0.01</td>
<td>$[-24.441,-24.404]^*$ / 0.02</td>
<td>60% , 53%</td>
<td>99% , 11%</td>
</tr>
</tbody>
</table>
Figure 4.4: Eigenvalue ranges of Example 4. Initial bounds are shown with 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.

Results and eigenvalue ranges for this example are shown in Table 4.3 and Figure 4.4 respectively. Figures 4.5-4.7 are indicative of the algorithm’s behaviour. In the first case (Fig. 4.5) nodes are being removed at a high rate and thus convergence is achieved very fast. In the second case (Fig. 4.6) 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 (Fig. 4.7) 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 4.8 we show the progress of the bounds on $\lambda_4$ when we do not make use of local search algorithms. A comparison of this figure with Figure 4.6 demonstrates that the use of the local search can significantly increase the convergence speed.

Figure 4.5: Bounding $\lambda_5$: 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 4.6: Bounding $\lambda_1$: 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 4.7: 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 4.8: Bounding $\lambda_4$ 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.

4.6.6 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 $7 \times 7$ symmetric matrix with a small number of interval entries (12 off-diagonal and 2 diagonal entries randomly chosen with $R = 5$) is considered. The matrix is given in Appendix A. Results are given in Table 4.4 and eigenvalue bounds are plotted in Figure 4.9. 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.

Table 4.4: Result table for Example 5. An asterisk indicates the bound widths have converged to $10^{-1}$. $I_{10}$ and $\overline{I}_{10}$ are the percentages of improvement, for the bounds of $\bar{\lambda}_k$ and $\overline{\lambda}_k$ respectively, after 10 steps (eq. 4.15 and 4.16), $S$ is the percentage of relative sharpness of the final bounds (eq. 4.17), and $OBI$ is the percentage of outer bounds improvement (eq. 4.18).

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>$\lambda_k$ bounds / time (CPU s)</th>
<th>$\overline{\lambda}_k$ bounds / time (CPU s)</th>
<th>$I_{10}$, $\overline{I}_{10}$</th>
<th>$S$, $OBI$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[\lambda_1]$</td>
<td>[36.059, 36.297] / 9.41</td>
<td>[46.023, 46.081]* / 0.03</td>
<td>56%, 74%</td>
<td>97%, 53%</td>
</tr>
<tr>
<td>$[\lambda_2]$</td>
<td>[29.451, 29.552]* / 0.05</td>
<td>[40.183, 40.272]* / 6.36</td>
<td>64%, 62%</td>
<td>98%, 50%</td>
</tr>
<tr>
<td>$[\lambda_3]$</td>
<td>[4.713, 4.814]* / 0.07</td>
<td>[14.494, 14.497]* / 0.04</td>
<td>68%, 69%</td>
<td>99%, 53%</td>
</tr>
<tr>
<td>$[\lambda_4]$</td>
<td>[-15.814, -15.713]* / 1.26</td>
<td>[-7.855, -7.850]* / 0.08</td>
<td>54%, 54%</td>
<td>93%, 64%</td>
</tr>
<tr>
<td>$[\lambda_5]$</td>
<td>[-20.491, -20.389]* / 0.76</td>
<td>[-12.244, -12.063] / 6.18</td>
<td>52%, 51%</td>
<td>97%, 64%</td>
</tr>
<tr>
<td>$[\lambda_7]$</td>
<td>[-51.939, -51.838]* / 0.06</td>
<td>[-39.955, -39.859]* / 0.70</td>
<td>66%, 60%</td>
<td>98%, 47%</td>
</tr>
</tbody>
</table>
Figure 4.9: Eigenvalue ranges of Example 5. Initial bounds are shown with 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.

4.6.7 Example 6: Tridiagonal matrix, $n = 10$, $R = 5$

In this example we apply Algorithm 7 to a $10 \times 10$ randomly generated tridiagonal symmetric matrix, again with $R = 5$. The matrix is given in Appendix B. Results are given in Table 4.5 and eigenvalue bounds are plotted in Figure 4.10. 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 4.5: Result table for Example 6. An asterisk indicates the bound widths have converged to \(10^{-1}\). \(I_{10}\) and \(T_{10}\) are the percentages of improvement, for the bounds of \(\underline{\lambda}_k\) and \(\overline{\lambda}_k\) respectively, after 10 steps (eq. 4.15 and 4.16), \(S\) is the percentage of relative sharpness of the final bounds (eq. 4.17) and \textit{OBI} is the percentage of outer bounds improvement (eq. 4.18).

<table>
<thead>
<tr>
<th>Eigenvalue</th>
<th>(\underline{\lambda}_k) bounds / time (CPU s)</th>
<th>(\overline{\lambda}_k) bounds / time (CPU s)</th>
<th>(I_{10}), (T_{10})</th>
<th>(S), \textit{OBI}</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\lambda_1])</td>
<td>([19.584, 19.883]) / 9.01</td>
<td>([34.603, 34.841]) / 7.72</td>
<td>50% , 52%</td>
<td>96% , 15%</td>
</tr>
<tr>
<td>([\lambda_2])</td>
<td>([9.789, 10.451]) / 9.79</td>
<td>([22.553, 22.755]) / 7.00</td>
<td>41% , 44%</td>
<td>93% , 37%</td>
</tr>
<tr>
<td>([\lambda_3])</td>
<td>([7.707, 8.682]) / 9.26</td>
<td>([18.238, 18.272]) / 0.15</td>
<td>34% , 68%</td>
<td>90% , 38%</td>
</tr>
<tr>
<td>([\lambda_4])</td>
<td>([3.900, 4.001])^* / 0.25</td>
<td>([15.399, 15.947]) / 9.82</td>
<td>69% , 47%</td>
<td>95% , 34%</td>
</tr>
<tr>
<td>([\lambda_5])</td>
<td>([2.662, 2.763])^* / 0.39</td>
<td>([14.837, 15.426]) / 10.12</td>
<td>70% , 50%</td>
<td>95% , 30%</td>
</tr>
<tr>
<td>([\lambda_6])</td>
<td>([-5.772, -5.452]) / 8.55</td>
<td>([4.877, 5.313]) / 10.46</td>
<td>61% , 58%</td>
<td>93% , 33%</td>
</tr>
<tr>
<td>([\lambda_7])</td>
<td>([-12.201, -11.282]) / 9.47</td>
<td>([-3.034, -2.248]) / 10.61</td>
<td>40% , 45%</td>
<td>83% , 42%</td>
</tr>
<tr>
<td>([\lambda_8])</td>
<td>([-19.074, -18.253]) / 10.81</td>
<td>([-9.121, -8.691]) / 8.89</td>
<td>36% , 41%</td>
<td>88% , 45%</td>
</tr>
<tr>
<td>([\lambda_9])</td>
<td>([-29.158, -28.833]) / 8.76</td>
<td>([-12.945, -12.223]) / 8.98</td>
<td>25% , 31%</td>
<td>94% , 16%</td>
</tr>
<tr>
<td>([\lambda_{10}])</td>
<td>([-36.027, -35.600]) / 11.24</td>
<td>([-25.408, -24.931]) / 11.45</td>
<td>55% , 51%</td>
<td>92% , 36%</td>
</tr>
</tbody>
</table>
Figure 4.10: Eigenvalue ranges of Example 6. Initial bounds are shown with 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.

4.6.8 Examples 7-9: Identification of non index-1 areas.

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 4.1-4.5, 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 as we have seen in Chapter 3. In Table 4.6 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 \(\bar{\lambda}_{n-1}\) and determine whether the region may contain an index-1 saddle point: Rohn’s method (i.e., the initial step of Algorithm \(\text{A}\)) or Algorithm \(\text{A}\) for a maximum of only five steps. For each approach, we report the number of hyper-rectangle found such that \(\bar{\lambda}_{n-1} \leq 0\).

As test functions we use Ackley’s function (Example 7, (4.19)), Levy’s function (Example 8, (4.20)) and Himmelblau’s function (Example 9, (4.21)):

\[
\begin{align*}
\text{f} (x) &= -20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i) \right) + 20 + e, \quad (4.19) \\
\text{f} (x) &= \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left[ 1 + 10 \sin^2(\pi y_{i+1}) \right] + (y_n - 1)^2, \quad (4.20) \\
\text{f} (x) &= \sum_{i<j}^{n} \left[ (x_i^2 + x_j - 11)^2 + (x_i + x_j^2 - 7)^2 \right], \quad (4.21)
\end{align*}
\]

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

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

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

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

where \(n = 5\) and \(x \in [-5, 5]^5\).
Table 4.6: Comparison of two approaches to identify areas that cannot contain an index-1 saddle point. “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 7, respectively, column 7 refers to the percentage increase in the number of hyper-rectangles found with the proposed approach.

<table>
<thead>
<tr>
<th>Example</th>
<th>Function</th>
<th>Dim.</th>
<th>$L$</th>
<th>Rohn’s method</th>
<th>Alg. 7 (max 5 steps)</th>
<th>Improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Ackley</td>
<td>3</td>
<td>0.32</td>
<td>1072</td>
<td>1482</td>
<td>38</td>
</tr>
<tr>
<td>8</td>
<td>Levy</td>
<td>5</td>
<td>0.9</td>
<td>1025</td>
<td>1803</td>
<td>75</td>
</tr>
<tr>
<td>9</td>
<td>Himmelblau</td>
<td>5</td>
<td>2</td>
<td>1058</td>
<td>1189</td>
<td>12</td>
</tr>
</tbody>
</table>

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 7. Furthermore, the solution given by the local search method can be used in a stopping criterion since if the lower bound of $\lambda_{n-1}$ 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 $\lambda_{n-1}$. Finally, in the same way described above we can use the algorithm to identify convex areas ($\lambda_n \geq 0$) and index-1 areas ($\lambda_n < 0$ and $\lambda_{n-1} > 0$).
4.7 Summary

In this Chapter we have presented a Branch-and-Bound algorithm (MBB) for the bounding of individual (interval) eigenvalues of symmetric interval matrices. The branching occurs on the interval entries of the matrix while we use Rohn’s theorem (extension of Weyl’s theorem) for obtaining improving bounds. We have also used local methods for the upper bounding problem, which as shown by the examples, can accelerate the convergence of the algorithm. We have presented two such local methods, one taken from the literature and one developed here, and presented a comparison of their performances. Although in the worst-case scenario the complexity of the MBB is prohibiting, as shown in practise the algorithm does converge in a number of cases and even when it does not converge it still provides a significant reduction with respect to the initial bound widths. Therefore, the algorithm can be practical for small-sized problems, where the size of the problem depends not on the size of the matrix but on the number of interval entries of the matrix. Note also that this is the only method (to the best of knowledge) that deals with the general problem and can provide bounds even when the ranges of individual eigenvalues overlap. Moreover, as shown, a small number of iterations of the algorithm can be potentially used in order to significantly improve the performance of the Rohn method presented in the previous chapter, as a test for identifying non-index-1 symmetric interval matrices.
5 Tighter αBB relaxations through a refinement scheme for the scaled Gerschgorin theorem.

5.1 Introduction

The αBB algorithm [21, 8, 2, 1] is a branch-and-bound algorithm which is based on creating convex underestimators for general twice-continuously differentiable ($C^2$) functions. The tightness of the underestimator plays a key role in the efficiency of the algorithm. In the αBB method, the underestimator of a $C^2$ term or function is obtained by adding an appropriate quadratic term to the original expression. The validity of the underestimator depends on the calculation of the so-called α values, which must be chosen appropriately in order to ensure convexity. One must take care, however, not to be over-conservative by selecting α values that are larger than needed as the smaller the α values the tighter the underestimator is with respect to the original function.
A number of methods for the calculation of $\alpha$ values that are rigorously valid, i.e., such that the underestimator is guaranteed to be convex have been presented in the literature \cite{26, 2, 31, 31}. It is usual, but not necessary, for a trade-off between tightness of the underestimator and computational cost to exist. A comparative study among different methods for calculating $\alpha$ values for the original $\alpha$BB underestimator as well as methods that employ different underestimators \cite{2, 31, 4, 5, 56, 43} has been presented by Guzman et al. \cite{19}.

One important aspect of the choice of $\alpha$ values is with relation to the so-called cluster problem \cite{15}. The cluster problem describes the situation where a branch-and-bound algorithm creates a large number of unfathomed boxes around a solution because it creates nodes much faster than it fathoms. This effect is of course dependent on the quality of the underestimator and can significantly impact the performance of the algorithm. As shown by Wechsung et al. \cite{84}, improving the $\alpha$ values can be critical with respect to the clustering effect during execution of the $\alpha$BB algorithm.

Motivated by the above observations, we introduce a “refinement” algorithm, based on Haynsworth’s theorem \cite{23, 7}, to improve the $\alpha$ values given by the scaled Gerschgorin method \cite{2}. Although the algorithm can be applied to improve the $\alpha$ values given by any of the methods used in the original $\alpha$BB method (see \cite{2}) we choose the scaled Gerschgorin method because it usually gives good $\alpha$ values, it is computationally cheap and the use of a different $\alpha$ value for each variable (non-uniform shift) allows for more flexibility than other uniform shift methods.
The algorithm is tested on a number of randomly generated (symmetric) interval matrices and randomly taken interval Hessians from test functions. We use as a measure of tightness the maximum separation distance between the underestimators and the original function.

In Section 5.2 we begin by briefly presenting the $\alpha$BB underestimator for general $C^2$ functions and the scaled Gerschgorin method for calculating $\alpha$ values for the underestimator. In Section 5.3 we state Haynsworth’s theorem which is the basis of our new method. In Section 5.4 we present the refinement algorithm. We begin with an example to help the reader understand how we use Haynsworth’s theorem for our purpose. We then give a pseudocode form of the algorithm and close the section with another example where we apply the refinement algorithm. In Sections 5.5 and 5.6 we present the results of comparing the scaled Gerschgorin method and the refinement method. In Section 5.7 we present results from randomly generated symmetric interval matrices while in Section 5.8 results from Hessian matrices taken from test functions. Finally, we give a brief summary in Section 5.7.

5.2 The $\alpha$BB underestimator and the scaled Gerschgorin method

Given a general nonlinear function, $f \in C^2$, a convex underestimator, $F(x) = f(x) + q(x)$, of $f$ over a given hyper-rectangular domain $X = [[x_1, \bar{x}_1], \ldots, [x_n, \bar{x}_n]]^T$
is constructed within the αBB algorithm [61, 53, 21, 14], where

\[ q(x) = \sum_{i=1}^{n} \alpha_i (x_i - \bar{x}_i)(\overline{x}_i - x_i), \quad \alpha_i \geq 0, \ i = 1, \ldots, n. \tag{5.1} \]

Note that \( q(x) \leq 0, \ \forall x \in X \) and thus \( F(x) \) is indeed an underestimator of \( f(x) \) over that domain. The \( \alpha \) values have to be determined so as to ensure \( F \) is convex. This is accomplished with the use of the interval Hessian, \( [H_f] \) over the hyper-rectangular domain of interest. The interval Hessian matrix \( [H_f] \) is obtained by constructing the matrix \( H_f \) of second-order derivatives of \( f \) and deriving an interval enclosure \( [h_{ij}, \overline{h_{ij}}] \) for each element \( h_{ij}(x) \) over the domain \( X \). In the scaled Gerschgorin method, [2] the \( \alpha \) values are calculated as

\[ \alpha_i = \max \left\{ 0, -\frac{1}{2} \left( \frac{1}{h_{ii}} - \sum_{j \neq i} \max \{|h_{ij}|, |\overline{h_{ij}}|\} \frac{k_j}{k_i} \right) \right\}, \tag{5.2} \]

with \( k_i, \ i = 1, \ldots, n, \) being positive integers. An important feature of the αBB underestimator is that the maximum separation distance between \( f(x) \) and the underestimator \( F(x) \) over \( X \) is explicitly given by

\[ \max_{x \in X} D(x) = \max_{x \in X} (f(x) - F(x)) = \sum_{i=1}^{n} \alpha_i \frac{(\overline{x}_i - x_i)^2}{4}. \tag{5.3} \]

We can see from Eq. (5.3) that even if the \( \alpha \) values were not to improve as we subdivide the domain, the maximum separation distance would nevertheless improve quadratically. This is an important feature of the αBB underestimator which relates
A theoretical analysis of the cluster problem was first carried out in [13]. This analysis showed that the relaxations in a branch-and-bound algorithm must have at least second-order convergence to “avoid” the cluster effect. Note that we say (we use the definition given in [84]) that the order of convergence of a convex relaxation, \( F_X : X \to \mathbb{R}^n \), of a function, \( f : B \to \mathbb{R}^n \), where \( X \) is a hyper-rectangular subset of \( B \subseteq \mathbb{R}^n \) is \( \beta \geq 1 \) if there exists \( K > 0 \) such that

\[
\min_{x \in X} f(x) - \min_{x \in X} F_X(x) \leq K w(X)^\beta, \quad \forall X \in B,
\]

where \( w(X) = \max_{i=1,...,n} (\bar{x}_i - \underline{x}_i) \). It is easy to see that the \( \alpha \)-BB relaxation has convergence order of \( \beta = 2 \). In a later paper [84], it was shown that the pre-factor of the convergence order also plays a crucial role. For the \( \alpha \)-BB algorithm, the pre-factor corresponds to the \( \alpha \) values. Therefore, an improvement on these values could have a significant effect on the performance of the \( \alpha \)-BB algorithm.

As is evident from Eq. (5.3) we would like to make the \( \alpha \) values as small as possible while ensuring that the Hessian of \( F(x) \), \( H_F(x) = H_f(x) + D \) where \( D \) is the diagonal matrix with diagonal entries \( d_i = 2\alpha_i \), is positive semi-definite over the area of interest. With the help of Haynsworth’s theorem we can improve (reduce) the \( \alpha \) values obtained by the scaled Gerschgorin method.

In the following section we briefly revisit the definition of the inertia of symmetric matrices as well as Haynsworth’s theorem which were first introduced in Chapter 3.
5.3 Haynsworth’s theorem

The inertia of a symmetric matrix is defined as follows:

**Definition 5.3.0.1** (Inertia of a symmetric scalar matrix) *Given a symmetric matrix $M$, the inertia of $M$, $\text{In}(M)$, is the triplet $(\pi(M), \nu(M), \delta(M))$ of the numbers of positive, negative and zero eigenvalues of $M$ respectively.*

Haynsworth’s theorem, which is the basis of the refinement method, has as follows:

**Theorem 5.3.0.2** (Haynsworth [23]) *Given a symmetric matrix $M$ partitioned in the form,

$$
M = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}
$$

and assuming $A$ is non-singular, then $\text{In}(M) = \text{In}(A) + \text{In}(C - B^T A^{-1} B)$.*

Theorem 5.3.0.2 can be used recursively for the complete calculation of the inertia of a scalar matrix ([9]) and therefore for revealing whether the matrix is positive semi-definite or not. This can be accomplished by choosing $A$ to be a single diagonal entry, noting its sign, then calculating the Schur complement, $C - B^T A^{-1} B$ and repeating the process on this newly formed matrix. Assume for example that for a given $n \times n$ symmetric matrix $M$, we repeat this procedure $n$ times and find

$$
\text{In}(M) = \text{In}(A_1) + \ldots + \text{In}(A_{n-1}) + \text{In}(A_n),
$$

(5.5)

with $A_i > 0$ for $i = 1, \ldots, n-1$ and $A_n \geq 0$ where $A_i$ is the entry $m_{11}^{(i)}$ of the $i$-th Schur complement, $M_i$, with $M_1$ being the initial matrix $M$. Then by Theorem
we conclude that the matrix $M$ is positive semi-definite.

For scalar matrices we can always proceed to calculate the complete inertia even if at some step there is no non-zero diagonal entry that can be chosen (see [2] for details). An extension of the recursive use of Theorem 5.3.0.2 for the calculation of the inertia of symmetric interval matrices has been presented in [65]. In this work, however, we are not interested in calculating the inertia but rather guaranteeing semi-definiteness. In a similar way, the extension of the recursive procedure for determining the positive semi-definiteness of scalar matrices to the case of interval matrices is straightforward.

For example assume that we have a symmetric interval matrix $[M]$ and that by following the same procedure as in the scalar case but now using interval arithmetic for the calculation of each subsequent (interval) Schur complement we find

$$In([M]) = In([A_1]) + \ldots + In([A_{n-1}]) + In([A_n]),$$

(5.6)

with $[A_i] = \left[ m_{11}^{(i)}, m_{11}^{(n)} \right]$ being strictly positive intervals for $i = 1, \ldots, n - 1$ and $m_{11}^{(n)} \geq 0$. Note that when we multiply two interval matrices, $[A]$ and $[B]$, we have that $[C] = [A][B] \supseteq \{ AB : A \in [A] \text{ and } B \in [B] \}$ thus each time we calculate an interval Schur complement an overestimation takes place. Therefore, in the above scenario, we can safely conclude that the interval matrix $[M]$ is positive semi-definite (i.e. all the symmetric scalar matrices contained in $[M]$ are positive semi-definite).

In the next section we begin with an example of this procedure in order to help
the reader understand how we utilize Theorem 5.3.0.2.

5.4 The refinement algorithm

Consider a 3-dimensional function $f : B \subset \mathbb{R}^3$. We want to construct the $\alpha$BB underestimator over an area $X \subseteq B$. After calculation of the interval Hessian $[H_f]$ over $X$ and calculation of the $\alpha$ values using Eq. 5.2, we consider the convex underestimator $F(x) = f(x) + q(x)$ with its Hessian,

$$[H_F] = \begin{bmatrix}
[h'_{11}] & [h_{12}] & [h_{13}] \\
[h_{21}] & [h'_{22}] & [h_{23}] \\
[h_{31}] & [h_{32}] & [h'_{33}]
\end{bmatrix},$$

where $[h_{ij}] = \left[ \frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{x=X}$ is calculated using interval arithmetic and $[h'_{ii}] = [h_{ii}] + d_i$ where $d_i = 2\alpha_i$ with $\alpha_i$ calculated using Eq. (5.2) for $i = 1, 2, 3$. Now assume that after applying Haynsworth’s theorem recursively on $[H_F]$ we get:

at step 1,

$$[h'_{11}] > 0,$$

(5.8)

at step 2,

$$[h'_{22}] - \frac{[h_{12}]^2}{[h_{11}]} > 0$$

(5.9)
and finally at step 3,

\[
[h'_{33}] - \frac{[h_{13}]^2}{[h'_{11}]} - \left(\frac{[h_{23}] - \frac{[h_{12}]}{[h_{21}]} \frac{[h'_{22}] - \frac{[h_{12}]}{[h_{11}]} \frac{[h'_{22}][h'_{23}]}{[h_{23}]}}{[h_{12}][h_{21}]}}{[h'_{22}] - \frac{[h_{12}]}{[h_{11}]} \frac{[h'_{22}][h'_{23}]}{[h_{23}]}}\right)^2 = [R_3, \bar{R}_3] \geq 0. \tag{5.10}
\]

Notice that the left-hand sides of inequalities (5.8), (5.9), (5.10) are the (interval) entries \([a_{11}]\) of each subsequent (interval) Schur complement starting with matrix 5.7. In the above scenario, based on Theorem 5.3.0.2 we would know that the interval matrix is positive semi-definite.

It is interesting to examine the value of the lower bound, \(r_3 = R_3\). Notice that if \(r_3\) is positive we can reduce \([h'_{33}]\) (i.e. reduce \(d_3\)) by any value in \([0, \min\{r_3, d_3\}]\) without affecting inequalities (5.8) and (5.9), thus maintaining the positive semi-definiteness of the interval Hessian \(H_F\). We refer to the value \(r_3\) (or \(r_i\) in the general case) as the residual. After we reduce \([h'_{33}]\) by a certain value we can interchange the second and third rows and columns of the new matrix and repeat the same process calculating the residual \(r_2\) and reduce, if possible, \([h'_{22}]\). Similarly we calculate \(r_1\) and reduce \([h'_{11}]\). We give a pseudocode of the refinement method in Algorithm 11. Note that the input of the algorithm is the interval Hessian, \([H_F] = [H_f] + D\), of the \(\alpha\)BB underestimator with \(d_i = 2\alpha_i\) calculated by Eq. (5.6). However, as mentioned in the Introduction, the input matrix can be the interval Hessian of the underestimator where the \(\alpha\) values have been calculated with any other method.
**Algorithm 10** Refinement algorithm \((O(n^4))\)

1: Inputs: \(n \times n\) interval Hessian, \([H_F]\), of the αBB underestimator.

2: Initialize \(m_i = 0\) for \(i = 1, ..., n\).

3: for \(i = 0, 1, ..., n - 1\) do

4: Calculate residual \(r_{n-i}\). If \(r_{n-i} \leq 0\) (or if at any step during calculation of \(r_{n-i}\) the result is non-positive) stop.

5: Reduce the diagonal entry \(h_{n-i,n-i}\) by a value \(m_{n-i} \in [0, \min\{r_{n-i}, d_{n-i}\}]\).

6: If \(i > 0\), interchange rows \(n-i\), \(n\) and columns \(n-i\), \(n\) of the input matrix.

7: end for

8: The new \(α\) values are \(α'_i = (d_i - m_i)/2\).

Now the question arises of how to choose a value for \(m_{n-i} \in [0, \min\{r_{n-i}, d_{n-i}\}]\) at step 5 of Algorithm 10. We could choose at the first iteration \(m_n = \min\{r_n, d_n\}\). However, it might be wiser to “spread” the reduction to all the diagonal elements (if possible). We consider three approaches:

\[
m_{n-i} = \min \left\{ \frac{r_{n-i}}{n-i}, d_{n-i} \right\}, \quad i = 0, 1, ..., n - 1
\]

(5.11)

\[
m_{n-i} = \min \left\{ \frac{r_{n-i}}{n-i} + w_{n-i} \left( r_{n-i} - \frac{r_{n-i}}{n-i} \right), d_{n-i} \right\},
\]

where \(w_{n-i} = \frac{d_{n-i}}{\sum_{j=1}^{n-i} d_j}\) and \(i = 0, 1, ..., n - 1\)

(5.12)
\[ m_{n-i} = \begin{cases} 
\min \left\{ \frac{d_{n-i}}{\sum_{j=1}^{n-1} d_j} r_{n-i}, d_{n-i} \right\}, & i = 0, 1, \ldots, n-2 \\
0, & \text{if } d_1 = 0 \text{ and } i = n - 1 \\
r_1, & \text{otherwise} 
\end{cases} \] (5.13)

We will refer to option (5.11) as the “Shared” option, to (5.12) as the “Extra-weighted” option and to (5.13) as the “Weighted” option. In the Shared option, the current reduction is equal to the current residual divided by the number of remaining diagonal entries to be reduced. In the Extra-weighted option the current reduction has the same value as in the Shared option plus a weighted portion \((w_{n-i})\) of what remains if we subtract this value from the residual. In the Weighted option the reduction value is a portion of the current residual which is given by to the ratio of \(d_{n-i}\) over the sum of the remaining \(d_j\) values to be reduced.

Let us now give an example of the refinement algorithm so that it may become clearer to the reader. We will use the Shared reduction option for this example. Consider the \(3 \times 3\) (symmetric) interval matrix (the fact that the diagonal elements of the example matrix are scalar bears no significance. In fact, in practice, only the lower bounds of the diagonal elements need to be considered (see Lemma 3.4.5.1 in Chapter 3),

\[
[H_f] = \begin{bmatrix}
[6, 7] & [5, 6] & -4
\end{bmatrix}.
\] (5.14)
Calculating the $\alpha$ values using Eq. (5.2) (with $k_i = 1$, for $i = 1, 2, 3$) we have, $\alpha_1 = 8$, $\alpha_2 = 6$, $\alpha_3 = 8.5$. The hypothetical underestimator would have the interval Hessian,


Using Haynsworth’s theorem on $[H_F]$ (eq. 5.8-5.10) we have, at step 1: 11, at step 2: [8.54,9.18] and at step 3: [6.31,9.18]. Therefore $r_3 = 6.31$ and we now reduce the entry $h_{33}$ of $[H_F]$ by $m_3 = r_3/3 = 2.1$ and we interchange rows and columns 2,3 resulting with the matrix,


Again, using eq. 5.8-5.10 we get 11, [6.43,7.62] and [5.58,8.39] respectively. Thus now $m_2 = r_2/2 = 2.79$ and the new matrix is,


Performing the same calculations once more we get 7.21, [5.89,742], [4.67,8.79]
and so finally \( m_1 = r_1 = 4.67 \). The reduced \( \alpha \) values are: 
\[
\alpha'_1 = \alpha_1 - m_1/2 = 5.665,
\alpha'_2 = \alpha_2 - m_2/2 = 4.605 \text{ and } \alpha'_3 = \alpha_3 - m_3/2 = 7.45.
\]

Although we cannot calculate actual minima in this case, since our example matrix was not derived from a specific function, we can measure the improvement obtained with the reduced values, \( \alpha'_i \), using Eq. (5.3). More specifically, we can set \((x_i - x_i)^2 = 1, i = 1, 2, ..., n\) and consider the percentage of improvement with respect to the (hypothetical) maximal separation distance,

\[
I = 100 \left( 1 - \frac{n \sum_{i=1}^{n} \alpha'_i}{n \sum_{i=1}^{n} \alpha_i} \right) \%	ag{5.18}
\]

The value of \( I \) can vary from 0\% (no reduction at all in the \( \alpha \) values), up to 100\% (the initial matrix is identified as positive semi-definite). For our example we have \( I = 21.2\% \), meaning that (by this measure) the refinement led to a 21.2\% reduction in the maximal separation distance.

Note that, we could simply apply the recursive procedure given in Eq. (5.6) on the initial Hessian matrix to determine whether it is positive semi-definite. This concept was proposed in [57]. In this work, however, we are interested in reducing the \( \alpha \) values and not identifying only whether the initial interval Hessian is positive semi-definite or not.
5.5 Results on random symmetric interval matrices.

In this section we present results from the application of the refinement algorithm on randomly generated symmetric interval matrices. We have generated four groups of one thousand random matrices each with dimension 3, 4, 5 and 7 respectively and with the interval bounds in each matrix varying from −10 to 10. For each matrix in each group we apply Algorithm 10 with all three different reduction options (Eq. (5.11)-(5.13)) and we calculate the percentage improvement (reduction) in the maximum separation distance, $I$, given by Eq. (5.18). For each group of matrices we plot three histograms of the $I$ values obtained after applying the refinement algorithm with each reduction option respectively in Figures 5.1-5.4. Furthermore, in Table 5.5 we give the mean $I$ value attained by each reduction option in each group of random matrices.

Table 5.1: Mean $I$ values for each reduction option in each group of random matrices.

<table>
<thead>
<tr>
<th>Option/Dimension</th>
<th>D=3</th>
<th>D=4</th>
<th>D=5</th>
<th>D=7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared (5.11)</td>
<td>6.9%</td>
<td>10.8%</td>
<td>12.8%</td>
<td>15.3%</td>
</tr>
<tr>
<td>Extra-weighted (5.12)</td>
<td>7.4%</td>
<td>11.3%</td>
<td>13.3%</td>
<td>16.3%</td>
</tr>
<tr>
<td>Weighted (5.13)</td>
<td>6.2%</td>
<td>9.3%</td>
<td>10.5%</td>
<td>11.2%</td>
</tr>
</tbody>
</table>

We can make the following observations. First, as the matrix dimension increases the mean $I$ values improve (increase) for all cases. Second, the Shared (5.11) and
Extra-weighted (5.12) options perform significantly better than the Weighted option (5.13) in all four cases while the Extra-Weighted option performs slightly better than the Shared option. For a more detailed analysis of the performance of the Shared and Extra-weighted options, we plot a histogram (Figure 5.5) of the values $I_2 - I_1$, where $I_1$ and $I_2$ are the values shown in Figures 5.4 a) and b) respectively. As can be seen, the majority of cases in Figure 5.5 are positive. Therefore, we can conclude that the Extra-weighted option might be preferable overall.
Figure 5.1: Histogram of the $I$ values \[(5.11)\] for the 1000 $3 \times 3$ random matrices using a) the Shared option \[(5.13)\], b) the Extra-weighted option \[(5.12)\] and c) the Weighted option \[(5.13)\].
Figure 5.2: Histogram of the $I$ values (5.18) for the 1000 $4 \times 4$ random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).
Figure 5.3: Histogram of the $I$ values for the 1000 $5 \times 5$ random matrices using a) the Shared option, b) the Extra-weighted option and c) the Weighted option.
Figure 5.4: Histogram of the $I$ values (5.18) for the 1000 $7 \times 7$ random matrices using a) the Shared option (5.11), b) the Extra-weighted option (5.12) and c) the Weighted option (5.13).
Figure 5.5: Comparison of the Shared (5.11) and Extra-weighted (5.12) reduction options: histogram of the values $I_2 - I_1$ where $I_1$ (Shared) and $I_2$ (Extra-weighted) are the values shown in Figures 5.4 a) and b) respectively.

5.6 Results on random interval Hessian matrices.

In this section we present results from the application of the refinement algorithm on symmetric interval Hessian matrices calculated over random domains of the following three test functions:

Griewank:

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}), \ n = 4, \ x \in [-5, 5]^4.$$  \quad (5.19)
Levy:
\[
f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2, \quad \text{(5.20)}
\]
\[
y_i = 1 + (x_i - 1)/4, \ n = 5, \ x \in [-5, 5]^5.
\]

Himmelblau (extension to \(n\) dimensions):
\[
f(x) = \sum_{i<j}^{n} [(x_i^2 + x_j - 11)^2 + (x_i + x_j^2 - 7)^2], \ n = 5, \ x \in [-5, 5]^5. \quad \text{(5.21)}
\]

For each function we calculate three groups of one thousand Hessian matrices each, over random hyper-rectangular domains with randomly chosen centres and with sides of randomly varying length within \((0, L)\) for a) \(L = 2\), b) \(L = 1\) and c) \(L = 0.2\). We then calculate the \(\alpha_i\) values using the scaled Gerschgorin method (Eq. (5.2) ) with \(k_i = x_i - x_i\). Next we calculate the refined \(\alpha'_i\) values using the Extra-weighted reduction option (Eq. (6.12) ) and we plot a corresponding histogram of the values

\[
I = 100 \left( 1 - \frac{\sum_{i=1}^{n} \alpha_i'(x_i - x_i)^2}{\sum_{i=1}^{n} \alpha_i(x_i - x_i)^2} \right) \%.
\quad \text{(5.22)}
\]

The results are given in Figures 5.6-5.8. We also give the mean \(I\) value attained for each test function for each value of \(L\) in Table 5.3.
Table 5.2: Mean $I$ values for each test function for each $L$ value.

<table>
<thead>
<tr>
<th>Test func./L-value</th>
<th>L=2</th>
<th>L=1</th>
<th>L=0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Griewank (4D)</td>
<td>14.2%</td>
<td>14.0%</td>
<td>14.1%</td>
</tr>
<tr>
<td>Levy (5D)</td>
<td>0.5%</td>
<td>3.3%</td>
<td>11.5%</td>
</tr>
<tr>
<td>Himmel. (5D)</td>
<td>21.5%</td>
<td>27.4%</td>
<td>32.6%</td>
</tr>
</tbody>
</table>

As mentioned earlier, the results differ for each case since the Hessians have a certain structure and entry values. In Figure 5.6 (Griewank Hessians) the refinement method results in an improvement of approximately 14% regardless of the value of $L$. In Figure 5.7 (Levy Hessians) we see that the refinement method is successful only when $L = 0.2$ with average improvement of 11.5% percent. In Figure 5.8 (extended Himmelblau Hessians) the refinement algorithm performs well for all cases with increasing improvement, 21.5%, 27.4% and 32.6% as the value of $L$ becomes smaller. Note that there are cases with values equal to zero. This means that the interval Hessian of the initial function was positive semi-definite and there would be no need for constructing the underestimator. Finally, in Figure 5.9 we once more compare the Shared reduction option and the Extra-weighted reduction option by comparing the $I$ values (given by Eq. (5.22)) they produce when applied to the Hessians corresponding to Figure 5.7c) (Griewank Hessians, $L = 2$). Again, the Extra-weighted option is better.
Figure 5.6: Histogram of the $I$ values for interval Hessians of the Griewank function on randomly selected hyper-rectangular areas with a) $L = 2$, b) $L = 1$ and c) $L = 0.2$. 

![Histogram of $I$ values for interval Hessians of the Griewank function on randomly selected hyper-rectangular areas with different values of $L$.]
Figure 5.7: Histogram of the $I$ values for interval Hessians of the Levy function on randomly selected hyper-rectangular areas with a) $L = 2$, b) $L = 1$ and c) $L = 0.2$. 

(a) $L = 2$

(b) $L = 1$

(c) $L = 0.2$
Figure 5.8: Histogram of the $I$ values for interval Hessians of the extended Himmelblau function on randomly selected hyper-rectangular areas with

a) $L = 2$, b) $L = 1$ and c) $L = 0.2$. 

- a) $L = 2$
- b) $L = 1$
- c) $L = 0.2$
5.7 Summary

In this Chapter we have presented an algorithm \( O(n^4) \) for further improving the \( \alpha \) values of the \( \alpha \)BB underestimator given by the scaled Gerschgorin method. In a previous study, Guzman et al. \[19\] have shown that among a number of methods the scaled Gerschgorin method was the most cost-effective. We have compared the...
two methods using as measure the maximal separation distance between the under-
estimator and the underestimating function. The results show that the refinement 
method can significantly reduce the $\alpha$ values and therefore potentially improve the 
performance of the $\alpha$BB. However, further investigation is needed in order to verify 
this hypothesis.
6 Conclusions and future work

6.1 TS location

In Chapter 3 we considered the problem of enclosing all transition states (TSs) of general nonlinear functions in $C^2$ using global deterministic methods. We introduced five tests that can be applied prior to the bounding step of branch-and-bound algorithm. These tests help to identify areas of the search space which do not contain any TSs or may contain at most one. In the first case we fathom/remove the area while in the second we perform a local search and if a solution is found we then fathom the area. With the tests we aim to focus the computational effort on the location of TSs rather than the identification of all critical points. We have implemented this approach within the αBB algorithm and presented the successful application of the proposed tests to a number of low-dimensional problems in $C^3$, with up to six variables. The problems typically exhibit numerous stationary points.

The results indicate that the addition of the tests can reduce the computational time significantly while locating all the transition states successfully. Furthermore,
the use of a local search in areas that are identified to contain at most one TS is found to be advantageous, reducing both CPU time and iteration number. We note that the proposed tests can be used within any branch-and-bound algorithm or within the interval Newton method and that, with the exception of the $2 \times 2$ inertia test, they can be altered in order to locate any index-k critical point. The RecIn/xRecIn tests are particularly effective for all problems considered.

The use of the tests is a useful step towards the application of a branch-and-bound algorithm to the identification of transition states for larger problems: within the $\alpha$BB algorithm, the tests can be implemented at relatively low cost because the required interval Hessian matrix is computed implicitly as part of the underestimation procedure. Thus, the overhead arising from the tests can be kept low, while achieving a reduction in iteration number. As future work we aim to apply the proposed approach to real case studies taken from the field of chemical engineering. There is also a potential for a combined use of the proposed method with local methods. For example, in a case where the proposed method does not converge after a given number of maximum iterations or a CPU time limit, as a second stage, we can initiate multiple local searches where the starting points fall inside areas of the initial domain that have not been removed by the end of the execution of the Branch-and-Bound method.
6.2 Interval Matrix Branch and Bound

In Chapter 4 we 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 $10 \times 10$ 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 [30] and one proposed here. The method by Hladík et al. ([31]) 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.

As future work we would consider a further investigation of the performance of the algorithm using a large number of random matrices in order to acquire more accurate statistics and perform an average case complexity analysis.

6.3 Refinement method

We have presented a refinement method which we use in conjunction with the scaled Gerschgorin method in order to improve (reduce) the $\alpha$ values needed for the convex underestimator of the deterministic global optimization algorithm $\alpha$BB. However, the refinement method can be utilized with other available methods for the calculation of the $\alpha$ values.

We have applied our algorithm on randomly generated symmetrical interval matrices as well as interval Hessian matrices taken from test functions. In order to compare the scaled Gerschgorin method and the refinement method we used as a measure the maximal separation distance of the underestimator.

In the experiments with the randomly generated matrices we used four groups of matrices with dimension 3, 4, 5 and 7 respectively and with each group consisting of a thousand matrices. The results showed that the refinement method improved the maximal separation distance by an average of 7%, 11%, 13% and 16% for each group respectively.

In the experiments with the interval Hessian matrices we used three test functions: 3D-Griewank, 5D-Levy and a 5D-extension of the Himmelblau function. For each
test function we calculated three groups of a thousand interval Hessians each. The Hessians were calculated over randomly chosen hyper-rectangular areas with sides of length $(0, L)$ where $L = 2$, $L = 1$ and $L = 0.2$ for each group respectively. As is natural, the results differ for each function. For the Griewank function the results were similar regardless of the value of $L$ with an average improvement of approximately 14%. For the Levy function there was no significant improvement for $L = 2$ and $L = 1$. However for $L = 0.2$ the improvement was 11.5%. Finally, for the extended Himmelblau function we had 21.5%, 27.4% and 32.6% improvement for $L = 2$, $L = 1$ and $L = 0.2$ respectively with many of cases having 100% improvement.

Furthermore, we have tested three different reduction options for step 5 of the refinement algorithm and based on the results we have concluded that the Extra-weighted option Eq. (5.12) performs best.

From the above we conclude that the refinement method can result in a considerable improvement with respect to the maximal separation distance. As a future work, it remains to be seen whether the algorithm is cost-effective, when integrated for use into the $\alpha$BB algorithm and if yes, up to what extent.
List of publications

Parts of several chapters in this thesis have appeared in the following publications:


Bibliography


[42] Land, A.H., Doig, A.G.: An automatic method of solving discrete program-


[48] Lin, Y., Stadtherr, M.A.: Locating stationary points of sorbate-zeolite poten-


for the 2D XY model: Minima, transition states, and pathways. The Journal of Chemical Physics 139(19), 194503 (2013)


