2006

Construction of symmetric matrices with prescribed spectra

Viet H. Nguyen
San Jose State University

CONSTRUCTION OF SYMMETRIC MATRICES WITH PRESCRIBED SPECTRA

A Thesis

Presented to

The Faculty of the Department of Mathematics

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Viet H. Nguyen

August 2006

APPROVED FOR THE DEPARTMENT OF MATHEMATICS

Dr. Wasin So

Dr. Jane Day

Dr. Timothy Hsu

APPROVED FOR THE UNIVERSITY

06/23/06

ABSTRACT

CONSTRUCTION OF SYMMETRIC MATRICES WITH PRESCRIBED SPECTRA

by Viet H. Nguyen

This thesis focuses on the idea of Inverse Eigenvalue Problem (IEP) that concerns the reconstruction of symmetric matrices with specific properties from prescribed spectra. The thesis begins with a brief history of IEP and its applications. Some definitions, theorems, and lemmas are reviewed in the first chapter. Then it introduces three inverse eigenvalue problems that involve the leading principal submatrix, the rank-1 perturbation, and the symmetric sum. The goal of this thesis is not only interested in the existence result, but it is also interested in the explicit construction algorithms. For each problem, it provides a proof of a necessary and sufficient condition under which the inverse eigenvalue problem has a solution. Then an algorithm based on the proof is given. Finally, a program in Matlab is included.

DEDICATION

To the memory of my mother, Le-Khanh T. Hoang, who gave me a love of life.

ACKNOWLEDGEMENTS

I would very much like to thank my advisor, Dr. Wasin So, for giving me the opportunity to work in a very interesting topic. Without his guidance and encouragement, the completion of this thesis would not have been possible. I would also like to thank the other members of my committee, Dr. Jane Day and Dr. Timothy Hsu, for their assistances and suggestions.

CONTENTS

CHAPTER 1

MOTIVATION AND BACKGROUND

This chapter consists of two sections: Motivation and Background. The first section gives a brief history of Inverse Eigenvalue Problem (IEP) and its applications. The other section reviews some definitions, lemmas, and theorems that will be useful for later chapters.

1.1 Motivation

While an eigenvalue problem concerns the computation of eigenvalues of a given matrix, an inverse eigenvalue problem concerns the reconstruction of a matrix from prescribed spectral data. The spectral data involved may consist of complete or only partial information of eigenvalues or eigenvectors. Also, the spectral data may involve a mixture of information about eigenvalues or eigenvectors. Furthermore, it is often necessary, for feasibility, to restrict the construction to special classes of matrices such as symmetric, rank-1, tridiagonal, leading principal, and so on. The objective of an inverse eigenvalue problem is to construct a matrix that maintains a certain specific structure as well as the given spectral property. This thesis focuses on the construction real symmetric matrices with desired eigenvalues.

An inverse eigenvalue problem has two fundamental questions: the theoretic issue on solvability and the practical issue on computability. Solvability concerns obtaining a necessary and sufficient condition under which an inverse eigenvalue problem has a solution. Computability concerns developing a procedure by which, knowing a priori that the given spectral data are feasible, a matrix can be constructed numerically. Both questions are difficult and challenging.

An inverse eigenvalue problem arises in a remarkable variety of applications, such as [CG01, pp. 1-10]

◇ Mathematical analysis: Inverse Sturm-Liouville problems.

◇ Numerical analysis: Preconditioning, Computing B-stable RK methods with real poles, and Gaussian quadratures.

◇ Applied physics: Compute the electronic structure of an atom from measure energy levels, Neutron transport theory.

◇ Applied mechanics and structure design: Construct a model of a (damped) mass-spring system with prescribed natural frequencies/modes.

◇ System identification and control theory: State/output feedback pole assignment problems.

1.2 Background

Let $M_n = M_n(\mathbf{R})$ denote the set of all $n \times n$ real matrices. All matrices in this paper are *real*.

Definition 1.2.1 (real symmetric matrix). A matrix $A = [a_{ij}] \in M_n$ is said to be *symmetric* if $A = A^T$. Here A^T denotes the transpose of A.

Definition 1.2.2 (real orthonormal set). Let $\{x_i\} \subseteq \mathbf{R}^n$ be a set of vectors. $\{x_i\}$ is an *orthonormal set* if

$$x_i^T x_j = \begin{cases} 0 & i \neq j, \\ 1 & i = j. \end{cases}$$

Definition 1.2.3 (real orthogonal matrix). A matrix $U \in M_n$ is said to be *orthogonal* if $UU^T = U^TU = I$. Here I is the identity matrix.

Theorem 1.2.4 (diagonalization of real symmetric matrix). *Let $A \in M_n$, then A is symmetric if and only if $A = U\Lambda U^T$, for a diagonal matrix $\Lambda \in M_n$ and an orthogonal matrix $U \in M_n$. The diagonal entries of Λ are the eigenvalues of A and the columns of U are the corresponding orthonormal eigenvectors of A.*

Proof. See [HJ85, pp. 171-172]. □

Lemma 1.2.5 (dimension formulas). *Given subspaces $S_1, S_2, S_3 \subseteq \boldsymbol{R}^n$, then*

(1) $dim(S_1 \cap S_2) + dim(S_1 + S_2) = dim\ S_1 + dim\ S_2$.

(2) $dim(S_1 \cap S_2) \geq dim\ S_1 + dim\ S_2 - n$.

(3) $dim(S_1 \cap S_2 \cap S_3) \geq dim\ S_1 + dim\ S_2 + dim\ S_3 - 2n$.

Proof. See [Axl97, pp. 33-34]. □

Theorem 1.2.6 (Rayleigh-Ritz for real symmetric). *Let $A \in M_n$ be real symmetric and let the eigenvalues of A be ordered as $\alpha_{max} = \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_{n-1} \geq \alpha_n = \alpha_{min}$. Then*

$$\alpha_1 x^T x \geq x^T A x \geq \alpha_n x^T x,\ \forall x \in \boldsymbol{R}^n$$
$$\alpha_{max} = \alpha_1 = \max_{x \neq 0} \frac{x^T A x}{x^T x} = \max_{x^T x = 1} x^T A x,$$
$$\alpha_{min} = \alpha_n = \min_{x \neq 0} \frac{x^T A x}{x^T x} = \min_{x^T x = 1} x^T A x.$$

Proof. See [HJ85, pp. 176-177]. □

Corollary 1.2.7. *Let $A \in M_n$ be real symmetric and let the eigenvalues of A be ordered as $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{n-1} \geq \alpha_n$ and orthonormal eigenvectors $u_1, \cdots, u_n$. Consider $\boldsymbol{S} = span\{u_{i_1}, \cdots, u_{i_r}\}$ where $1 \leq i_1 \leq i_2 \leq \cdots \leq i_r \leq n$, then*

$$\alpha_{i_1} x^T x \geq x^T A x \geq \alpha_{i_r} x^T x, \ \forall x \in \boldsymbol{S}.$$

□

Theorem 1.2.8 (Cauchy's interlacing inequalities). *Let*

$$A = \begin{pmatrix} B & C \\ C^T & D \end{pmatrix}$$

be an $n \times n$ real symmetric matrix and B be $m \times m$ $(m < n)$. Let the eigenvalues of A and B be $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{n-1} \geq \alpha_n$ and $\beta_1 \geq \beta_2 \geq \dots \geq \beta_{m-1} \geq \beta_m$, respectively. Then

$$\alpha_k \geq \beta_k \geq \alpha_{k+n-m}, \qquad k = 1, \dots, m.$$

Proof. See [HM87, pp. 352-353]. □

Theorem 1.2.9 (Lagrange interpolating polynomial). *If $x_0, x_1, \dots, x_n$ are $n+1$ distinct numbers and f is a function whose values are given at these numbers, then there exists a unique polynomial $P(x)$ of degree at most n with the property that*

$$f(x_k) = P(x_k), \qquad \text{for each } k = 0, 1, \dots, n.$$

This polynomial is given by

$$P(x) = f(x_0)L_{n,0}(x) + \dots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} \left(f(x_k)L_{n,k}(x)\right),$$

where

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_0)(x_k-x_1)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}$$
$$= \prod_{i=0, i\neq k}^{n} \frac{(x-x_i)}{(x_k-x_i)}.$$

Proof. See [BF97, pp. 107-110]. □

Lemma 1.2.10 (real symmetric *rank* 1 matrix). *Let $A \in M_n$ be an $n \times n$ real symmetric matrix with $rank(A) = 1$, then*

(1) A may be written in the form $A = \alpha x x^T$ where x is a unit vector in $\mathbf{R}^n$ and α is nonzero.

(2) A has exactly one nonzero eigenvalue α.

Proof.

Since A is real symmetric, by Theorem 1.2.4,

$$A = U \begin{pmatrix} \alpha & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix} U^T$$

where $\alpha \neq 0$ and U orthogonal.

This is equivalent to $A = \alpha x x^T$ where $x = U \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$. □

Definition 1.2.11 (positive definite matrix). An $n \times n$ real symmetric matrix A is said to be *positive definite* if $x^T A x > 0$ for all nonzero vectors $x \in \mathbf{R}^n$.

Notation: $A > 0$ means A is real symmetric and positive definite.

Definition 1.2.12 (positive semidefinite matrix). An $n \times n$ real symmetric matrix A is said to be *positive semidefinite* if $x^T A x \geq 0$ for all vectors $x \in \mathbf{R}^n$.

Notation: $A \geq 0$ means A is real symmetric and positive semidefinite.

Lemma 1.2.13 (eigenvalues of a positive definite/semidefinite matrix). *Let A be an $n \times n$ real symmetric matrix. If A is positive definite then all eigenvalues of A are positive. If A is positive semidefinite then all eigenvalues of A are nonnegative.*

Proof. Suppose λ_k is an eigenvalue of A where $k = 1, 2, \ldots, n$. Let $v_k \in \mathbf{R}^n$ be an eigenvector corresponding to λ_k. If $A > 0$, by Definition 1.2.11, we have

$$0 < v_k^T A v_k = v_k^T \lambda v_k = \lambda v_k^T v_k = \lambda_k |v_k|^2$$

Therefore, $\lambda_k > 0$ since $|v_k|^2 > 0$. Similarly, if $A \geq 0$, then $\lambda_k \geq 0$ since $v_k^T A v_k \geq 0$ (Definition 1.2.12) and $|v_k|^2 > 0$. □

Theorem 1.2.14 (Weyl's inequalities). *Let A, $B \in M_n$ be real symmetric matrices, and let the eigenvalues of A, B, and $A + B$ be arranged in non-increasing order as*

$$\alpha_{max} = \alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_{n-1} \geq \alpha_n = \alpha_{min}$$

$$\beta_{max} = \beta_1 \geq \beta_2 \geq \ldots \geq \beta_{n-1} \geq \beta_n = \beta_{min}$$

$$\gamma_{max} = \gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_{n-1} \geq \gamma_n = \gamma_{min}$$

Then for every pair of integers i, j such that $1 \leq i, j \leq n$ and $i + j \leq n + 1$, we have

$$\gamma_{i+j-1} \leq \alpha_i + \beta_j$$

and for every pair of integers i, j such that $1 \leq i, j \leq n$ and $i + j \geq n + 1$, we have

$$\gamma_{i+j-n} \geq \alpha_i + \beta_j$$

Proof. [IIM87, pp. 352-353]

Let $\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_n$ be real *orthonormal* eigenvectors (Definition 1.2.2) of A with respect to $\{\alpha_i\}$, $\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n$ be real *orthonormal* eigenvectors of B with respect to $\{\beta_i\}$, and $\overrightarrow{w}_1, \overrightarrow{w}_2, \cdots, \overrightarrow{w}_n$ be real *orthonormal* eigenvectors of $A+B$ with respect to $\{\gamma_i\}$.

Define

$$S_1 = Span\{\overrightarrow{u}_i, \cdots, \overrightarrow{u}_n\} \text{ so dim } S_1 = n - i + 1,$$

$$S_2 = Span\{\overrightarrow{v}_j, \cdots, \overrightarrow{v}_n\} \text{ so dim } S_2 = n - j + 1,$$

$$S_3 = Span\{\overrightarrow{w}_1, \cdots, \overrightarrow{w}_{i+j-1}\} \text{ so dim } S_3 = i + j - 1.$$

By Lemma 1.2.5, $\dim(S_1 \cap S_2 \cap S_3) \geq \dim S_1 + \dim S_2 + \dim S_3 - 2n = 1$. Therefore, there exists $\overrightarrow{x} \in (S_1 \cap S_2 \cap S_3)$, where $\overrightarrow{x} \neq 0$ and $x^T x = 1$. Because $\overrightarrow{x} \in S_3$, $\gamma_{i+j-1} \leq x^T(A+B)x$ by Corollary 1.2.7. Because $\overrightarrow{x} \in S_1 \cap S_2$, $x^T A x \leq \alpha_i$ and $x^T B x \leq \beta_j$.

Hence, for $1 \leq i, j \leq n$ and $i + j \leq n + 1$, we obtain

$$\gamma_{i+j-1} \leq x^T(A+B)x = x^T A x + x^T B x \leq \alpha_i + \beta_j.$$

Similarly, define

$$S_1 = Span\{\overrightarrow{u}_1, \cdots, \overrightarrow{u}_i\} \text{ so dim } S_1 = i,$$

$$S_2 = Span\{\overrightarrow{v}_1, \cdots, \overrightarrow{v}_j\} \text{ so dim } S_2 = j,$$

$$S_3 = Span\{\overrightarrow{w}_{i+j-n}, \cdots, \overrightarrow{w}_n\} \text{ so dim } S_3 = 2n - i - j + 1.$$

Again, by Lemma 1.2.5, $\dim(S_1 \cap S_2 \cap S_3) \geq \dim S_1 + \dim S_2 + \dim S_3 - 2n = 1$. Therefore, there exists $\overrightarrow{x} \in (S_1 \cap S_2 \cap S_3)$, where $\overrightarrow{x} \neq 0$ and $x^T x = 1$. In the same way as above, for $1 \leq i, j \leq n$ and $i + j \geq n + 1$, we obtain

$$\gamma_{i+j-n} \geq x^T(A+B)x = x^T A x + x^T B x \geq \alpha_i + \beta_j. \qquad \square$$

Lemma 1.2.15. *Let A be an $n \times n$ real symmetric matrix with $A > 0$. Then $A = (A_1)^2$ where $A_1 > 0$ and symmetric.*

Proof.

Since A is real symmetric, there exists a real orthogonal matrix U such that $A = UDU^T$ where $D = diag(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\alpha_i \in \sigma(A)$.

Since $A > 0$, each $\alpha_i > 0$. Therefore, $\sqrt{D}$ exists and $\sqrt{D} = (\sqrt{D})^T > 0$. Observe that

$$\begin{aligned} A = UDU^T &= U(\sqrt{D}\sqrt{D})U^T \\ &= U\sqrt{D}U^T U(\sqrt{D})^T U^T \\ &= (U\sqrt{D}U^T)(U\sqrt{D}U^T)^T = A_1 A_1^T \end{aligned}$$

where $A_1 = U\sqrt{D}U^T$. Since $U\sqrt{D}U^T$ is symmetric, so is A_1. Therefore, $A_1 = A_1^T$. This implies $A = (A_1)^2$. Since $A > 0$, $A_1 > 0$ and this completes the proof. □

Theorem 1.2.16 (singular value decomposition). *If $A \in M_{m,n}$ has rank k, then it may be written in the form $A = V\Sigma W^T$ where $V \in M_m$ and $W \in M_n$ are orthogonal. The matrix $\Sigma = [\sigma_{ij}] \in M_{m,n}$ has $\sigma_{ij} = 0$ for al $i \neq j$, and $\sigma_{11} \geq \sigma_{22} \geq \cdots \geq \sigma_{kk} > \sigma_{k+1,k+1} = \cdots = \sigma_{qq} = 0$ where $q = min\{m, n\}$.*

Proof. See [HJ85, pp. 414-415]. □

Lemma 1.2.17. *Let C be an $n \times n$ real symmetric matrix with $C > 0$ and $C = C_1 C_1^T$ where $C_1 \in \boldsymbol{M}_{n,(n+1)}$. Then $\sigma(C_1^T C_1) = \sigma(C_1 C_1^T) \cup \{0\}$.*

Proof.

Let $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ be the set of eigenvalues of C. Since $C > 0$, each $\gamma_i > 0$. By the *singular value decomposition* (Theorem 1.2.16), we may write

$$C_1 = V \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T$$

where $V \in \mathbf{M}_n$ and $W \in \mathbf{M}_{n+1}$ are orthogonal, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0$ are singular values such that $\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\} = \sigma(C_1 C_1^T) = \sigma(C) = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$.

Now calculate:

$$C_1C_1^T = V\begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T \left(V \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T \right)^T$$

$$= V\begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \vdots & 0 & \sigma_n \\ 0 & 0 & \cdots & 0 \end{pmatrix} V^T$$

$$= V\begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_n^2 \end{pmatrix} V^T$$

$$= V\begin{pmatrix} \gamma_1 & 0 & \cdots & 0 \\ 0 & \gamma_2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & \gamma_n \end{pmatrix} V^T \tag{1.1}$$

and

$$C_1^T C_1 = \left(V \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T \right)^T V \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T$$

$$= W \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \vdots & 0 & \sigma_n \\ 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & \cdots & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n & 0 \end{pmatrix} W^T$$

$$= W \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2^2 & 0 & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \sigma_n^2 & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix} W^T$$

$$= W \begin{pmatrix} \gamma_1 & 0 & \cdots & 0 & 0 \\ 0 & \gamma_2 & 0 & \vdots & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & \gamma_n & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix} W^T \tag{1.2}$$

From (1.1) and (1.2), we see that $\sigma(C_1^T C_1) = \sigma(C_1 C_1^T) \cup \{0\}$. □

Lemma 1.2.18. *Let $\alpha_1 \geq \alpha_2$, $\beta_1 \geq \beta_2$, and $\gamma_1 \geq \gamma_2$ be real numbers such that*

$$max\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\} \leq \gamma_1 \leq \alpha_1 + \beta_1, \tag{1.3}$$

$$\alpha_2 + \beta_2 \leq \gamma_2 \leq min\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\}, \tag{1.4}$$

$$\gamma_1 + \gamma_2 = \alpha_1 + \alpha_2 + \beta_1 + \beta_2. \tag{1.5}$$

Then $(\alpha_1 + \beta_2)(\alpha_2 + \beta_1) \geq \gamma_1\gamma_2 \geq (\alpha_1 + \beta_1)(\alpha_2 + \beta_2)$.

Proof.

Let $p = \alpha_1 + \beta_2$ and $q = \alpha_2 + \beta_1$, we need to show that $pq \geq \gamma_1\gamma_2$. By (1.4), we get $\gamma_2 \leq p$ and $\gamma_2 \leq q$. Let $S = \gamma_1 + \gamma_2$, by (1.5) we have

$$p + q = \gamma_1 + \gamma_2 = S \tag{1.6}$$

By (1.3), $\gamma_1 \geq q$ and $\gamma_1 \geq p$. This implies that

$$\gamma_2 \leq p, q \leq \gamma_1 \tag{1.7}$$

By (1.7), we have

$$(\gamma_2 \leq p \leq \gamma_1) + (-\gamma_1 \leq -q \leq -\gamma_2) = (\gamma_2 - \gamma_1 \leq p - q \leq \gamma_1 - \gamma_2)$$

$$(\gamma_2 \leq q \leq \gamma_1) + (-\gamma_1 \leq -p \leq -\gamma_2) = (\gamma_2 - \gamma_1 \leq q - p \leq \gamma_1 - \gamma_2)$$

This means

$$|p - q| \leq |\gamma_1 - \gamma_2| \tag{1.8}$$

This is equivalent to

$$(p - q)^2 \leq (\gamma_1 - \gamma_2)^2 \tag{1.9}$$

Consider

$$pq = \frac{1}{4}(4pq) = \frac{1}{4}[(p+q)^2 - (p-q)^2] = \frac{1}{4}[S^2 - (p-q)^2] \tag{1.10}$$

$$\gamma_1\gamma_2 = \frac{1}{4}(4\gamma_1\gamma_2) = \frac{1}{4}[(\gamma_1 + \gamma_2)^2 - (\gamma_1 - \gamma_2)^2] = \frac{1}{4}[S^2 - (\gamma_1 - \gamma_2)^2] \tag{1.11}$$

Hence, by (1.9), (1.10), and (1.11) we get $pq \geq \gamma_1\gamma_2$.

Similarly, let $p = \alpha_1 + \beta_1$ and $q = \alpha_2 + \beta_2$, we want to show that $pq \leq \gamma_1\gamma_2$. By (1.3), we get $\gamma_1 \leq \alpha_1 + \beta_1$. This forces $\gamma_1 \leq p$. By (1.4), we get $\gamma_2 \leq \alpha_1 + \beta_2 \leq \alpha_1 + \beta_1$. This forces $\gamma_2 \leq p$. Let $S = \gamma_1 + \gamma_2$, by (1.5) we have

$$p + q = \gamma_1 + \gamma_2 = S \tag{1.12}$$

Since $\gamma_1 \leq p$ and $\gamma_2 \leq p$, by (1.12) we obtain $\gamma_2 \geq q$ and $\gamma_1 \geq q$. This implies that

$$q \leq \gamma_2, \gamma_1 \leq p \tag{1.13}$$

By (1.13) and the same argument, we obtain $pq \leq \gamma_1\gamma_2$. □

CHAPTER 2

THE LEADING PRINCIPAL SUBMATRIX

This chapter introduces a problem that involves the leading principal submatrix. An $n \times n$ symmetric matrix will be constructed if the prescribed spectra of this matrix and its leading principal submatrix are given. This problem came from *Sur L'equation 'a L'aide de Laquelle on D'etermine les In'egalit'es S'eculaires des Mouvements des Plan'etes* of A.L. Cauchy in 1841 [Cau41, pp. 174-195]. A proof is presented first. Its sufficient part is taken from *Matrix Analysis* of Roger Horn and Charles Johnson [HJ85, pp. 186-188]. Alternate proofs of the necessary part can be found in [Fis04, pp. 118] and [Hwa04, pp. 157-159]. Then an algorithm according to this proof is obtained. Finally, a program written in Matlab will conclude this chapter.

2.1 The Leading Principal Submatrix Problem

Given $n \geq 2$ and

$$\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{n-1} \geq \alpha_n \tag{2.1}$$

$$\alpha_1' \geq \alpha_2' \geq \dots \geq \alpha_{n-1}' \tag{2.2}$$

Construct a symmetric matrix A such that

$$\sigma(A) = \{\alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n\} \tag{2.3}$$

$$\sigma(A') = \{\alpha_1', \alpha_2', \dots, \alpha_{n-1}'\} \tag{2.4}$$

where $A^{'}$ is the leading principal submatrix of A.

2.2 Solution of the Leading Principal Submatrix Problem

The necessary and sufficient condition is

$$\alpha_1 \geq \alpha_1^{'} \geq \alpha_2 \geq \alpha_2^{'} \geq \ldots \geq \alpha_{n-1} \geq \alpha_{n-1}^{'} \geq \alpha_n.$$

2.2.1 Proof of Necessity

Let

$$A = \begin{pmatrix} A^{'} & y \\ y^T & a \end{pmatrix}$$

be an $n \times n$ symmetric matrix (Definition 1.2.1), where $A^{'} \in M_{n-1}$. Assume that the spectrum of A satisfies (2.1) and (2.3), the spectrum of $A^{'}$ satisfies (2.2) and (2.4). We need to prove $\alpha_1 \geq \alpha_1^{'} \geq \alpha_2 \geq \alpha_2^{'} \geq \ldots \geq \alpha_{n-1} \geq \alpha_{n-1}^{'} \geq \alpha_n$. In other words, we need to show that $\alpha_k \geq \alpha_k^{'} \geq \alpha_{k+1}$ where k = 1, 2, ..., n - 1.

Indeed, using *Cauchy's* interlace (Theorem 1.2.8) with $m = n - 1$, we are done.

2.2.2 Proof of Sufficiency

Let $\{\alpha_i^{'} : i = 1, 2, ..., n-1\}$ and $\{\alpha_i : i = 1, 2, ..., n\}$ be two sequences of real numbers such that $\alpha_1 \geq \alpha_1^{'} \geq \alpha_2 \geq \alpha_2^{'} \geq \ldots \geq \alpha_{n-1} \geq \alpha_{n-1}^{'} \geq \alpha_n$. Let $A^{'} = diag(\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_{n-1}^{'})$. This implies that $\{\alpha_1^{'}, \alpha_2^{'}, \ldots, \alpha_{n-1}^{'}\}$ is the set of eigenvalues of $A^{'}$. If we can prove that there exists a real number a and a vector $y \in \boldsymbol{R}^{n-1}$ such that $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the set of eigenvalues of the symmetric matrix

$$A = \begin{pmatrix} A^{'} & y \\ y^T & a \end{pmatrix} \in \boldsymbol{M}_n \text{ then we are done.}$$

Since $trA = trA' + a$, we must have $a = trA - trA' = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n-1} \alpha_i'$. Therefore, it remains to find $(n-1)$ real numbers y_i so that $p_A(\alpha_k) = 0$ for $k = 1, 2, \ldots, n$. Here $p_A(t) = det(tI - A)$ is the characteristic polynomial of A.

Assume that $t \neq \alpha_i'$ for $i = 1, 2, \cdots, n-1$. Then $(tI - A')$ is invertible and

$$det \begin{bmatrix} I & 0 \\ [(tI - A')^{-1}y]^T & 1 \end{bmatrix} = 1 = det \begin{bmatrix} I & [(tI - A')^{-1}y] \\ 0 & 1 \end{bmatrix};$$

and because A' is diagonal, $y^T(tI - A')^{-1}y = \sum_{i=1}^{n-1} y_i^2 \frac{1}{t-\alpha_i'}$.

Also

$$\begin{aligned} det(tI - A) &= det \begin{bmatrix} tI - A' & -y \\ -y^T & t-a \end{bmatrix} \\ &= det \left\{ \begin{bmatrix} I & 0 \\ [(tI - A')^{-1}y]^T & 1 \end{bmatrix} \begin{bmatrix} tI - A' & -y \\ -y^T & t-a \end{bmatrix} \begin{bmatrix} I & [(tI - A')^{-1}y] \\ 0 & 1 \end{bmatrix} \right\} \\ &= det \begin{bmatrix} tI - A' & 0 \\ 0 & (t-a) - y^T(tI - A')^{-1}y \end{bmatrix} \\ &= [(t-a) - y^T(tI - A')^{-1}y] det(tI - A') \\ &= \left[(t-a) - \sum_{i=1}^{n-1} \left(y_i^2 \frac{1}{t - \alpha_i'} \right) \right] \prod_{i=1}^{n-1} (t - \alpha_i') \end{aligned} \tag{2.5}$$

Define the polynomials f with degree n, and g with degree $n-1$ as follows

$$f(t) = \prod_{i=1}^{n} (t - \alpha_i), \tag{2.6}$$

$$g(t) = \prod_{i=1}^{n-1} (t - \alpha_i') \tag{2.7}$$

By the Euclidean algorithm we must have

$$f(t) = g(t)(t - c) + r(t)$$

where c is a real number and $r(t)$ is a polynomial of degree at most $n-2$. By explicit computation we find that $c = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n-1} \alpha_i' = a$. Furthermore, $f(\alpha_k') = g(\alpha_k')(\alpha_k' - a) + r(\alpha_k') = r(\alpha_k')$ for $k = 1, 2, \ldots, n-1$ because $g(\alpha_k') = 0$.

Case 1. (*all* α_i' *are distinct*) The polynomial $r(t)$ is known at $n-1$ points and can be written explicitly in terms of *Lagrange interpolating polynomial* (Theorem 1.2.9) because the points of interpolation $\alpha_1', \alpha_2', \ldots, \alpha_{n-1}'$ are distinct. Under this assumption, $g(t)$ has only simple roots, and the Lagrange interpolation formula for $r(t)$ is

$$r(t) = \sum_{i=1}^{n-1} \left(f(\alpha_i') \frac{g(t)}{g'(\alpha_i')(t - \alpha_i')} \right) \tag{2.8}$$

Thus,

$$\frac{f(t)}{g(t)} = (t-a) + \frac{r(t)}{g(t)} = (t-a) - \sum_{i=1}^{n-1} \left(\frac{-f(\alpha_i')}{g'(\alpha_i')} \frac{1}{(t - \alpha_i')} \right)$$

Because $f(\alpha_k) = 0$ for all $k = 1, 2, \ldots, n$ we must have

$$(\alpha_k - a) - \sum_{i=1}^{n-1} \left(\frac{-f(\alpha_i')}{g'(\alpha_i')} \frac{1}{(\alpha_k - \alpha_i')} \right) = 0, \qquad k = 1, 2, \ldots, n$$

Notice that if $\alpha_k = \alpha_i'$ for $i = k-1$ or k, then the corresponding term $\frac{1}{t-\alpha_i'}$ has a zero coefficient and there is no singularity at $t = \alpha_k$. If we can set $y_i^2 \equiv \frac{-f(\alpha_i')}{g'(\alpha_i')}$ for $i = 1, 2, \ldots, n-1$, then (2.5) guarantees that $p_A(\alpha_k) = 0$ and we are done. Therefore, we must show that $\frac{f(\alpha_i')}{g'(\alpha_i')} \leq 0$ for $i = 1, 2, \ldots, n-1$, and it is now that the interlacing assumption must be used. Using the definitions of $f(t)$ and $g(t)$ and the interlacing assumption, we find that

$$f(\alpha_i') = (-1)^{n-i} \prod_{j=1}^{n} |\alpha_i' - \alpha_j|$$

$$g'(\alpha_i') = (-1)^{n-i-1} \prod_{j=1, j\neq i}^{n-1} |\alpha_i' - \alpha_j'|$$

and hence $f(\alpha_i')$ and $g'(\alpha_i')$ always have opposite signs. Therefore, $y_i = \sqrt{\frac{-f(\alpha_i')}{g'(\alpha_i')}}$.

Case 2. (*some of the* $\alpha_i^{'}$ *coincide*) If, for example, $\alpha_1^{'} = \alpha_2^{'} = \ldots = \alpha_k^{'} > \alpha_{k+1}^{'} \geq \ldots$ for some $k \geq 2$, then $\alpha_2 = \ldots = \alpha_k = \alpha_1^{'}$. The polynomial $f(t)$ in (2.6) has a factor $(t - \alpha_1)(t - \alpha_1^{'})^{k-1}$; the polynomial $g(t)$ in (2.7) has a factor $(t - \alpha_1^{'})^k$ and k is the exact multiplicity of $\alpha_1^{'}$ as a zero of $g(t)$. Therefore, we may modify $f(t), g(t)$, and $r(t)$ by dividing each by $(t - \alpha_1^{'})^{k-1}$. The modified polynomial $g(t)$ will have $\alpha_1^{'}$ as a simple zero. If we proceed in this way to remove all multiple roots of $g(t)$, the argument can proceed as case 1, and the conclusion is the same. □

2.3 Algorithm for the Leading Principal Submatrix Problem

This algorithm has 8 steps that are based on the previous proof.

2.3.1 Input

The prescribed spectra of an $n \times n$ symmetric matrix A and its leading principal submatrix.

2.3.2 Output

The $n \times n$ symmetric matrix A if there exists a solution.

2.3.3 Algorithm

Step 1 (Get Input). Get the prescribed spectra of an $n \times n$ symmetric matrix A and its leading principal submatrix.

Step 2 (Check the Length). Check whether the length of the prescribed spectrum of the symmetric matrix A is greater than the length of the prescribed spectrum of its leading principal submatrix exactly 1. If it is not, display an error message and stop.

Step 3 (Sort the Prescribed Spectra). Sort all prescribed spectra in a non-increasing order.

Step 4 (Interlacing Verification). Verify whether the prescribed spectra of the symmetric matrix A and its leading principal submatrix satisfy the interlacing property. If it does not, display an error message and stop.

Step 5 (Multiple Eigenvalues). Check whether the prescribed spectrum of the leading principal submatrix has some multiple eigenvalues and remove them in pairs from the described spectra for A' and A. For example, if $\alpha_i' = \alpha_{i+1}'$, remove α_i and α_i'. This prevents dividing by zero when we compute y_i. Let the reduced spectra have m and $m-1$ elements, respectively.

Step 6 (Re-check the New Length). Again, check whether the *new length* of the *distinct* prescribed eigenvalues of the symmetric matrix A is one greater than the *new length* of the *distinct* prescribed eigenvalues of its leading principal submatrix. If it is not, display an error message and stop.

Step 7 (Compute the Value a and the Vector y). Using the reduced set of α_i and α_i', and the associated functions $f(t)$ and $g(t)$, compute the value $a = \sum \alpha_i - \sum \alpha_i'$ and the vector y with the size $[1 \times (m-1)]$, $y_i = \sqrt{\frac{-f(\alpha_i')}{g'(\alpha_i')}}$. For each α_i' discarded, let $y_i = 0$.

Step 8 (Display the Output). Output the $n \times n$ symmetric matrix A where

$$A = \begin{pmatrix} A' & y \\ y^T & a \end{pmatrix} \in M_n \text{ and } A' \text{ is the leading principal submatrix of } A.$$

Remark 2.3.1. For the codes of this algorithm in Matlab, see [Appendix A, pp. 57-64].

2.4 Examples

Example 2.4.1.

Input:

$$\sigma(A) = \{5, 3, 1, -2\}$$
$$\sigma(A') = \{4.7, 2.2, 0\}.$$

Output:

$$A = \begin{pmatrix} 4.7000 & 0 & 0 & 1.0373 \\ 0 & 2.2000 & 0 & 1.4327 \\ 0 & 0 & 0 & 1.7033 \\ 1.0373 & 1.4327 & 1.7033 & 0.1000 \end{pmatrix}$$

Example 2.4.2.

Input:

$$\sigma(A) = \{6, 4, 4, 4, -3\}$$
$$\sigma(A') = \{5, 4, 4, 1\}.$$

Output:

$$A = \begin{pmatrix} 5.0000 & 0 & 0 & 0 & 1.4142 \\ 0 & 4.0000 & 0 & 0 & 0 \\ 0 & 0 & 4.0000 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 3.8730 \\ 1.4142 & 0 & 0 & 3.8730 & 1.0000 \end{pmatrix}$$

CHAPTER 3

THE RANK-1 PERTURBATION

This chapter introduces a problem that involves a *rank* 1 matrix (Lemma 1.2.10). This problem came from *Das asymptotische Verteilungsgestez der Eigenwert linearer partieller Differentialgleichungen (mit einer Anwendung auf der Theorie der Hohlraumstrahlung)* of H. Weyl in 1912 [Wey12, pp. 441-479]. The idea of the proof of sufficiency is taken from R.C. Thompson [Tho76, pp. 69-78]. Another proof can be found in [SA03, pp. 375-378]. Like the previous chapter, a proof is presented first and then an algorithm based on this proof. Finally, the chapter will be concluded by a program written in Matlab.

3.1 The Rank-1 Perturbation Problem

Given $n \geq 2$ and

$$\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{n-1} \geq \alpha_n \tag{3.1}$$

$$\gamma_1 \geq \gamma_2 \geq \dots \geq \gamma_{n-1} \geq \gamma_n \tag{3.2}$$

Construct symmetric matrices A and B, where $rank(B) = 1$ such that

$$\sigma(A) = \{\alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n\} \tag{3.3}$$

$$\sigma(A + B) = \{\gamma_1, \gamma_2, \dots, \gamma_{n-1}, \gamma_n\} \tag{3.4}$$

Without loss of generality, we assume that the *rank* 1 matrix B is *positive semidefinite* (Definition 1.2.12) throughout this chapter. If $B = 0$, then the solution is trivial, $C = A + B = A$. If $B \leq 0$, we consider the negative of this *rank* 1 matrix.

3.2 Solution of the Rank-1 Perturbation Problem

The necessary and sufficient condition is

$$\gamma_1 \geq \alpha_1 \geq \gamma_2 \geq \alpha_2 \geq \ldots \geq \gamma_n \geq \alpha_n.$$

3.2.1 Proof of Necessity

Let A and B be $n \times n$ symmetric matrices where $rank(B) = 1$. Assume that the spectrum of A satisfies (3.1) and the spectrum of $A + B$ satisfies (3.2). We need to show that $\gamma_1 \geq \alpha_1 \geq \gamma_2 \geq \alpha_2 \geq \ldots \geq \gamma_n \geq \alpha_n$. In other words, we need to prove $\gamma_k \geq \alpha_k \geq \gamma_{k+1}$ and $\gamma_n \geq \alpha_n$, where $k = 1, 2, \ldots n-1$.

Let $\sigma(B) = \{\beta_1, \beta_2, \ldots, \beta_{n-1}, \beta_n\}$ such that $\beta_1 \geq \beta_2 \geq \ldots \geq \beta_{n-1} \geq \beta_n$. Since $rank(B) = 1$ and $B \geq 0$, by Lemma 1.2.10 and Lemma 1.2.13, we must have $\beta_1 > \beta_2 = \beta_3 = \ldots = \beta_n = 0$.

By the *Weyl's inequalities* (Theorem 1.2.14), for every pair of integers i, j such that $1 \leq i, j \leq n$, we have

$$\gamma_{i+j-1} \leq \alpha_i + \beta_j, \qquad for \;\; i + j \leq n + 1 \tag{3.5}$$

$$\gamma_{i+j-n} \geq \alpha_i + \beta_j, \qquad for \;\; i + j \geq n + 1 \tag{3.6}$$

When $i = k$ and $j = n$, $i + j = k + n \geq n + 1$ for all $k = 1, 2, \ldots n$, so (3.6) implies $\gamma_k \geq \alpha_k$ because $\beta_n = 0$.

Similarly, when $i = k$ and $j = 2$, $i + j = k + 2 \leq n + 1$ for all $k = 1, 2, \ldots n - 1$, so (3.5) implies $\alpha_k \geq \gamma_{k+1}$ because $\beta_2 = 0$. Hence, $\gamma_k \geq \alpha_k \geq \gamma_{k+1}$ where $k = 1, 2, \ldots n - 1$ and $\gamma_n \geq \alpha_n$.

3.2.2 Proof of Sufficiency

Let $\{\alpha_i : i = 1, 2, ..., n\}$ and $\{\gamma_i : i = 1, 2, ..., n\}$ be two sequences of real numbers such that $\gamma_1 \geq \alpha_1 \geq \gamma_2 \geq \alpha_2 \geq \ldots \geq \gamma_n \geq \alpha_n$. Let $A = diag(\alpha_1, \alpha_2, \ldots, \alpha_n)$, this implies that $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the set of eigenvalues of A. If we can prove that there exists a *rank* 1 matrix B such that $\sigma(A + B) = \{\gamma_1, \gamma_2, \ldots, \gamma_{n-1}, \gamma_n\}$ then we are done.

First of all, if $\alpha_n \leq 0$ we shift all α_i's by adding a number $t > 0$ such that $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_{n-1} \geq \alpha_n > 0$. Add the same t to all γ_i's, so the interlacing property between γ_i's and α_i's is preserved.

Since $\sigma(A) > 0$, we have $A > 0$. Notice that $A = A_1 A_1^T$ where $A_1 = A_1^T = diag(\sqrt{\alpha_1}, \ldots, \sqrt{\alpha_n})$. Since B is a *rank* 1 matrix, by Lemma 1.2.10, we may write $B = xx^T$ where x is a nonzero vector in $\mathbf{R}^n$.

If such x exists then

$$\begin{aligned} A + B &= A + xx^T = A_1 A_1^T + xx^T \\ &= \begin{pmatrix} A_1 & x \end{pmatrix} \begin{pmatrix} A_1^T \\ x^T \end{pmatrix} = \begin{pmatrix} A_1 & x \end{pmatrix} \begin{pmatrix} A_1 & x \end{pmatrix}^T \\ &= C_1 C_1^T \end{aligned}$$

where $C_1 = \begin{pmatrix} A_1 & x \end{pmatrix} \in M_{n,n+1}$, but $C_1 C_1^T \in M_n$ and the eigenvalues of $C_1 C_1^T$ are $\gamma_1 \geq \gamma_2 \geq \ldots \geq \gamma_{n-1} \geq \gamma_n$. By Lemma 1.2.17, we have $C_1^T C_1 \in M_{n+1}$ and $\sigma(C_1^T C_1) = \sigma(C_1 C_1^T) \cup \{0\}$. Furthermore, zero will be the smallest eigenvalue of $C_1^T C_1$ because all γ_i's are positive due to shifting, so the n eigenvalues of A and the $n + 1$ eigenvalues of $C_1^T C_1$ will satisfy the interlacing condition of 2.2 in the previous chapter.

Finally, note that

$$C_1^T C_1 = \begin{pmatrix} A_1 & x \end{pmatrix}^T \begin{pmatrix} A_1 & x \end{pmatrix} = \begin{pmatrix} A_1^T \\ x^T \end{pmatrix} \begin{pmatrix} A_1 & x \end{pmatrix}$$

$$= \begin{pmatrix} A_1^T A_1 & A_1^T x \\ x^T A_1 & x^T x \end{pmatrix}$$

$$= \begin{pmatrix} A & A_1^T x \\ x^T A_1 & x^T x \end{pmatrix}$$

and x must satisfy $x^T x = tr(C_1^T C_1) - tr(A) = tr(C_1 C_1^T) + \{0\} - tr(A) = tr(A + B) - tr(A) = tr(B)$. Thus we have a *leading principal submatrix* problem, which can be solved as in the previous chapter. That is, there exists $y \in \mathbf{R}^n$ so that

$$\begin{pmatrix} A & y \\ y^T & x^T x \end{pmatrix}$$

has eigenvalues $\gamma_1, \cdots, \gamma_n, 0$.

Now we need to show x exists so that $y = A_1^T x$. Indeed, if α_i is not repeated, then $x_i = [(A_1^T)^{-1} y]_i = [(A_1)^{-1} y]_i = \frac{y_i}{\sqrt{\alpha_i}}$. If $\alpha_i = \alpha_{i+1}$, then $x_i = 0$. □

Remark 3.2.1. Based on this proof, we derive the following lemma which is used for the 3×3 matrices in the next chapter.

Lemma 3.2.2 (3×3 *rank* 1 perturbation). *Let* $\beta > 0$ *and*

$$\alpha_1 \geq \alpha_2 \geq \alpha_3$$

$$\gamma_1 \geq \gamma_2 \geq \gamma_3$$

such that $\gamma_1 \geq \alpha_1 \geq \gamma_2 \geq \alpha_2 \geq \gamma_3 \geq \alpha_3$ *and* $\beta + \alpha_1 + \alpha_2 + \alpha_3 = \gamma_1 + \gamma_2 + \gamma_3$. *Then there exist two real symmetric matrices* A *and* B, *where* $B = xx^T$, $0 \neq \overrightarrow{x} \in \boldsymbol{R}^3$

such that

$$\sigma(A) = \{\alpha_1, \alpha_2, \alpha_3\}$$
$$\sigma(A + xx^T) = \{\gamma_1, \gamma_2, \gamma_3\}$$

with $A = diag(\alpha_1, \alpha_2, \alpha_3)$ *and* $B = xx^T$. *If* α_i *is not repeated, then* $x_i = \sqrt{\frac{\prod_{k=1}^{3} |\alpha_i - \gamma_k|}{\prod_{k=1,k\neq i}^{3} |\alpha_i - \alpha_k|}}$. *If* $\alpha_i = \alpha_{i+1}$, *then* $x_i = 0$.

3.3 Algorithm for the Rank-1 Perturbation Problem

This algorithm has 9 steps that are based on the previous proof.

3.3.1 Input

The prescribed spectra of two $n \times n$ symmetric matrices A and C, where $C = A + B$ and $rank(B) = 1$.

3.3.2 Output

The $n \times n$ symmetric matrix A and the *rank* 1 matrix B if there exists a solution.

3.3.3 Algorithm

Step 1 (Get Input). Get the prescribed spectra of two $n \times n$ symmetric matrices A and C, where $C = A + B$ and $rank(B) = 1$.

Step 2 (Check the Length). Check whether the lengths of two prescribed spectra are exactly equal to each other. If it is not, display an error message and stop.

Step 3 (Sort the Prescribed Spectra). Sort all prescribed spectra in a non-increasing order.

Step 4 (Shifting Spectra). Shift the prescribed spectra of A and $A + B$ (with the same number) such that $A > 0$.

Step 5 (*Interlacing* Verification). Verify whether the prescribed spectra of the symmetric matrices A and $A + B$ satisfy the *interlacing* condition.

Step 6 (Multiple Eigenvalues). Check whether the prescribed spectra of the matrices A and C have some multiple eigenvalues.

Step 7 (Re-check the Length). Check whether the lengths of two spectra after removing the multiple eigenvalues are exactly equal to each other. If it is not, display an error message and stop.

Step 8 (Compute the Vector x). Compute the vector $x \in \mathbf{R}^n$.

Step 9 (Display the Output). Output the $n \times n$ symmetric matrices A and B, where $A = diag(\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, \alpha_n)$ and $B = xx^T$.

Remark 3.3.1. For the codes of this algorithm in Matlab, see [Appendix B, pp. 65-75].

3.4 Examples

Example 3.4.1.

Input:

$$\sigma(A) = \{5, 3, 1\}$$

$$\sigma(C) = \{6, 4, 2\}.$$

Output:

$$C = A + B = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0.3750 & 0.5303 & 0.8385 \\ 0.5303 & 0.7500 & 1.1859 \\ 0.8385 & 1.1859 & 1.8750 \end{pmatrix}$$

Example 3.4.2.

Input:

$$\sigma(A) = \{3, 2, 2, -5\}$$

$$\sigma(C) = \{2.5, 2, 2, -6.5\}.$$

Output:

$$C = A + B = \begin{pmatrix} 2.9167 & 0 & 0 & 0.8122 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0.8122 & 0 & 0 & -4.9167 \end{pmatrix} + \begin{pmatrix} -0.4167 & 0 & 0 & -0.8122 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.8122 & 0 & 0 & -1.5833 \end{pmatrix}$$

CHAPTER 4

THE SYMMETRIC SUM

This chapter introduces a problem that involves the symmetric sum. In 1912, Hermann Weyl raised the question: what are the possible eigenvalues of a sum of two Hermitian matrices whose eigenvalues are given? This question was later formalized as the Horn's conjecture on eigenvalues of sums of Hermitian matrices in 1962 [Hor62, pp. 225-241]. This conjecture was recently proved by Allen Knutson and Terence Tao who were awarded the Levi L. Conant Prize in 2005 for their article "Honeycombs and Sums of Hermitian Matrices" [KT01, pp. 175-186]. Knutson and Tao introduced the concept of "Honeycombs" and used them to prove this conjecture. Since the concept of "Honeycombs" is a high level tool in mathematics, this chapter is looking for an elementary proof to this problem. However, it seems very difficult to find an elementary proof of the general case $n \times n$. Therefore, this chapter only considers two cases: 2×2 and 3×3. The idea of the proof is taken from Wasin So [So06]. For the general n, the chapter introduces a special case: construct two $n \times n$ symmetric matrices from their prescribed spectra so that their sum has an arbitrary γ_k as its k^{th} eigenvalue. The sufficient proof of the case $n \times n$ is taken from R.C. Thompson [Tho91]. Like the two previous chapters, the proof-algorithm-program-example format will be used.

4.1 The Symmetric Sum Problem: 2×2 case

Given

$$\alpha_1 \geq \alpha_2 \tag{4.1}$$

$$\beta_1 \geq \beta_2 \tag{4.2}$$

$$\gamma_1 \geq \gamma_2 \tag{4.3}$$

Construct symmetric matrices A and B such that

$$\sigma(A) = \{\alpha_1, \alpha_2\} \tag{4.4}$$

$$\sigma(B) = \{\beta_1, \beta_2\} \tag{4.5}$$

$$\sigma(C) = \{\gamma_1, \gamma_2\} \tag{4.6}$$

where $C = A + B$.

4.2 Solution of the Symmetric Sum Problem: 2×2 case

The necessary and sufficient conditions are

$$max\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\} \leq \gamma_1 \leq \alpha_1 + \beta_1, \tag{4.7}$$

$$\alpha_2 + \beta_2 \leq \gamma_2 \leq min\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\}, \tag{4.8}$$

$$\gamma_1 + \gamma_2 = \alpha_1 + \alpha_2 + \beta_1 + \beta_2. \tag{4.9}$$

4.2.1 Proof of Necessity

Let A and B be 2×2 symmetric matrices. Assume that the spectra of A, B, and $A+B$ satisfy (4.1)-(4.6). We need to show that (4.7)-(4.9) must be held. Indeed, we obtain (4.9) because $trace(A + B) = trace(A) + trace(B)$, and (4.7), (4.8) hold by *Weyl's* inequalities (Theorem 1.2.14).

4.2.2 Proof of Sufficiency

Let $\{\alpha_i : i = 1,2\}$, $\{\beta_i : i = 1,2\}$, and $\{\gamma_i : i = 1,2\}$ be three sets of real numbers that satisfy (4.1)-(4.3) and (4.7)-(4.9). Let $A = diag(\alpha_1, \alpha_2)$. This implies that $\{\alpha_1, \alpha_2\}$ is the set of eigenvalues of A. If we can prove that there exist b_i's such that

$$\sigma(B) = \sigma\left(\begin{bmatrix} b_1 & b_3 \\ b_3 & b_2 \end{bmatrix}\right) = \{\beta_1, \beta_2\}$$

and

$$\sigma(C) = \sigma(A+B) = \sigma\left(\begin{bmatrix} \alpha_1 + b_1 & b_3 \\ b_3 & \alpha_2 + b_2 \end{bmatrix}\right) = \{\gamma_1, \gamma_2\}$$

then we are done.

We consider 2 possible cases:

Case 1. (*one of the three sets is constant*) This implies that either $\alpha_1 = \alpha_2$, $\beta_1 = \beta_2$, or $\gamma_1 = \gamma_2$. Without loss of generality, assume that $\beta_1 = \beta_2 = \beta$ since if $\alpha_1 = \alpha_2$, we interchange the roles of A and B. Similarly, if $\gamma_1 = \gamma_2$, we interchange the roles of C and B by letting $A' = A$, $B' = -C$, and $C' = -B$. This yields $A' + B' = C'$.

Now, we claim that it is a trivial solution with $b_3 = 0$ and $b_1 = b_2 = \beta$. Indeed, from (4.7) and (4.8), we have $\gamma_i = \alpha_i + \beta$, for $i = 1, 2$. Hence, the solution will be

$$\begin{pmatrix} \alpha_1 & 0 \\ 0 & \alpha_2 \end{pmatrix} + \begin{pmatrix} \beta & 0 \\ 0 & \beta \end{pmatrix} = \begin{pmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{pmatrix}.$$

Case 2. (*all three sets have distinct elements*) This implies that $\alpha_1 > \alpha_2$, $\beta_1 > \beta_2$, and $\gamma_1 > \gamma_2$.

The characteristic polynomial $p_B(t)$ of B is computed as

$$det(tI - B) = det\begin{bmatrix} t - b_1 & -b_3 \\ -b_3 & t - b_2 \end{bmatrix} = (t - \beta_1)(t - \beta_2).$$

Compare the coefficients of t, yields

$$b_1 + b_2 = \beta_1 + \beta_2 \tag{4.10}$$

$$b_1 b_2 - b_3^2 = \beta_1 \beta_2 \tag{4.11}$$

Similarly, the characteristic polynomial $p_C(t)$ of $C = A + B$ is computed as

$$det(tI - C) = det \begin{bmatrix} t - (\alpha_1 + b_1) & -b_3 \\ -b_3 & t - (\alpha_1 + b_2) \end{bmatrix} = (t - \gamma_1)(t - \gamma_2).$$

Compare the coefficients of t, yields

$$\alpha_1 + \alpha_2 + b_1 + b_2 = \gamma_1 + \gamma_2$$

$$\alpha_1 \alpha_2 + \alpha_2 b_1 + \alpha_1 b_2 + b_1 b_2 - b_3^2 = \gamma_1 \gamma_2 \tag{4.12}$$

By (4.11) and (4.12), we get

$$\alpha_2 b_1 + \alpha_1 b_2 = \gamma_1 \gamma_2 - \beta_1 \beta_2 - \alpha_1 \alpha_2 \tag{4.13}$$

By (4.10) and (4.13), we obtain

$$b_1 = \frac{-(\gamma_1 \gamma_2 - \beta_1 \beta_2 - \alpha_1 \alpha_2) + \alpha_1(\beta_1 + \beta_2)}{\alpha_1 - \alpha_2} \tag{4.14}$$

$$b_2 = \frac{(\gamma_1 \gamma_2 - \beta_1 \beta_2 - \alpha_1 \alpha_2) - \alpha_2(\beta_1 + \beta_2)}{\alpha_1 - \alpha_2}. \tag{4.15}$$

Again, by (4.11) and straight computation, we have

$$\begin{aligned} b_3^2 &= b_1 b_2 - \beta_1 \beta_2 \\ &= \frac{[\gamma_1 \gamma_2 - (\alpha_1 + \beta_1)(\alpha_2 + \beta_2)][(\alpha_1 + \beta_2)(\alpha_2 + \beta_1) - \gamma_1 \gamma_2]}{(\alpha_1 - \alpha_2)^2}. \end{aligned} \tag{4.16}$$

By Lemma 1.2.18, we obtain $b_3^2 \geq 0$ and therefore a real matrix does exist so that the eigenvalues of $A + B$ are $\gamma_1 \geq \gamma_2$. □

Remark 4.2.1. Another approach, [Bha01, pp. 292-293], to this problem is to take $A = diag(\alpha_1, \alpha_2)$ and $B = U\ diag(\beta_1, \beta_2)\ U^T$,

$$\text{where } U = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \text{ with } \theta \in \mathbf{R}.$$

Then solve for $\cos\theta$ and $\sin\theta$ in terms of α's, β's, and γ's.

4.3 Algorithm for the Symmetric Sum Problem: 2×2 case

This algorithm has 6 steps that are based on the previous proof.

4.3.1 Input

The prescribed spectra of 2×2 symmetric matrices A, B, and their sum C.

4.3.2 Output

The 2×2 symmetric matrices A and B if there exists a solution.

4.3.3 Algorithm

Step 1 (Get Input). Get the prescribed spectra of 2×2 symmetric matrices A, B, and their sum C.

Step 2 (Check the Length). Check whether the lengths of the prescribed eigenvalues of the symmetric matrices A, B, and C are exactly equal to 2. If one of them is not, display an error message and stop.

Step 3 (Sort the Prescribed Spectra). Sort all prescribed spectra in a non-increasing order.

Step 4 (Trace and *Weyl*'s Inequalities Verification). Verify whether the prescribed spectra of the symmetric matrices A, B, and C satisfy the trace property and *Weyl*'s inequalities. If it does not, display an error message and stop.

Step 5 (Compute the values b_i). Compute the values b_i of the matrix B using (4.14)-(4.16).

Step 6 (Display the Output). Output the 2×2 symmetric matrices A and B, where $A = diag(\alpha_1, \alpha_2)$ and $B = [b_{ij}]$.

Remark 4.3.1. For the codes of this algorithm in Matlab, see [Appendix C.1, pp. 76-81].

4.4 Examples

Example 4.4.1.

Input:

$$\sigma(A) = \{1, -1\}$$
$$\sigma(B) = \{3, 2\}$$
$$\sigma(C) = \{3.7, 1.3\}.$$

Output:

$$C = A + B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} + \begin{pmatrix} 2.5950 & 0.4909 \\ 0.4909 & 2.4050 \end{pmatrix}$$

Example 4.4.2.

Input:

$$\sigma(A) = \{12.4, -15.7\}$$

$$\sigma(B) = \{6, 3.3\}$$

$$\sigma(C) = \{17.5, -11.5\}.$$

Output:

$$C = A + B = \begin{pmatrix} 12.4000 & 0 \\ 0 & -15.7000 \end{pmatrix} + \begin{pmatrix} 5.0423 & 1.2917 \\ 1.2917 & 4.2577 \end{pmatrix}$$

4.5 The Symmetric Sum Problem: 3×3 case

Given

$$\alpha_1 \geq \alpha_2 \geq \alpha_3 \tag{4.17}$$

$$\beta_1 \geq \beta_2 \geq \beta_3 \tag{4.18}$$

$$\gamma_1 \geq \gamma_2 \geq \gamma_3 \tag{4.19}$$

Construct symmetric matrices A and B such that

$$\sigma(A) = \{\alpha_1, \alpha_2, \alpha_3\} \tag{4.20}$$

$$\sigma(B) = \{\beta_1, \beta_2, \beta_3\} \tag{4.21}$$

$$\sigma(C) = \{\gamma_1, \gamma_2, \gamma_3\} \tag{4.22}$$

where $C = A + B$.

4.6 Solution of the Symmetric Sum Problem: 3×3 case

The necessary and sufficient conditions are

$$max\{\alpha_1 + \beta_3, \alpha_3 + \beta_1, \alpha_2 + \beta_2\} \leq \gamma_1 \leq \alpha_1 + \beta_1, \tag{4.23}$$

$$max\{\alpha_2 + \beta_3, \alpha_3 + \beta_2\} \leq \gamma_2 \leq min\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\}, \tag{4.24}$$

$$\alpha_3 + \beta_3 \leq \gamma_3 \leq min\{\alpha_1 + \beta_3, \alpha_3 + \beta_1, \alpha_2 + \beta_2\}, \tag{4.25}$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \beta_1 + \beta_2 + \beta_3 = \gamma_1 + \gamma_2 + \gamma_3. \tag{4.26}$$

4.6.1 Proof of Necessity

Let A and B be 3×3 symmetric matrices. Assume that the spectra of A, B, and $C = A + B$ satisfy (4.17)-(4.22). We need to show that (4.23)-(4.26) must be held. As the 2×2 case, we obtain (4.26) due to the *trace* property and (4.23)-(4.25) due to *Weyl's* inequalities.

4.6.2 Proof of Sufficiency

Let $\{\alpha_1, \alpha_2, \alpha_3\}$, $\{\beta_1, \beta_2, \beta_3\}$, and $\{\gamma_1, \gamma_2, \gamma_3\}$ be three sets of real numbers such that (4.17)-(4.19) and (4.23)-(4.26) are satisfied.

Let $A = diag(\alpha_1, \alpha_2, \alpha_3)$, so $\{\alpha_1, \alpha_2, \alpha_3\}$ is the set of eigenvalues of A. If we can prove that there exist b_i's such that

$$\sigma(B) = \sigma\left(\begin{bmatrix} b_1 & b_4 & b_6 \\ b_4 & b_2 & b_5 \\ b_6 & b_5 & b_3 \end{bmatrix}\right) = \{\beta_1, \beta_2\, \beta_3\}$$

and

$$\sigma(C) = \sigma(A+B) = \sigma\left(\begin{bmatrix} \alpha_1 + b_1 & b_4 & b_6 \\ b_4 & \alpha_2 + b_2 & b_5 \\ b_6 & b_5 & \alpha_3 + b_3 \end{bmatrix}\right) = \{\gamma_1, \gamma_2\, \gamma_3\}$$

then we are done.

We consider 3 possible cases:

Case 1. (*one of the three sets is constant*) This implies that either $\alpha_1 = \alpha_2 = \alpha_3$, $\beta_1 = \beta_2 = \beta_3$, or $\gamma_1 = \gamma_2 = \gamma_3$. Without loss of generality, assume that $\beta_1 = \beta_2 = \beta_3 = \beta$ since if $\alpha_1 = \alpha_2 = \alpha_3$, we interchange the roles of A and B. Similarly, if $\gamma_1 = \gamma_2 = \gamma_3$, we interchange the roles of C and B by letting $A' = A$, $B' = -C$, and $C' = -B$. This yields $A' + B' = C'$.

Now, we claim that it is a trivial solution with $b_4 = b_5 = b_6 = 0$ and $b_1 = b_2 = b_3 = \beta$. Indeed, from (4.23)-(4.25), we have

$$max\{\alpha_1 + \beta, \alpha_3 + \beta, \alpha_2 + \beta\} \leq \gamma_1 \leq \alpha_1 + \beta$$
$$max\{\alpha_2 + \beta, \alpha_3 + \beta\} \leq \gamma_2 \leq min\{\alpha_1 + \beta, \alpha_2 + \beta\}$$
$$\alpha_3 + \beta \leq \gamma_3 \leq min\{\alpha_1 + \beta, \alpha_3 + \beta, \alpha_2 + \beta\}$$

Therefore,

$$\alpha_1 + \beta \leq \gamma_1 \leq \alpha_1 + \beta$$
$$\alpha_2 + \beta \leq \gamma_2 \leq \alpha_2 + \beta$$
$$\alpha_3 + \beta \leq \gamma_3 \leq \alpha_3 + \beta$$

This forces

$$\gamma_1 = \alpha_1 + \beta$$
$$\gamma_2 = \alpha_2 + \beta$$
$$\gamma_3 = \alpha_3 + \beta$$

Hence, the solution will be trivial

$$A = diag(\alpha_1, \alpha_2, \alpha_3), \ B = diag(\beta_1, \beta_2, \beta_3).$$

Case 2. (*one of the three sets has two distinct elements*) This implies that either

$$\alpha_1 > \alpha_2 = \alpha_3 \qquad or \qquad \alpha_1 = \alpha_2 > \alpha_3,$$

$$\beta_1 > \beta_2 = \beta_3 \qquad or \qquad \beta_1 = \beta_2 > \beta_3,$$

$$\gamma_1 > \gamma_2 = \gamma_3 \qquad or \qquad \gamma_1 = \gamma_2 > \gamma_3.$$

By interchanging the roles of A, B, and C, without loss of generality, we can assume there are only two distinct β_i. Furthermore, assume that $\beta_1 > \beta_2 = \beta_3 = \beta$ since if $\beta_1 = \beta_2 > \beta_3$ and we let $B' = -B$ then $\sigma(B') = \{-\beta_3 > -\beta_2 = -\beta_1\} = \{\beta_1' > \beta_2' = \beta_3'\}$.

Since $C = A + B = (A + \beta I) + (B - \beta I)$, let $C' = C$, $A' = A + \beta I$, and $B' = B - \beta I$. Therefore, $A' + B' = C'$ where

$$\begin{aligned}
\sigma(A') &= \{\alpha_1', \alpha_2', \alpha_3'\} = \{\alpha_1 + \beta, \alpha_2 + \beta, \alpha_3 + \beta\},\\
\sigma(B') &= \{\beta_1', \beta_2', \beta_3'\} = \{\beta_1 - \beta, 0, 0\},\\
\sigma(C') &= \{\gamma_1', \gamma_2', \gamma_3'\} = \{\gamma_1, \gamma_2, \gamma_3\}.
\end{aligned}$$

By direct checking, we see that (4.23)-(4.26) are equivalent to

$$\begin{aligned}
max\{\alpha_1', \alpha_3' + \beta_1', \alpha_2'\} &\leq \gamma_1' \leq \alpha_1' + \beta_1'\\
max\{\alpha_2', \alpha_3'\} &\leq \gamma_2' \leq min\{\alpha_1', \alpha_2' + \beta_1'\}\\
\alpha_3' &\leq \gamma_3' \leq min\{\alpha_1', \alpha_3' + \beta_1', \alpha_2'\}\\
\alpha_1' + \alpha_2' + \alpha_3' + \beta_1' &= \gamma_1' + \gamma_2' + \gamma_3'
\end{aligned}$$

This means

$$\gamma_1' \geq \alpha_1' \geq \gamma_2' \geq \alpha_2' \geq \gamma_3' \geq \alpha_3'.$$

Since $\beta_1' = \beta_1 - \beta = \beta_1 - \beta_2 > 0$, by Lemma 3.2.2, there exist two symmetric matrices

A' and B', and $0 \neq \vec{x} \in \mathbf{R}^3$ such that

$$\sigma(A') = \{\alpha_1', \alpha_2', \alpha_3'\}$$
$$\sigma(B') = \{\beta_1', 0, 0\}$$
$$\sigma(A' + B') = \{\gamma_1', \gamma_2', \gamma_3'\}$$

$$\text{with } A' = diag(\alpha_1', \alpha_2', \alpha_3') \text{ and } B' = xx^T,$$
$$\text{where } x_i = \sqrt{\frac{\prod_{k=1}^{3} |\alpha_i' - \gamma_k'|}{\prod_{k=1,k\neq i}^{3} |\alpha_i' - \alpha_k'|}} = \sqrt{\frac{\prod_{k=1}^{3} |\alpha_i + \beta - \gamma_k|}{\prod_{k=1,k\neq i}^{3} |\alpha_i - \alpha_k|}}.$$

Case 3. (*all three sets have three distinct elements*) This implies that

$$\alpha_1 > \alpha_2 > \alpha_3,\ \beta_1 > \beta_2 > \beta_3,\ and\, \gamma_1 > \gamma_2 > \gamma_3.$$

Observe that if any of $\gamma_k = \alpha_i + \beta_j$ for $i, j, k = 1, 2, 3$, then it reduces to the 2×2 case. For example, suppose that $\gamma_2 = \alpha_2 + \beta_1$, then the solution will be

$$\left(\begin{array}{c|c} \alpha_2 & 0 \\ \hline 0 & A' \end{array}\right) + \left(\begin{array}{c|c} \beta_1 & 0 \\ \hline 0 & B' \end{array}\right) = \left(\begin{array}{c|c} \gamma_2 & 0 \\ \hline 0 & C' \end{array}\right)$$

where $\sigma(A') = \{\alpha_1, \alpha_3\}$, $\sigma(B') = \{\beta_2, \beta_3\}$, and $\sigma(C') = \{\gamma_1, \gamma_3\}$.

Thus, assume that $\gamma_k \neq \alpha_i + \beta_j$ for $i, j, k = 1, 2, 3$. Then the necessary and sufficient conditions become

$$max\{\alpha_1 + \beta_3, \alpha_3 + \beta_1, \alpha_2 + \beta_2\} < \gamma_1 < \alpha_1 + \beta_1$$
$$max\{\alpha_2 + \beta_3, \alpha_3 + \beta_2\} < \gamma_2 < min\{\alpha_1 + \beta_2, \alpha_2 + \beta_1\}$$
$$\alpha_3 + \beta_3 < \gamma_3 < min\{\alpha_1 + \beta_3, \alpha_3 + \beta_1, \alpha_2 + \beta_2\}$$
$$\alpha_1 + \alpha_2 + \alpha_3 + \beta_1 + \beta_2 + \beta_3 = \gamma_1 + \gamma_2 + \gamma_3.$$

Let $\sum x_i x_j = x_1x_2 + x_1x_3 + x_2x_3$ and $\prod(x_i + y_i) = (x_1 + y_1)(x_2 + y_2)(x_3 + y_3)$.

The characteristic polynomial $p_B(t)$ of B is computed as

$$det(tI - B) = det\begin{bmatrix} t-b_1 & -b_4 & -b_6 \\ -b_4 & t-b_2 & -b_5 \\ -b_6 & -b_5 & t-b_3 \end{bmatrix} = (t-\beta_1)(t-\beta_2)(t-\beta_3).$$

Compare the coefficients of t, yields

$$b_1 + b_2 + b_3 = \beta_1 + \beta_2 + \beta_3 \tag{4.27}$$

$$\sum b_i b_j - b_4^2 - b_5^2 - b_6^2 = \sum \beta_i \beta_j \tag{4.28}$$

$$b_1 b_2 b_3 - b_1 b_5^2 - b_2 b_6^2 - b_3 b_4^2 + 2 b_4 b_5 b_6 = \beta_1 \beta_2 \beta_3. \tag{4.29}$$

Similarly, the characteristic polynomial $p_C(t)$ of $C = A + B$ is computed as

$$\begin{aligned} det(tI - C) &= det\begin{bmatrix} t-(\alpha_1+b_1) & -b_4 & -b_6 \\ -b_4 & t-(\alpha_2+b_2) & -b_5 \\ -b_6 & -b_5 & t-(\alpha_3+b_3) \end{bmatrix} \\ &= (t-\gamma_1)(t-\gamma_2)(t-\gamma_3). \end{aligned}$$

Compare the coefficients of t, yields

$$\alpha_1 + \alpha_2 + \alpha_3 + b_1 + b_2 + b_3 = \gamma_1 + \gamma_2 + \gamma_3$$

$$\begin{aligned} &(b_1+\alpha_1)(b_2+\alpha_2) + (b_2+\alpha_2)(b_3+\alpha_3) \\ &\quad +(b_3+\alpha_3)(b_1+\alpha_1) - b_4^2 - b_5^2 - b_6^2 = \sum \gamma_i \gamma_j \end{aligned} \tag{4.30}$$

$$\begin{aligned} &(b_1+\alpha_1)(b_2+\alpha_2)(b_3+\alpha_3) - (b_1+\alpha_1)b_5^2 \\ &\quad -(b_2+\alpha_2)b_6^2 - (b_3+\alpha_3)b_4^2 + 2b_4b_5b_6 = \gamma_1\gamma_2\gamma_3. \end{aligned} \tag{4.31}$$

By (4.28) and (4.30), we get

$$(\alpha_2+\alpha_3)b_1 + (\alpha_1+\alpha_3)b_2 + (\alpha_1+\alpha_2)b_3 = \sum \gamma_i\gamma_j - \sum \beta_i\beta_j - \sum \alpha_i\alpha_j \tag{4.32}$$

From (4.27) and (4.32), we need b_1, b_2, b_3 which satisfy the following linear system:

$$\begin{pmatrix} 1 & 1 & 1 \\ \alpha_2+\alpha_3 & \alpha_1+\alpha_3 & \alpha_1+\alpha_2 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} \beta_1+\beta_2+\beta_3 \\ \sum \gamma_i\gamma_j - \sum \beta_i\beta_j - \sum \alpha_i\alpha_j \end{pmatrix} \quad (4.33)$$

The identities (4.29) and (4.31) imply

$$\alpha_3 b_4^2 + \alpha_1 b_5^2 + \alpha_2 b_6^2 = \prod (b_i + \alpha_i) - b_1b_2b_3 + \beta_1\beta_2\beta_3 - \gamma_1\gamma_2\gamma_3 \quad (4.34)$$

From (4.28) and (4.34), we need b_4, b_5, b_6 which satisfy the following linear system:

$$\begin{pmatrix} 1 & 1 & 1 \\ \alpha_3 & \alpha_1 & \alpha_2 \end{pmatrix} \begin{pmatrix} b_4^2 \\ b_5^2 \\ b_6^2 \end{pmatrix} = \begin{pmatrix} \sum b_ib_j - \sum \beta_i\beta_j \\ \prod(b_i+\alpha_i) - b_1b_2b_3 + \beta_1\beta_2\beta_3 - \gamma_1\gamma_2\gamma_3 \end{pmatrix} \quad (4.35)$$

Observe that the linear systems (4.33) and (4.35) are always consistent, with b_1, b_2 in term of b_3 and b_4^2, b_5^2 in term of b_6^2, since all α_i are distinct:

$$b_1 = \frac{[k - (\alpha_1+\alpha_3)\sum \beta_i - (\alpha_2-\alpha_3)b_3]}{(\alpha_2-\alpha_1)} \quad (4.36)$$

$$b_2 = \frac{-[k - (\alpha_2+\alpha_3)\sum \beta_i - (\alpha_1-\alpha_3)b_3]}{(\alpha_2-\alpha_1)} \quad (4.37)$$

$$b_4^2 = \frac{-[q - \alpha_1 p + (\alpha_1-\alpha_2)b_6^2]}{(\alpha_1-\alpha_3)} \quad (4.38)$$

$$b_5^2 = \frac{[q - \alpha_3 p + (\alpha_3-\alpha_2)b_6^2]}{(\alpha_1-\alpha_3)} \quad (4.39)$$

where

$$k = \sum \gamma_i\gamma_j - \sum \beta_i\beta_j - \sum \alpha_i\alpha_j$$
$$q = \prod(b_i+\alpha_i) - b_1b_2b_3 + \beta_1\beta_2\beta_3 - \gamma_1\gamma_2\gamma_3$$
$$p = \sum b_ib_j - \sum \beta_i\beta_j.$$

From (4.36)-(4.39), we see that b_3 and b_6^2 are free variables. With the help of Matlab, we may find a very close approximate solution. □

Remark 4.6.1. In order to be sure there are real values of all b_i, we have to know that there is a real value for b_3 which allows b_6^2 to be chosen positive, but small enough that b_4^2 and b_5^2 are positive (see next section). We have not succeeded in proving this with algebra only, although it must be true because it has been proved with powerful analytic methods that such A and B do exist [KT01].

4.7 Experimental Results of the Symmetric Sum Problem: 3×3 case

Since b_3 and b_6^2 are free, there exists a question: How to choose them? Before answering this question, let's try to simplify (4.36)-(4.39). Let

$$\alpha_i' = \alpha_i - \alpha_3$$
$$\beta_i' = \beta_i - \beta_3$$
$$\gamma_i' = \gamma_i - \alpha_3 - \beta_3.$$

If all α_i, all β_i, and all γ_i respectively are distinct, and if no $\alpha_i + \beta_j = \gamma_k$, then

$$\alpha_1' > \alpha_2' > \alpha_3' = 0,\ \beta_1' > \beta_2' > \beta_3' = 0,\ and\, \gamma_1' > \gamma_2' > \gamma_3' > 0.$$

The necessary and sufficient conditions become

$$max\{\alpha_1', \beta_1', \alpha_2' + \beta_2'\} < \gamma_1' < \alpha_1' + \beta_1' \tag{4.40}$$
$$max\{\alpha_2', \beta_2'\} < \gamma_2' < min\{\alpha_1' + \beta_2', \alpha_2' + \beta_1'\} \tag{4.41}$$
$$0 < \gamma_3' < min\{\alpha_1', \beta_1', \alpha_2' + \beta_2'\} \tag{4.42}$$
$$\alpha_1' + \alpha_2' + \beta_1' + \beta_2' = \gamma_1' + \gamma_2' + \gamma_3'. \tag{4.43}$$

and

$$b_1 = \frac{[k - \alpha_1'(\beta_1' + \beta_2') - \alpha_2' b_3]}{(\alpha_2' - \alpha_1')}$$

$$= \frac{[\alpha_1'(\beta_1' + \beta_2') + \alpha_2' b_3 - k]}{(\alpha_1' - \alpha_2')} \quad (4.44)$$

$$b_2 = \frac{-[k - \alpha_2'(\beta_1' + \beta_2') - \alpha_1' b_3]}{(\alpha_2' - \alpha_1')}$$

$$= \frac{[k - \alpha_2'(\beta_1' + \beta_2') - \alpha_1' b_3]}{(\alpha_1' - \alpha_2')} \quad (4.45)$$

$$b_4^2 = \frac{-[q - \alpha_1' p + (\alpha_1' - \alpha_2') b_6^2]}{\alpha_1'}$$

$$= \frac{[\alpha_1' p - (\alpha_1' - \alpha_2') b_6^2 - q]}{\alpha_1'} \quad (4.46)$$

$$b_5^2 = \frac{[q - \alpha_2' b_6^2]}{\alpha_1'} \quad (4.47)$$

where, because $\beta_3' = 0$:

$$k = \sum \gamma_i' \gamma_j' - \beta_1' \beta_2' - \alpha_1' \alpha_2' \quad (4.48)$$

$$q = (b_1 + \alpha_1')(b_2 + \alpha_2') b_3 - b_1 b_2 b_3 - \gamma_1 \gamma_2 \gamma_3 \quad (4.49)$$

$$p = \sum b_i b_j - \beta_1' \beta_2'. \quad (4.50)$$

Notice k is determined, and b_1 and b_2 can be calculated for any choice of b_3. Then p and q can be calculated.

Since $\alpha_1' > 0$, by (4.46) $b_4^2 > 0$ if and only if $[\alpha_1' p - (\alpha_1' - \alpha_2') b_6^2 - q] > 0$, if and only if

$$\frac{\alpha_1' p - q}{(\alpha_1' - \alpha_2')} > b_6^2.$$

Similarly, by (4.47) $b_5^2 > 0$ if and only if

$$\frac{q}{\alpha_2'} > b_6^2.$$

This implies

$$0 < b_6^2 < min\{\frac{q}{\alpha_2'}, \frac{\alpha_1' p - q}{(\alpha_1' - \alpha_2')}\}. \tag{4.51}$$

Therefore, if b_3 can be chosen so that q and $\alpha_1' p - q$ are positive, and we then choose $b_6^2 \in (0, min\{\frac{q}{\alpha_2'}, \frac{\alpha_1' p - q}{(\alpha_1' - \alpha_2')}\})$, then $b_4^2 > 0$ and $b_5^2 > 0$. So there will be real solutions for all b_i if such b_3 exists.

Define

$$f = \alpha_1' p - q$$

From (4.44)-(4.50), we can express q and f as functions of b_3:

$$\begin{aligned}
q &= -(\alpha_1' + \alpha_2')b_3^2 + (\alpha_1'\alpha_2' + k)b_3 - \gamma_1'\gamma_2'\gamma_3' \\
f &= \frac{1}{(\alpha_1' - \alpha_2')^2}\{(\alpha_2')^2(\alpha_2' - 2\alpha_1')b_3^2 \\
&+ (\alpha_1'[(\alpha_1' + \alpha_2')k - (\beta_1' + \beta_2')(\alpha_1'^2 + \alpha_2'^2)] + (\alpha_1' - \alpha_2')^2[\alpha_1'(\beta_1' + \beta_2') - k - \alpha_2'\alpha_2'])b_3 \\
&+ \alpha_1'[\alpha_1'(\beta_1' + \beta_2') - k][k - \alpha_2'(\beta_1' + \beta_2')] + (\alpha_1' - \alpha_2')^2(\gamma_1\gamma_2\gamma_3 - \alpha_1'\beta_1'\beta_2')\}.
\end{aligned}$$

Observe that q and f are quadratic functions in b_3 with the coefficients of b_3^2 negative. Therefore, the graphs of q and f are concave down. If the graph of q or f is below the x-axis, then there is no b_3 so that $b_6^2 \geq 0$. Hence, no real solution exists.

Thus, a solution *may* exist if the *vertices* of both graphs q and f are above the x-axis. If q and f satisfy this condition, then we have two possible cases:

Case 1. (their graphs meet each other) Let v be the intersection point of these graphs. Then v can be above, on, or below the x-axis. If v is on or below, there is no solution. That is, even though both graphs q and f are positive for some values of b_3, since v is on or below the x-axis, this means there are some b_3's so that either $q > 0$ or $f > 0$, but there is no value of b_3 such that $q > 0$ and $f > 0$.

If v is above the x-axis, this means there is at least one value of b_3 such that $q > 0$ and $f > 0$. In this case, a solution exists. Since q is a function of b_3, solve q, we will have an interval $[l_1, r_1]$ in which $q \geq 0$. Similarly, since f is a function of b_3, solve f, we will have an interval $[l_2, r_2]$ in which $f \geq 0$. Let $[l, r] = [l_1, r_1] \bigcap [l_2, r_2]$. Notice that the point $v \in [l, r]$.

Case 2. (their graphs do not meet each other) In this case, one graph must be below the other since both graphs are concave down and have no intersection. By our assumption, the *vertices* of both graphs q and f are above the x-axis. This means there exist two intervals, one is contained in the other, such that q and f are positive. In this case, a solution exists. Let $[l, r]$ be the intersection of these intervals. In other words, let $[l, r]$ be the smaller one.

To obtain the closest solution, we calculate the zeros of q and f, then calculate l and r. Then start b_3 at l and compute k, p, q, b_1, and b_2. Now, we compute b_4^2 and b_5^2. Since $b_6^2 \in (0, min\{\frac{q}{\alpha_2'}, \frac{\alpha_1' p - q}{(\alpha_1' - \alpha_2')}\})$, it guarantees that $b_4^2 > 0$ and $b_5^2 > 0$. Anytime we get b_4^2 and b_5^2, we compute the temporary eigenvalues of B and find the *error* where error = abs(temporary eigenvalue of B − corresponding actual eigenvalue of B). If the error = 0 (exact solution), then we are done. Otherwise, we save the values of b_i with the smallest error and repeat the above steps until $b_3 = r$. For each time, we increase b_3 with a random number $\in (0, \frac{1}{100})$ if the length of the interval $[l, r]$ is small. Otherwise, we increase b_3 with $(r - l)$ times a random number $\in (0, \frac{1}{100})$. Of course, we will get a better solution if we increase b_3 with a very small number which varies inversely as the running time.

In conclusion, the solution is exact if *one of the three sets has a single element*, or *one of the three sets has two distinct elements*, or *all three sets have three distinct elements* but $\gamma_k = \alpha_i + \beta_j$ for $i, j, k = 1, 2, 3$. Otherwise, we will have an approximate but very close solution.

Example 4.7.1.

Input:

$$\sigma(A) = \{4, 0, -1\}$$
$$\sigma(B) = \{2, 1, -2\}$$
$$\sigma(C) = \{4.5, 2, -2.5\}.$$

Output:

$$C = A + B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & -1 \end{pmatrix} + \begin{pmatrix} 2.0000 & 0 & 0 \\ 0 & 0.2500 & 1.2990 \\ 0 & 1.2990 & -1.2500 \end{pmatrix}$$

Example 4.7.2.

Input:

$$\sigma(A) = \{12.4, 3.3, -15.7\}$$
$$\sigma(B) = \{6.8, -3.4, -5.1\}$$
$$\sigma(C) = \{16.9, 1, -19.6\}.$$

Output:

$$A = \begin{pmatrix} 12.4000 & 0 & 0 \\ 0 & 3.3000 & 0 \\ 0 & 0 & -15.7000 \end{pmatrix}, B = \begin{pmatrix} 2.2337 & 5.5613 & 0.0066 \\ 5.5613 & -0.0814 & 1.0288 \\ 0.0066 & 1.0288 & -3.8523 \end{pmatrix}$$

where the elapsed time is less than forty seconds.

Remark 4.7.3. The solution and the elapsed time may slightly vary with the same spectra since the values b_3 and b_6^2 are free.

4.8 Algorithm for the Symmetric Sum Problem: 3×3 case

This algorithm has 12 steps that are based on the previous proof.

4.8.1 Input

The prescribed spectra of 3×3 symmetric matrices A, B, and their sum C.

4.8.2 Output

The 3×3 symmetric matrices A and B if there exists a solution.

4.8.3 Algorithm

Step 1 (Get Input). Get the prescribed spectra of 3×3 symmetric matrices A, B, and their sum C.

Step 2 (Check the Length). Check whether the lengths of the prescribed eigenvalues of the symmetric matrices A, B, and C are exactly equal to 3. If one of them is not, display an error message and stop.

Step 3 (Sort the Prescribed Spectra). Sort all prescribed spectra in a non-increasing order.

Step 4 (Trace and *Weyl's* Inequalities Verification). Verify whether the prescribed spectra of the symmetric matrices A, B, and C are satisfy the trace property and *Weyl's* inequalities. If not, display an error message and stop.

If *one of the three sets has a single element*, then the solution is exact and trivial: $diag(A) + diag(B) = diag(C)$.

If *one of the three sets has two distinct elements*, then the solution is also exact, (Step 5)-(Step 9).

Step 5 (Identify the Set). Determine which set has two distinct eigenvalues in detail.

Step 6 (Shifting Spectra of A and C). Shift the prescribed spectra of A and $A+B$ (with the same number) such that $A > 0$.

Step 7 (Multiple Eigenvalues). Check whether the prescribed spectra of the matrices A and C have some multiple eigenvalues.

Step 8 (Re-check the Length). Check whether the lengths of two prescribed spectra of A and C after removing the multiple eigenvalues are exactly equal to each other. If it is not, display an error message and stop.

Step 9 (Compute the Vector x). Compute the vector $x \in \mathbf{R}^3$.

The final case is *all three sets have three distinct elements.* If there is one $\gamma_k = \alpha_i + \beta_j$, then it reduces to the 2×2 case. Therefore, the solution is exact (Step 10).

Step 10 (Spectra Verification). Check whether the prescribed spectra of A, B and C satisfy $\gamma_k = \alpha_i + \beta_j$.

Otherwise, the solution is an approximation.

Step 11 (Compute the values b_i). Compute the values b_i of the matrix B.

The following output is for the last two cases, when each set has at least two elements.

Step 12 (Display the Output). Output the 3×3 symmetric matrices A and B, where $A = diag(\alpha_1, \alpha_2, \alpha_3)$ and $B = [b_{ij}]$.

Remark 4.8.1. For the codes of this algorithm in Matlab, see [Appendix C.2, pp. 81-103].

In the last section, we construct two $n \times n$ symmetric matrices A and B with given spectra such that their sum $C = A + B$ has γ_k as its eigenvalue. The sufficient proof is taken from R.C. Thompson [Tho91].

4.9 The Symmetric Sum Problem: $n \times n$ case

Given $n \geq 2$ and

$$\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_{n-1} \geq \alpha_n \tag{4.52}$$

$$\beta_1 \geq \beta_2 \geq \dots \geq \beta_{n-1} \geq \beta_n \tag{4.53}$$

and an arbitrary number γ_k where $k = 1, 2, \cdots, n$.

Construct two symmetric matrices A and B such that

$$\sigma(A) = \{\alpha_1, \alpha_2, \dots, \alpha_{n-1}, \alpha_n\} \tag{4.54}$$

$$\sigma(B) = \{\beta_1, \beta_2, \dots, \beta_{n-1}, \beta_n\} \tag{4.55}$$

and γ_k is the k^{th} eigenvalue of the matrix sum, $A + B$.

4.10 Solution of the Symmetric Sum Problem: Case $n \times n$

The necessary and sufficient conditions are

$$\max_{i+j=k+n} \alpha_i + \beta_j \leq \gamma_k \leq \min_{i+j=k+1} \alpha_i + \beta_j.$$

4.10.1 Proof of Necessity

Let A and B be $n \times n$ symmetric matrices. Assume that the spectra of A and B satisfy (4.52)-(4.55) and γ_k is the k^{th} eigenvalue of $A + B$. We need to show that $\gamma_k \leq \alpha_i + \beta_j$ for $i + j = k + 1$, and $\gamma_k \geq \alpha_i + \beta_j$ for $i + j = k + n$.

Since γ_k is the k^{th} eigenvalue of $A+B$, by *Weyl's inequalities* (Theorem 1.2.14), for every pair of integers i, j such that $1 \leq i, j \leq n$, we have

$$\gamma_{i+j-1} \leq \alpha_i + \beta_j, \qquad i + j \leq n + 1 \tag{4.56}$$

$$\gamma_{i+j-n} \geq \alpha_i + \beta_j, \qquad i + j \geq n + 1 \tag{4.57}$$

When $k = i + j - 1$, $i + j = k + 1$ for all $k = 1, 2, \ldots n$. So (4.56) implies $\gamma_k \leq \alpha_i + \beta_j$ for $i + j = k + 1 \leq n + 1$.

Similarly, when $k = i + j - n$, $i + j = k + n$ for all $k = 1, 2, \ldots n$. So (4.57) implies $\gamma_k \geq \alpha_i + \beta_j$ for $i + j = k + n \geq n + 1$.

4.10.2 Proof of Sufficiency

Let $\{\alpha_i : i = 1, 2, ..., n\}$ and $\{\beta_i : i = 1, 2, ..., n\}$ be two sequences of real numbers such that

$$\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_{n-1} \geq \alpha_n,$$

$$\beta_1 \geq \beta_2 \geq \ldots \geq \beta_{n-1} \geq \beta_n.$$

Let γ_k be an arbitrary number such that $\gamma_k \leq \alpha_i + \beta_j$ for $i + j = k + 1$, and $\gamma_k \geq \alpha_i + \beta_j$ for $i + j = k + n$. We need to show that there exist two symmetric matrices A and B where $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $\{\beta_1, \beta_2, \ldots, \beta_n\}$ are the sets of eigenvalues of A and B respectively, and γ_k is the k^{th} eigenvalue of $A + B$.

Consider 3 cases:

Case 1. ($k = 1$) From the 2×2 case, there exist two 2×2 symmetric matrices A_1 and B_1 such that

$$\sigma(A_1) = \{\alpha_1, \alpha_n\}$$

$$\sigma(B_1) = \{\beta_1, \beta_n\}$$

$$\sigma(C_1) = \{\gamma_1, \gamma\}$$

where $C_1 = A_1 + B_1$, γ_1 is the larger eigenvalue of C_1, and $\gamma = \alpha_1 + \alpha_n + \beta_1 + \beta_n - \gamma_1$. This is possible since $\gamma_1 \leq \alpha_1 + \beta_1$, $\gamma_1 \geq \alpha_1 + \beta_n$, and $\gamma_1 \geq \alpha_n + \beta_1$.

Let $A = A_1 \oplus diag(\alpha_2, \ldots, \alpha_{n-1})$ and $B = B_1 \oplus diag(\beta_{n-1}, \ldots, \beta_2)$. If we can show that γ_1 is the largest eigenvalue of C, then we are done. Indeed,

$$C = \left(\begin{array}{c|ccc} A_1 & & & \\ \hline & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_{n-1} \end{array}\right) + \left(\begin{array}{c|ccc} B_1 & & & \\ \hline & \beta_{n-1} & & \\ & & \ddots & \\ & & & \beta_2 \end{array}\right)$$

Hence, $\gamma_1 \geq max\{\alpha_2 + \beta_{n-1}, \cdots, \alpha_{n-1} + \beta_2\}$ due to $\gamma_1 \geq \alpha_i + \beta_j$ for $i + j = 1 + n$.

Case 2. ($k = n$) Similarly, from the 2×2 case, there exist two 2×2 symmetric matrices A_1 and B_1 such that

$$\sigma(A_1) = \{\alpha_1, \alpha_n\}$$
$$\sigma(B_1) = \{\beta_1, \beta_n\}$$
$$\sigma(C_1) = \{\gamma, \gamma_n\}$$

where $C_1 = A_1 + B_1$, γ_n is the smaller eigenvalue of C_1, and $\gamma = \alpha_1 + \alpha_n + \beta_1 + \beta_n - \gamma_n$. This is possible since $\gamma_n \leq \alpha_1 + \beta_n$, $\gamma_n \leq \alpha_n + \beta_1$, and $\gamma_n \geq \alpha_n + \beta_n$.

Let $A = A_1 \oplus diag(\alpha_2, \ldots, \alpha_{n-1})$ and $B = B_1 \oplus diag(\beta_{n-1}, \ldots, \beta_2)$. If we can show that γ_n is the smallest eigenvalue of C, then we are done. Indeed,

$$C = \left(\begin{array}{c|ccc} A_1 & & & \\ \hline & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_{n-1} \end{array}\right) + \left(\begin{array}{c|ccc} B_1 & & & \\ \hline & \beta_{n-1} & & \\ & & \ddots & \\ & & & \beta_2 \end{array}\right)$$

Hence, $\gamma_n \leq min\{\alpha_2 + \beta_{n-1}, \cdots, \alpha_{n-1} + \beta_2\}$ due to $\gamma_n \leq \alpha_i + \beta_j$ for $i + j = n + 1$.

Case 3. ($2 \leq k \leq n - 1$) Again from the 2×2 case, there exist two 2×2 symmetric matrices A_1 and B_1 such that

$$\sigma(A_1) = \{\alpha_k, \alpha_n\}$$
$$\sigma(B_1) = \{\beta_1, \beta_n\}$$
$$\sigma(C_1) = \{\gamma_k, \gamma\}$$

where $C_1 = A_1 + B_1$, γ_k is the larger eigenvalue of C_1, and $\gamma_k \geq \gamma = \alpha_k + \alpha_n + \beta_1 + \beta_n - \gamma_k$. This is possible since $\gamma_k \leq \alpha_k + \beta_1$, $\gamma_k \geq \alpha_k + \beta_n$, and $\gamma_k \geq \alpha_n + \beta_1$.

Observe that only the last condition may be false. Consider 2 subcases:

Subcase 1. ($\gamma_k \geq \alpha_n + \beta_1$) Let $A = diag(\alpha_1, \ldots, \alpha_{k-1}) \oplus A_1 \oplus diag(\alpha_{k+1}, \ldots, \alpha_{n-1})$ and $B = diag(\beta_k, \ldots, \beta_2) \oplus B_1 \oplus diag(\beta_{n-1}, \ldots, \beta_{k+1})$. If we can show that γ_k is the k^{th} eigenvalue of C, then we are done. Indeed,

$$\left(\begin{array}{c|c|c} \begin{matrix}\alpha_1 & & \\ & \ddots & \\ & & \alpha_{k-1}\end{matrix} & & \\ \hline & A_1 & \\ \hline & & \begin{matrix}\alpha_{k+1} & & \\ & \ddots & \\ & & \alpha_{n-1}\end{matrix}\end{array}\right) + \left(\begin{array}{c|c|c} \begin{matrix}\beta_k & & \\ & \ddots & \\ & & \beta_2\end{matrix} & & \\ \hline & B_1 & \\ \hline & & \begin{matrix}\beta_{n-1} & & \\ & \ddots & \\ & & \beta_{k+1}\end{matrix}\end{array}\right)$$

Since $\gamma_k \leq min\{\alpha_1 + \beta_k, \cdots, \alpha_{k-1} + \beta_2\}$ due to $\gamma_k \leq \alpha_i + \beta_j$ for $i + j = k + 1$, the first direct summand of $C = A + B$ is satisfied. Observe that there are $k - 1$ elements in the first direct summand. Similarly, $\gamma_k \geq max\{\alpha_{k+1} + \beta_{n-1}, \cdots, \alpha_{n-1} + \beta_{k+1}\}$ due to $\gamma_k \geq \alpha_i + \beta_j$ for $i + j = k + n$, the trailing direct summand of $C = A + B$ is also satisfied. Finally, we have a 2×2 block C_1 with γ_k as its larger eigenvalue since $\gamma_k \geq \gamma = \alpha_k + \alpha_n + \beta_1 + \beta_n - \gamma_k$.

Subcase 2. ($\gamma_k < \alpha_n + \beta_1$) Let $A = diag(\alpha_2, \ldots, \alpha_{k-1}) \oplus A_1 \oplus diag(\alpha_{n-1}, \ldots, \alpha_k)$ and $B = diag(\beta_{k-1}, \ldots, \beta_2) \oplus B_1 \oplus diag(\beta_{k+1}, \ldots, \beta_n)$ such that

$$\sigma(A_1) = \{\alpha_1, \alpha_n\}$$

$$\sigma(B_1) = \{\beta_1, \beta_k\}$$

$$\sigma(C_1) = \{\gamma, \gamma_k\}$$

where $C_1 = A_1 + B_1$ and $\gamma = \alpha_1 + \alpha_n + \beta_1 + \beta_k - \gamma_k$. If we can show that γ_k is the k^{th} eigenvalue of C, then we are done. Indeed,

$$\left(\begin{array}{c|c|c} \begin{matrix}\alpha_2 & & \\ & \ddots & \\ & & \alpha_{k-1}\end{matrix} & & \\ \hline & A_1 & \\ \hline & & \begin{matrix}\alpha_{n-1} & & \\ & \ddots & \\ & & \alpha_k\end{matrix}\end{array}\right) + \left(\begin{array}{c|c|c} \begin{matrix}\beta_{k-1} & & \\ & \ddots & \\ & & \beta_2\end{matrix} & & \\ \hline & B_1 & \\ \hline & & \begin{matrix}\beta_{k+1} & & \\ & \ddots & \\ & & \beta_n\end{matrix}\end{array}\right)$$

Since $\gamma_k \leq min\{\alpha_2 + \beta_{k-1}, \cdots, \alpha_{k-1} + \beta_2\}$ due to $\gamma_k \leq \alpha_i + \beta_j$ for $i + j = k + 1$, the first direct summand of $C = A + B$ is satisfied. Observe that there are $k - 2$ elements in the first direct summand. Similarly, $\gamma_k \geq max\{\alpha_{n-1} + \beta_{k+1}, \cdots, \alpha_k + \beta_n\}$ due to $\gamma_k \geq \alpha_i + \beta_j$ for $i + j = k + n$, the trailing direct summand of $C = A + B$ is also satisfied. Finally, we have a 2×2 block C_1 with γ_k as its smaller eigenvalue since $\gamma_k < \alpha_n + \beta_1$ and $\gamma_k \leq \alpha_1 + \beta_k$ imply $\gamma_k \leq \gamma = \alpha_1 + \alpha_n + \beta_1 + \beta_k - \gamma_k$.

In short, for $2 \leq k \leq n - 1$, we check the condition $\gamma_k \geq \alpha_n + \beta_1$ and according to this, we set up A, B, the block A_1, and the block B_1.

4.11 Algorithm for the Symmetric Sum Problem: $n \times n$ case

This algorithm has 9 steps that are based on the previous proof.

4.11.1 Input

The prescribed spectra of two $n \times n$ symmetric matrices A and B, and an arbitrary value which is to be an eigenvalue of $C = A + B$.

4.11.2 Output

The $n \times n$ symmetric matrices A and B and the k^{th} subscript of the arbitrary eigenvalue of C if there exists a solution.

4.11.3 Algorithm

Step 1 (Get Input). Get the prescribed spectra of two $n \times n$ symmetric matrices A and B, and an arbitrary eigenvalue of C where $C = A + B$.

Step 2 (Check the Length). Check whether the lengths of two prescribed spectra of A and B are exactly equal to each other. If it is not, display an error message and stop.

Step 3 (Sort the Prescribed Spectra). Sort all prescribed spectra in a non-increasing order.

Step 4 (*Weyl*'s Inequalities Verification). Verify whether the prescribed spectra of the symmetric matrices A and B, and the arbitrary eigenvalue of C satisfy the *Weyl*'s inequalities. If it does not, display an error message and stop.

Step 5 (Determine Possible k^{th} Subscript(s)). Since the arbitrary eigenvalue of C satisfy the *Weyl*'s inequalities, determine its possible k^{th} subscript(s).

Step 6 (Choose a k^{th} Subscript). In case, the arbitrary eigenvalue of C may satisfy more than one k^{th} subscript, ask the user choose his/her favorite k^{th} subscript.

Step 7 (Compute the Eigenvalues of Block C_1). Based on the chosen k^{th} subscript, we get the corresponding blocks A_1 and B_1. Compute the eigenvalues of block C_1 where $C_1 = A_1 + B_1$.

Step 8 (*Weyl*'s Inequalities Verification for Block C_1). Verify whether the spectra of the symmetric matrices A_1 and B_1, and C_1 satisfy the *Weyl*'s inequalities. If it does not, display an error message and ask the user re-choose another possible k^{th} subscript. Repeat it until the block C_1 satisfies *Weyl's* inequalities.

Step 9 (Display the Output). Display two $n \times n$ symmetric matrices A and B and the k^{th} subscript of the arbitrary eigenvalue of C.

Remark 4.11.1. For the codes of this algorithm in Matlab, see [Appendix C.3, pp. 103-118].

4.12 Examples

Example 4.12.1.

Input:

$$\sigma(A) = \{5, 3, 1\}$$
$$\sigma(B) = \{6, 4, 2\}$$
$$\sigma(C) = \{., ., 4.5\}.$$

Output:

$$C = A + B = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} + \begin{pmatrix} 3.5625 & 1.9516 & 0 \\ 1.9516 & 4.4375 & 0 \\ 0 & 0 & 4.0000 \end{pmatrix}$$

Example 4.12.2.

Input:

$$\sigma(A) = \{3, 2, 2, -5\}$$
$$\sigma(B) = \{2.5, 2, 2, -6.5\}$$
$$\sigma(C) = \{., ., 2.3, .\}.$$

Output:

$$C = A + B = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & -5 \end{pmatrix} + \begin{pmatrix} 2.0000 & 0 & 0 & 0 \\ 0 & 2.0000 & 0 & 0 \\ 0 & 0 & -1.8371 & 4.4971 \\ 0 & 0 & 4.4971 & -2.1629 \end{pmatrix}$$

BIBLIOGRAPHY

[Axl97] Sheldon Axler, Linear Algebra Done Right, Springer-Verlag, New York, 1997.

[BF97] Richard L. Burden and J. Douglas Faires, Numerical Analysis, Brooks/Cole Publishing Company, Pacific Grove, CA, 1997.

[Bha01] Rajendra Bhatia, Linear Algebra to Quantum Cohomology: The Story of Alfred Horn's Inequalities, The American Mathematical Monthly **108** (2001), 289–318, available from http://links.jstor.org/sici?sici=0002-9890(200104)108.0.CO

[Cau41] A.L. Cauchy, Sur L'equation 'a L'aide de Laquelle on D'etermine les In'egalit'es S'eculaires des Mouvements des Plan'etes, GauthierVillars, Paris **9** (1841), 174–195, available from http://citeseer.ist.psu.edu/context/292406/0

[CG01] Moody T. Chu and Gene Golub, Inverse Eigenvalue Probelms: Theory and Applications, Department of Mathematics, North Carolina State University, North Carolina, 2001, available from http://www4.ncsu.edu:8030/ mtchu/Research/Lectures/Iep/preface.pdf

[Dav75] P. J. Davis, Interpolation and Approximation, Dover, New York, 1975.

[Fis04] Steve Fisk, A Very Short Proof of Cauchy's Interlace Theorem, The American Mathematical Monthly **112** (2004), 118, available from http://arxiv.org/PS_cache/math/pdf/0502/0502408.pdf

[HJ85] Roger A. Horn and Charles R. Johnson, Matrix Analysis, Cambridge University Press, Cambridge, UK, 1985.

[Hor62] A. Horn, Eigenvalues of Sums of Hermitian Matrices, Pacific Journal of Mathematics **12** (1962), 225–241, available from http://projecteuclid.org/Dienst/UI/1.0/Summarize/euclid.pjm/1103036720

[Hwa04] Suk-Geun Hwang, Cauchy's Interlace Theorem for Eigenvalues of Hermitian Matrices, The American Mathematical Monthly **111** (2004), 157–159, available from http://matrix.skku.ac.kr/Series-E/Monthly-E.pdf

[IIM87] Yasuhiko Ikebe, Toshiyuki Inagaki, and Sadaaki Miyamoto, The Monotonocity Theorem, Cauchy's Interlace Theorem, and the Courant-Fischer Theorem, The American Mathematical Monthly **94** (1987), 352–354, available from http://portal.acm.org/citation.cfm?id=26266.26269

[Joh89] C. R. Johnson, Precise Intervals for Specific Eigenvalues of a Product of a Positive Definite and a Hermitian Matrix, Linear Algebra and its Applications **117** (1989), 159–164, available from http://www.ams.org/bull/2000-37-03/S0273-0979-00-00865-X/home.html

[KT01] Allen Knutson and Terence Tao, Honeycombs and Sums of Hermitian Matrices, Notices of the AMS **48** (2001), 175–186, available from http://www.ams.org/notices/200102/fea-knutson.pdf

[SA03] Mandeep Singh and Jaspal Singh Aujla, A Note on Weyl's Interlacing Inequality, Mathematical Inequalities and Applications **6** (2003), 375–378, available from http://www.mia-journal.com/contents.asp?number=23

[So06] W. So, Constructive Proof of Horn's Theorem for n = 3, (2006), available from http://130.65.82.235/so

[Tho76] R. C. Thompson, The Behavior of Eigenvalues and Singular Values under Perturbations of Restricted Rank, Linear Algebra and its Applications **13** (1976), 69–78, available from http://locus.siam.org/linkedrefs/SIMAX/volume-09/0609004.html

[Tho91] ———, One Eigenvalue (Symmetric Sum Problem), (1991), available from http://130.65.82.235/so/thompson.html

[Wey12] H. Weyl, Das Asymptotische Verteilungsgestez der Eigenwert Linearer Partieller Differentialgleichungen (mit einer Anwendung auf der Theorie der Hohlraumstrahlung), Mathematische Annalen **71** (1912), 441–479, available from http://citeseer.ist.psu.edu/context/7868/0

APPENDIX A

THE CODES OF THE LEADING PRINCIPAL SUBMATRIX PROBLEM

There are 8 functions in this program.

The *main.m* function gets the spectra of an $n \times n$ symmetric matrix A and its leading principal submatrix B. It will display the matrix A if there exists a solution.

A.1 main.m

```
function [] = main()
reply = 'y';
while (reply == 'y')
    [A_eig, B_eig] = get_spectra;
    if (check_length(A_eig, B_eig))
       if (isempty(B_eig))
          A = A_eig; A
          disp('The matrix B is empty.');
       else
          [A_spec, B_spec] = descend_sort(A_eig, B_eig);
          if (interlacing_verify(A_spec, B_spec))
```

```
        [newA, newB, pointerB] = eig_coincide(A_spec, B_spec);
        if (check_newlength(newA, newB))
           n = length(B_spec);
           y = compute_y(newA, newB, pointerB, n);
           a = sum(A_spec) - sum(B_spec);
           A = zeros(1,n+1);
           for i=1:n
               A(i) = B_spec(i);
           end
           A(n+1) = a;
           A = diag(A);
           for j=1:n
               A(j,n+1) = y(j);
               A(n+1,j) = y(j);
           end
           A
           eigA = eig(A)
        end
     else
        disp('Hence, there is no solution.');
     end
  end
end
reply = input('\nDo you want more? y/n: ','s');
if isempty(reply)
    reply = 'y';
```

```
    end
  end
```

The *get_spectra.m* function gets the spectra of an $n \times n$ symmetric matrix A and its leading principal submatrix B.

A.2 get_spectra.m

```
function[A_spec, B_spec] = get_spectra() %%
A = input('\nEnter A spectrum: ','s');
A_spec = str2num(A); %%
B = input('\nEnter B spectrum: ','s');
B_spec = str2num(B);
```

The *check_length.m* function checks whether the number of the eigenvalues of the $n \times n$ symmetric matrix A is one greater than the number of the eigenvalues of its leading principal submatrix B.

A.3 check_length.m

```
function answer = check_length(A_spec, B_spec)
answer = 0;
if(length(A_spec) - length(B_spec) == 1)
   answer = 1;
elseif(length(A_spec) == 0 | length(B_spec) == 0)
   if (length(A_spec) == 0)
      disp('You did not enter any eigenvalue of the matrix A.');
   end
   if (length(B_spec) == 0)
```

```
      disp('You did not enter any eigenvalue of the submatrix B.');
   end
else
   disp('Number of the e-values of A must be one greater than ');
   disp('number of the e-values of its leading principal submatrix.');
end
```

The *descend_sort.m* function sorts the spectra of the matrix A and its leading principal submatrix B in a decreasing order.

A.4 descend_sort.m

```
function[A_spec, B_spec] = descend_sort(A_spec, B_spec) %%
A_spec = -1.* sort(-1 .* A_spec);
B_spec = -1.* sort(-1 .* B_spec);
```

The *interlacing_verify.m* function verifies whether the spectra of the matrix A and its leading principal submatrix B satisfy the interlacing property.

A.5 interlacing_verify.m

```
function flag = interlacing_verify(A_spec, B_spec)
flag = 1; %%
n = length(B_spec); i = 1;
while (i <= n)
    if ((A_spec(i) < B_spec(i)) | (B_spec(i) < A_spec(i+1)))
        flag = 0;
        disp('The spectra do not satisfy the interlacing property.');
        return;
```

```
    end
    i = i+1;
end
```

The *eigenvalue_coincide.m* function checks whether the spectrum of the leading principal submatrix B has some coincide eigenvalues.

A.6 eigenvalue_coincide.m

```
function [newA, newB, pointerB] = eig_coincide(A_spec, B_spec) %%
n = length(B_spec); i=1; j=1;
while (i <= n)
    newB(j) = B_spec(i);
    pointerB(j) = 1;
    while((i < n) & (B_spec(i) == B_spec(i+1)))
        pointerB(j) = pointerB(j)+1;
        i = i+1;
    end
    i = i+1;
    j = j+1;
end
m = length(pointerB); newA(1) = A_spec(1); u = 2; k = 1; h = 1;
while (h <= m)
    if(pointerB(h) > 1)
        k = k + pointerB(h);
    else
        k = k+1;
```

```
    end
    newA(u) = A_spec(k);
    h = h+1;
    u = u+1;
end
```

The *check_newlength.m* function checks whether the number of the eigenvalues of the matrix A is one greater than the number of the eigenvalues of its leading principal submatrix B.

A.7 check_newlength.m

```
function answer = check_newlength(A_spec, B_spec)
answer = 1;
if((length(A_spec) - length(B_spec)) ~= 1)
   answer = 0;
   disp('Something wrong in the function eig_coincide.');
   disp('Number of the e-values of A must be one greater than ');
   disp('number of the e-values of its leading principal submatrix.');
end
```

The *compute_vector_y.m* function computes the vector y.

A.8 compute_vector_y.m

```
function [Y] = compute_y(newA, newB, pointerB, q)
n = length(newB);
y = zeros(1,n);
for i=1:n
```

```
  F = zeros(1,n+1);
  g = zeros(1,n-1);
  for j=1:n+1
      F(j) = abs(newB(i) - newA(j));
  end
  p = 1;
  for k=1:n
      if (k ~= i)
          g(p) = abs(newB(i) - newB(k));
          p = p+1;
      end
  end
  F_of_beta_i = ((-1).^(n-i+1)).*prod(F);
  g_of_beta_i = ((-1).^(n-i)).*prod(g);
  if (pointerB(i) > 1)
      g_of_beta_i = g_of_beta_i .* pointerB(i);
  end
  y(i) = sqrt(- F_of_beta_i ./ g_of_beta_i);
end
r = 1;
for s=1:n
  Y(r) = y(s);
  if (pointerB(s) > 1)
     t = 1;
     while(t < pointerB(s))
          r = r+1;
```

```
            Y(r) = y(s);
            t = t+1;
        end
    end
    r = r+1;
end
Y = Y';
if(length(Y) ~= q)
    disp('The vector y is wrong!');
    Y = zeros(1,q);
end
```

APPENDIX B

THE CODES OF THE RANK-1 PERTURBATION PROBLEM

There are 10 functions in this program.

The *main.m* function gets the spectra of two $n \times n$ matrices: A and C, where $C = A + B$ and B is an $n \times n$ *rank-1* matrix. It will display the matrices A and B if there exists a solution.

B.1 main.m

```
function [] = main()
reply = 'y'; answer = 0;
while (reply == 'y')
    [A_spectrum, C_spectrum] = get_spectra;
    if (check_length(A_spectrum, C_spectrum))
        [A_spec, C_spec] = descend_sort(A_spectrum, C_spectrum);
        bool = Weyl_verify(A_spec, C_spec);
        if (bool)
           if (sum(A_spec) == sum(C_spec))
              A = diag(A_spec)
              disp('The rank-1 B is a zero matrix.');
           elseif(length(A_spec) == 1)
```

```
   A = diag(A_spec)
   B = sum(C_spec) - sum(A_spec)
else
   [A_newspec, C_newspec, negB] = shift_spectra(A_spec,
                                                  C_spec);
   if(interlacing_verify(A_newspec, C_newspec))
      [newC, newA, pointerA] = eig_coincide(C_newspec,
                                              A_newspec);
      if (check_newlength(newC, newA))
         n = length(A_newspec);
         x = compute_x(newC, newA, pointerA, n);
         if(negB == 0)
            A = diag(A_spec)
            B = x*x'
            eigB = eig(B)
            eigAB = eig(A+B)
         else
            C = diag(C_spec);
            B = -x*x'
            A = C - B
            eigB = eig(B)
            eigA = eig(A)
         end
      end
   end
end
```

```
        else
            disp('Hence, there is no solution.');
        end
    end
    reply = input('\nDo you want more? y/n: ','s');
    if isempty(reply)
        reply = 'y';
    end
end
```

The *get_spectra.m* function gets the spectra of the matrices A and $C = A + B$.

B.2 get_spectra.m

```
function[A_spec, C_spec] = get_spectra() %%
A = input('\nEnter A spectrum: ','s');
A_spec = str2num(A); %%
C = input('\nEnter C spectrum: ','s');
C_spec = str2num(C);
```

The *check_length.m* function checks whether the spectra of the matrices A and C have the same length.

B.3 check_length.m

```
function answer = check_length(A_spec, C_spec)
answer = 0;
if(length(A_spec)~=0 & length(C_spec)~=0 &
```

```
                          length(A_spec)==length(C_spec))
   answer = 1;
else
   if (isempty(A_spec))
      disp('You did not enter any eigenvalue of the matrix A.');
   end
   if (isempty(C_spec))
      disp('You did not enter any eigenvalue of the matrix C.');
   end
   if (length(A_spec) ~= length(C_spec))
      disp('The matrices A and C do not have the same length.');
   end
end
```

The *descend_sort.m* function sorts the spectra of the matrices A and C in a decreasing order.

B.4 descend_sort.m

```
function[A_spec, C_spec] = descend_sort(A_spectrum, C_spectrum)
A_spec = -1 .* sort(-1 .* A_spectrum); %%
C_spec = -1 .* sort(-1 .* C_spectrum);
```

The *Weyl_verify.m* function verifies whether the spectra of the matrices A, B, and $C = A + B$ satisfy Weyl's inequalities.

B.5 Weyl_verify.m

```
function flag = Weyl_verify(A_spec, C_spec)
```

```
flag = 1; %%
n =length(A_spec);
if (sum(C_spec) > sum(A_spec))
   beta_1 = sum(C_spec) - sum(A_spec);
   MAX = max(A_spec(1), (A_spec(n) + beta_1));
   if ((MAX > C_spec(1)) | (C_spec(1) > (A_spec(1) + beta_1)))
      flag = 0;
   else
      j = 2;
      while (j <= n)
         MIN = min(A_spec(j-1), (A_spec(j) + beta_1));
         if ((A_spec(j) > C_spec(j)) | (C_spec(j) > MIN))
            flag = 0;
            break;
         end
         j = j+1;
      end
   end
else
   beta_n = sum(C_spec) - sum(A_spec);
   MIN = min((A_spec(1) + beta_n), A_spec(n));
   if ((A_spec(n) + beta_n) > C_spec(n) | (C_spec(n) > MIN))
      flag = 0;
   else
      j = 1;
      while (j < n)
```

```
      MAX = max((A_spec(j) + beta_n), A_spec(j+1));
      if ((MAX > C_spec(j)) | (C_spec(j) > A_spec(j)))
         flag = 0;
         break;
      end
      j = j+1;
    end
  end
end
if (flag == 0)
   disp('The spectra do not satisfy the inequalities of Weyl.');
end
```

The *interlacing_verify.m* function verifies whether the spectra of the matrices A and C satisfy the interlacing property.

B.6 interlacing_verify.m

```
function flag = interlacing_verify(A_spec, C_spec)
flag = 1; %%
n = length(A_spec); i = 1;
while (i <= n)
  if ((A_spec(i) > C_spec(i)) | (A_spec(i) < C_spec(i+1)))
    flag = 0;
    disp('The spectra do not satisfy the interlacing property.');
    return;
  end
```

```
   i = i+1;
end
```

The *shift_spectra.m* function shifts the spectra of the matrices A and C such that their eigenvalues are positive.

B.7 shift_spectra.m

```
function [A_newspec, C_newspec, negB] = shift_spectra(A_spec,C_spec)
negB = 0; n = length(A_spec);
if(sum(C_spec) < sum(A_spec))
   negB = 1;
   tempA = zeros(n);
   tempC = zeros(n);
   for i=1:n
      tempA(i) = A_spec(i);
      tempC(i) = C_spec(i);
   end
   for i=1:n
      temp = A_spec(i);
      A_spec(i) = C_spec(i);
      C_spec(i) = temp;
   end
end
if(A_spec(n) <= 0)
   for i=1:n
      A_newspec(i) = A_spec(i) - A_spec(n) + 1;
```

```
        C_newspec(i) = C_spec(i) - A_spec(n) + 1;
    end
else
    for i=1:n
        A_newspec(i) = A_spec(i);
        C_newspec(i) = C_spec(i);
    end
end
C_newspec(n+1) = 0;
if(negB ~= 0)
    for i=1:n
        A_spec(i) = tempA(i);
        C_spec(i) = tempC(i);
    end
end
```

The *eig_coincide.m* function checks whether the spectrum of the leading principal submatrix A has some coincide eigenvalues.

B.8 eig_coincide.m

```
function [newC, newA, pointerA] = eig_coincide(C_spec, A_spec) %%
n = length(A_spec); i=1; j=1;
while (i <= n)
    newA(j) = A_spec(i);
    pointerA(j) = 1;
    while((i < n) & (A_spec(i) == A_spec(i+1)))
```

```
      pointerA(j) = pointerA(j)+1;
      i = i+1;
   end
   i = i+1; j = j+1;
end
m = length(pointerA); newC(1) = C_spec(1); u = 2; k = 1; h = 1;
while (h <= m)
    if(pointerA(h) > 1)
        k = k + pointerA(h);
    else
        k = k+1;
    end
    newC(u) = C_spec(k);
    h = h+1;
    u = u+1;
end
```

The *check_newlength.m* function checks whether the number of the eigenvalues of the matrix A is one greater than the number of the eigenvalues of its leading principal submatrix B.

B.9 check_newlength.m

```
function answer = check_newlength(A_spec, B_spec)
answer = 1;
if((length(A_spec) - length(B_spec)) ~= 1)
   answer = 0;
```

```
    disp('Something wrong in the function eig_coincide.');
    disp('Number of the e-values of A must be one greater than ');
    disp('number of the e-values of its leading principal submatrix.');
end
```

The *compute_x.m* function computes the vector x.

B.10 compute_x.m

```
function [X] = compute_x(newC, newA, pointerA, q)
n = length(newA);
x = zeros(1,n); F = 1; g = 1;
for i=1:n
   for j=1:n
       F = F .* abs(newA(i) - newC(j));
       if(j ~= i)
          g = g .* abs(newA(i) - newA(j));
       end
    end
    x(i) = sqrt(F ./ g);
    F = 1;
    g = 1;
 end
 r = 1;
 for s=1:n
   X(r) = x(s);
   if (pointerA(s) > 1)
```

```
      t = 1;
      while(t < pointerA(s))
            r = r+1;
            X(r) = 0;
            t = t+1;
       end
   end
   r = r+1;
end
X = X';
if(length(X) ~= q)
   disp('The vector x is wrong!');
end
```

APPENDIX C

THE CODES OF THE SYMMETRIC SUM PROBLEM

C.1 2×2 case

There are 7 functions in this program.

The *main.m* function gets the spectra of three 2×2 matrices: A, B, and $C = A + B$. It will display the matrices A and B if there exists a solution.

C.1.1 main.m

```
function [] = main()
reply = 'y';
while (reply == 'y')
   [A_eig, B_eig, C_eig] = get_spectra;
   if (check_length(A_eig, B_eig, C_eig))
      [A_spec, B_spec, C_spec] = descend_sort(A_eig, B_eig, C_eig);
      if (trace_verify(A_spec, B_spec, C_spec) &
          Weyl_verify(A_spec, B_spec, C_spec))
         if ((A_spec(1) == A_spec(2)) | (B_spec(1) == B_spec(2)))
            A = diag(A_spec); A
            B = diag(B_spec); B
         else
```

```
            A = diag(A_spec); A
            [b1, b2, b3] = compute_b(A_spec, B_spec, C_spec);
            B = [b1 b3;b3 b2]; B
            eigB = eig(B)
            eigC = eig(A+B)
         end
      else
         disp('Hence, there is no solution.');
      end
   end
   reply = input('\nDo you want more? y/n: ','s');
   if isempty(reply)
      reply = 'y';
   end
end
```

The *get_spectra.m* function gets the spectra of the matrices A, B, and C.

C.1.2 get_spectra.m

```
function[A_spec, B_spec, C_spec] = get_spectra() %%
A = input('\nEnter A spectrum: ','s');
A_spec = str2num(A); %%
B = input('\nEnter B spectrum: ','s');
B_spec = str2num(B); %%
C = input('\nEnter C spectrum: ','s');
C_spec = str2num(C);
```

The *check_length.m* function checks whether each spectrum has 2 elements.

C.1.3 check_length.m

```
function answer = check_length(A_spec, B_spec, C_spec)
answer = 0;
if (length(A_spec)==2 & length(B_spec)==2 & length(C_spec)==2)
   answer = 1;
else
   if (length(A_spec) == 0)
      disp('You did not enter any eigenvalue of the matrix A.');
   else
      if (length(A_spec) ~= 2)
         disp('Number of eigenvalues of the matrix A is not 2.');
      end
   end
   if (length(B_spec) == 0)
      disp('You did not enter any eigenvalue of the matrix B.');
   else
      if (length(B_spec) ~= 2)
         disp('Number of eigenvalues of the matrix B is not 2.');
      end
   end
   if (length(C_spec) == 0)
      disp('You did not enter any eigenvalue of the matrix C.');
   else
      if (length(C_spec) ~= 2)
```

```
        disp('Number of eigenvalues of the matrix C is not 2.');
      end
   end
end
```

The *descend_sort.m* function sorts the spectra of the matrices A, B, and C in a decreasing order.

C.1.4 descend_sort.m

```
function[A_spec,B_spec,C_spec] = descend_sort(A_spec,B_spec,C_spec)
A_spec = -1*(sort(-1*A_spec));
B_spec = -1*(sort(-1*B_spec)); %%
C_spec = -1*(sort(-1*C_spec));
```

The *trace_verify.m* function verifies whether the spectra of the matrices A, B, and C satisfy the trace property.

C.1.5 trace_verify.m

```
function flag1 = trace_verify(A_spec, B_spec, C_spec)
flag1 = 1; %%
a = A_spec(1)+A_spec(2)+B_spec(1)+B_spec(2); %%
b = C_spec(1)+C_spec(2);
a = num2str(a);
b = num2str(b);
if(~strcmp(a,b))
   flag1 = 0;
   disp('The spectra do not satisfy the trace property.');
```

```
end
```

The *Weyl_verify.m* function verifies whether the spectra of the matrices A, B, and C satisfy Weyl's inequalities.

C.1.6 Weyl_verify.m

```
function flag = Weyl_verify(A_spec, B_spec, C_spec)
flag = 1;
if((A_spec(1) + B_spec(2)) < (A_spec(2) + B_spec(1)))
    min = A_spec(1) + B_spec(2);
    max = A_spec(2) + B_spec(1);
else
    max = A_spec(1) + B_spec(2);
    min = A_spec(2) + B_spec(1);
end
if ((max > C_spec(1)) | (C_spec(1) > (A_spec(1) + B_spec(1))))
    flag = 0;
    disp('GAMMA(1) does not satisfy the inequalities of Weyl.');
    disp('GAMMA(1) should be in the interval: ');
    sprintf('[%g, %g].', max, (A_spec(1) + B_spec(1)))
end
if (((A_spec(2) + B_spec(2)) > C_spec(2)) | (C_spec(2) > min))
    flag = 0;
    disp('GAMMA(2) does not satisfy the inequalities of Weyl.');
    disp('GAMMA(2) should be in the interval: ');
    sprintf('[%g, %g].', (A_spec(2) + B_spec(2)), min)
```

```
end
```

The *compute_b.m* function computes the values of b_i.

C.1.7 compute_b.m

```
function [b1, b2, b3] = compute_b(A_spec, B_spec, C_spec) %%
b1 = (A_spec(1).*(B_spec(1)+B_spec(2))-(C_spec(1).*C_spec(2))+
     (B_spec(1).*B_spec(2))+(A_spec(1).*A_spec(2))) ./
     (A_spec(1)-A_spec(2));
b2 = ((C_spec(1).*C_spec(2))-(B_spec(1).*B_spec(2))-(A_spec(1).*
     A_spec(2))-A_spec(2).*(B_spec(1)+B_spec(2))) ./(A_spec(1)-
     A_spec(2));
b3 = sqrt(b1.*b2 - (B_spec(1).*B_spec(2)));
```

C.2 3×3 case

There are 13 functions in this program.

The *main.m* function gets the spectra of three 3×3 matrices: A, B, and $C = A + B$. It will display the matrices A and B if there exists a solution.

C.2.1 main.m

```
function [] = main()
tic
reply = 'y';
while(reply == 'y')
   [A_eig, B_eig, C_eig] = get_spectra;
   if(check_length(A_eig, B_eig, C_eig))
```

```
[A_spec, B_spec, C_spec] = descend_sort(A_eig, B_eig, C_eig);
if(trace_verify(A_spec, B_spec, C_spec) &
   Weyl_verify(A_spec, B_spec, C_spec))
   if ((A_spec(1) == A_spec(3)) | (B_spec(1) == B_spec(3)) |
                                  (C_spec(1) == C_spec(3)))
      A = diag(A_spec); A
      B = diag(B_spec); B
   elseif((A_spec(1) == A_spec(2))|(A_spec(2) == A_spec(3))|
          (B_spec(1) == B_spec(2))|(B_spec(2) == B_spec(3))|
          (C_spec(1) == C_spec(2))|(C_spec(2) == C_spec(3)))
      [A_exch, C_exch, values, answer]= identify(A_spec,
                                            B_spec, C_spec);
      [A_newspec, C_newspec] = shift_spectra_A(A_exch, C_exch);
      [newC, newA, pointerA] = eig_coincide(C_newspec,
                                         A_newspec);
      if (check_newlength(newC, newA))
         n = length(A_newspec);
         x = compute_x(newC, newA, pointerA, n);
         switch answer
            case 1
               C = diag(A_exch) - values(1)*eye(3);
               A = -(x*x') - values(1)*eye(3)
               B = C - A
               eigA = eig(A)
               eigB = eig(B)
               eigAB = eig(A+B)
```

```
    disp('Case: the first two eigenvalues of A
                are equal.');
case 2
    B = diag(B_spec)
    A = x*x' + values(1)*eye(3)
    eigA = eig(A)
    eigB = eig(B)
    eigAB = eig(A+B)
    disp('Case: the last two eigenvalues of A
                are equal.');
case 3
    C = diag(A_exch) - values(2)*eye(3);
    B = -(x*x') - values(2)*eye(3)
    A = C - B
    eigA = eig(A)
    eigB = eig(B)
    eigAB = eig(A+B)
    disp('Case: the first two eigenvalues of B
                are equal.');
case 4
    A = diag(A_spec)
    B = x*x' + values(2)*eye(3)
    eigA = eig(A)
    eigB = eig(B)
    eigAB = eig(A+B)
    disp('Case: the last two eigenvalues of B
```

```
                        are equal.');
        case 5
            B = diag(B_spec)
            C = -(x*x') - values(3)*eye(3);
            A = C - B
            eigA = eig(A)
            eigB = eig(B)
            eigAB = eig(A+B)
            disp('Case: the first two eigenvalues of C
                        are equal.');
        otherwise
            A = -diag(A_exch) + values(3)*eye(3)
            C = x*x' + values(3)*eye(3);
            B = C - A
            eigA = eig(A)
            eigB = eig(B)
            eigAB = eig(A+B)
            disp('Case: the last two eigenvalues of C
                        are equal.');
      end
    end
else
    [A, B, answ] = gamma_verify(A_spec, B_spec, C_spec);
    if(answ)
        A
        B
```

```
            eigB = eig(B)
            eigAB = eig(A+B)
         else
            [b1,b2,b3,b4,b5,b6,flag] = compute_bi(A_spec, B_spec,
                                                              C_spec);
            B = [b1 b4 b6; b4 b2 b5; b6 b5 b3];
            B = B + B_spec(3).*eye(3);
            A = diag(A_spec); A
            if(flag)
               disp('The matrix B is an approximation.');
            end
            B
            eiB = eig(B)
            eiAB = eig(A+B)
         end
      end
   else
      disp('Hence, there is no solution.');
   end
end
reply = input('\nDo you want more? y/n: ','s');
if isempty(reply)
   reply = 'y';
end
toc
end
```

The *get_spectra.m* function gets the spectra of the matrices A, B, and C.

C.2.2 get_spectra.m

```
function[A_spec, B_spec, C_spec] = get_spectra() %%
A = input('\nEnter A spectrum: ','s');
A_spec = str2num(A); %%
B = input('\nEnter B spectrum: ','s');
B_spec = str2num(B); %%
C = input('\nEnter C spectrum: ','s');
C_spec = str2num(C);
```

The *check_length.m* function checks whether each spectrum has 3 elements.

C.2.3 check_length.m

```
function answer = check_length(A_spec, B_spec, C_spec)
answer = 0;
if (length(A_spec)==3 & length(B_spec)==3 & length(C_spec)==3)
   answer = 1;
else
   if (length(A_spec) == 0)
      disp('You did not enter any eigenvalue of the matrix A.');
   else
      if (length(A_spec) ~= 3)
         disp('Number of eigenvalues of the matrix A is not 3.');
      end
   end
```

```
    if (length(B_spec) == 0)
        disp('You did not enter any eigenvalue of the matrix B.');
    else
        if (length(B_spec) ~= 3)
            disp('Number of eigenvalues of the matrix B is not 3.');
        end
    end
    if (length(C_spec) == 0)
        disp('You did not enter any eigenvalue of the matrix C.');
    else
        if (length(C_spec) ~= 3)
            disp('Number of eigenvalues of the matrix C is not 3.');
        end
    end
end
```

The *descend_sort.m* function sorts the spectra of the matrices A, B, and C in a decreasing order.

C.2.4 descend_sort.m

```
function[A_spec, B_spec, C_spec] = descend_sort(A_spec, B_spec,
                                                        C_spec)
A_spec = -1*(sort(-1*A_spec)); %%
B_spec = -1*(sort(-1*B_spec));
C_spec = -1*(sort(-1*C_spec));
```

The *trace_verify.m* function verifies whether the spectra of the matrices A, B, and C satisfy the trace property.

C.2.5 trace_verify.m

```
function flag1 = trace_verify(A_spec, B_spec, C_spec)
flag1 = 1; %%
a = sum(A_spec) + sum(B_spec);
b = sum(C_spec);
a = num2str(a); %%
b = num2str(b);
if (~strcmp(a,b))
   flag1 = 0;
   disp('The spectra do not satisfy the trace property.');
end
```

The *Weyl_verify.m* function verifies whether the spectra of the matrices A, B, and C satisfy Weyl's inequalities.

C.2.6 Weyl_verify.m

```
function flag = Weyl_verify(A_spec, B_spec, C_spec)
flag = 1;
if((A_spec(1) + B_spec(2)) < (A_spec(2) + B_spec(1)))
    min = A_spec(1) + B_spec(2);
else
    min = A_spec(2) + B_spec(1);
end
if ((A_spec(2) + B_spec(3)) < (A_spec(3) + B_spec(2)))
```

```
    max = A_spec(3) + B_spec(2);
else
   max = A_spec(2) + B_spec(3);
end
for i=1:3
   temp(i) = A_spec(i) + B_spec(4-i);
end
temp = -1 .* sort(-1.*temp); mymax = temp(1); mymin = temp(3);
if ((mymax > C_spec(1)) | (C_spec(1) > (A_spec(1) + B_spec(1))))
    flag = 0;
    disp('GAMMA(1) does not satisfy the inequalities of Weyl!');
    disp('GAMMA(1) should be in the interval: ');
    sprintf('[%g, %g].', mymax, (A_spec(1) + B_spec(1)))
end
if ((max > C_spec(2)) | (C_spec(2) > min))
    flag = 0;
    disp('GAMMA(2) does not satisfy the inequalities of Weyl!');
    disp('GAMMA(2) should be in the interval: ');
    sprintf('[%g, %g].', max, min)
end
if (((A_spec(3) + B_spec(3)) > C_spec(3)) | (C_spec(3) > mymin))
    flag = 0;
    disp('GAMMA(3) does not satisfy the inequalities of Weyl!');
    disp('GAMMA(3) should be in the interval: ');
    sprintf('[%g, %g].', (A_spec(3) + B_spec(3)), mymin)
end
```

The *identify.m* function checks whether one of the three spectra has two distinct elements.

C.2.7 identify.m

```
function [A_exch, C_exch, values, answer] = identify(A_spec, B_spec,
                                                              C_spec)
answer = 0; A2 = 0; B2 = 0; C2 = 0; values = zeros(3);
if((A_spec(1) == A_spec(2))|(A_spec(2) == A_spec(3)))
   if(A_spec(1) == A_spec(2))
      A_exchanged = -1 * A_spec;
      A_exchanged = -1*(sort(-1*A_exchanged));
      values(1) = A_exchanged(2);
      for i=1:3
         A_exch(i) = C_spec(i) + A_exchanged(2);
         C_exch(i) = B_spec(i);
      end
      answer = 1;
   else
      answer = 2;
      for i=1:3
         A_exch(i) = B_spec(i) + A_spec(2);
         C_exch(i) = C_spec(i);
      end
      values(1) = A_spec(2);
   end
elseif((B_spec(1) == B_spec(2))|(B_spec(2) == B_spec(3)))
```

```
if(B_spec(1) == B_spec(2))
   B_exchanged = -1 * B_spec;
   B_exchanged = -1*(sort(-1*B_exchanged));
   values(2) = B_exchanged(2);
   for i=1:3
      A_exch(i) = C_spec(i) + B_exchanged(2);
      C_exch(i) = A_spec(i);
   end
   answer = 3;
else
   answer = 4;
   for i=1:3
      A_exch(i) = A_spec(i) + B_spec(2);
      C_exch(i) = C_spec(i);
   end
   values(2) = B_spec(2);
end
else
if(C_spec(1) == C_spec(2))
   C_exchanged = -1 * C_spec;
   C_exchanged = -1*(sort(-1*C_exchanged));
   values(3) = C_exchanged(2);
   A_exchanged = -1 * A_spec;
   A_exchanged = -1*(sort(-1*A_exchanged));
   for i=1:3
      A_exch(i) = B_spec(i) + C_exchanged(2);
```

```
      C_exch(i) = A_exchanged(i);
    end
    answer = 5;
  else
    answer = 6;
    A_exchanged = -1 * A_spec;
    A_exchanged = -1*(sort(-1*A_exchanged));
    for i=1:3
      A_exch(i) = A_exchanged(i) + C_spec(2);
      C_exch(i) = B_spec(i);
    end
    values(3) = C_spec(2);
  end
end
```

The *shift_spectra_A.m* function shifts the spectra of the matrices A and C such that their eigenvalues are positive.

C.2.8 shift_spectra_A.m

```
function [A_newspec, C_newspec] = shift_spectra_A(A_exch, C_exch) %%
n = length(A_exch);
if(A_exch(n) <= 0)
  for i=1:n
    A_newspec(i) = A_exch(i) - A_exch(n) + 1;
    C_newspec(i) = C_exch(i) - A_exch(n) + 1;
  end
```

```
else
   for i=1:n
      A_newspec(i) = A_exch(i);
      C_newspec(i) = C_exch(i);
   end
end
C_newspec(n+1) = 0;
```

The *eig_coincide.m* function checks whether the spectrum of the leading principal submatrix A has some coincide eigenvalues.

C.2.9 eig_coincide.m

```
function [newC, newA, pointerA] = eig_coincide(C_spec, A_spec) %%
n = length(A_spec); i=1; j=1;
while (i <= n)
    newA(j) = A_spec(i);
    pointerA(j) = 1;
    while((i < n) & (A_spec(i) == A_spec(i+1)))
        pointerA(j) = pointerA(j)+1;
        i = i+1;
    end
    i = i+1;
    j = j+1;
end
m = length(pointerA); newC(1) = C_spec(1); u = 2; k = 1; h = 1;
while (h <= m)
```

```
    if(pointerA(h) > 1)
        k = k + pointerA(h);
    else
        k = k+1;
    end
    newC(u) = C_spec(k);
    h = h+1;
    u = u+1;
end
```

The *check_newlength.m* function checks whether the number of the eigenvalues of the matrix C is one greater than the number of the eigenvalues of its leading principal submatrix A.

C.2.10 check_newlength.m

```
function answer = check_newlength(C_spec, A_spec)
answer = 1;
if((length(C_spec) - length(A_spec)) ~= 1)
   answer = 0;
   disp('Something wrong in the function eig_coincide.');
   disp('Number of the e-values of C must be one greater than ');
   disp('number of the e-values of its leading principal submatrix.');
end
```

The *compute_x.m* function computes the vector x.

C.2.11 compute_x.m

```
function [X] = compute_x(newC, newA, pointerA, q)
n = length(newA);
x = zeros(1,n); F = 1; g = 1;
for i=1:n
   for j=1:n
       F = F .* abs(newA(i) - newC(j));
       if(j ~= i)
          g = g .* abs(newA(i) - newA(j));
       end
    end
    x(i) = sqrt(F ./ g);
    F = 1;
    g = 1;
 end
 r = 1;
 for s=1:n
   X(r) = x(s);
   if (pointerA(s) > 1)
      t = 1;
      while(t < pointerA(s))
            r = r+1;
            X(r) = 0;
            t = t+1;
       end
```

```
   end
   r = r+1;
end
X = X';
if(length(X) ~= q)
   disp('The vector x is wrong.');
end
```

The *gamma_verify.m* function verifies whether $\gamma_k = \alpha_i + \beta_j$.

C.2.12 gamma_verify.m

```
function [A, B, answ] = gamma_verify(A_spec, B_spec, C_spec) %%
A = eye(3); B = eye(3); answ = 0; flag1 = 0;
for i=1:3
   for j=1:3
      if(C_spec(1) == A_spec(i) + B_spec(j))
         flag1 = 1;
         A_index = i;
         B_index = j;
         C_index = 1;
         break;
      end
      if(C_spec(2) == A_spec(i) + B_spec(j))
         flag1 = 1;
         A_index = i;
         B_index = j;
```

```
          C_index = 2;
          break;
       end
       if(C_spec(3) == A_spec(i) + B_spec(j))
          flag1 = 1;
          A_index = i;
          B_index = j;
          C_index = 3;
          break;
       end
    end
end
if(flag1==1)
    answ = 1;
    A_s = zeros(2);
    B_s = zeros(2);
    C_s = zeros(2);
    v = 1;
    for k=1:3
       if(A_index ~= k)
          A_s(v) = A_spec(k);
          v = v+1;
       end
    end
    v = 1;
    for k=1:3
```

```
        if(B_index ~= k)
            B_s(v) = B_spec(k);
            v = v+1;
        end
    end
    v = 1;
    for k=1:3
        if(C_index ~= k)
            C_s(v) = C_spec(k);
            v = v+1;
        end
    end
b1 = (A_s(1).*(B_s(1)+B_s(2))-(C_s(1).*C_s(2))+(B_s(1).*B_s(2))+
      (A_s(1).*A_s(2)))./(A_s(1)-A_s(2)); %%
b2 = ((C_s(1).*C_s(2))-(B_s(1).*B_s(2))-(A_s(1).*A_s(2))-A_s(2).*
       (B_s(1)+B_s(2)))./(A_s(1)-A_s(2));
b3 = sqrt(b1.*b2 - (B_s(1).*B_s(2))); %%
A = [A_spec(A_index),0,0;0,A_s(1),0; 0,0,A_s(2)]; %%
B = [B_spec(B_index),0,0; 0,b1,b3; 0,b3,b2];
end
```

The *compute_bi.m* function computes the values of b_i.

C.2.13 compute_bi.m

```
function [b1,b2,b3,b4,b5,b6, flag] = compute_bi(A_spec, B_spec,
                                                       C_spec)
```

```
syms b3 b6
flag = 1; G = zeros(5);
for i=1:3
   A_s(i) = A_spec(i) - A_spec(3);
   B_s(i) = B_spec(i) - B_spec(3);
   C_s(i) = C_spec(i) - A_spec(3) - B_spec(3);
end
det_C = prod(C_s);
alter_A = A_s(1).*A_s(2); %%
alter_B = B_s(1).*B_s(2); %%
alter_C = C_s(1).*C_s(2) + C_s(1).*C_s(3) + C_s(2).*C_s(3); %%
k = alter_C - alter_B - alter_A; %%
b1 = (A_s(1).*(B_s(1)+B_s(2)) + A_s(2).*b3 - k)./(A_s(1)-A_s(2)); %%
b2 = (k - A_s(2).*(B_s(1)+B_s(2)) - A_s(1).*b3)./(A_s(1)-A_s(2));
det_bb = b1.*b2.*b3;
alter_bb = b1.*b2 + b1.*b3 + b2.*b3; %%
p = alter_bb - alter_B; %%
q = (b1+A_s(1)).*(b2+A_s(2)).*b3 - det_bb - det_C; %%
b4_sq = (A_s(1).*p - (A_s(1)-A_s(2)).*(b6)^2 - q)./A_s(1); %%
b5_sq = (q - A_s(2).*(b6)^2)./A_s(1);
Q = q./A_s(2); %%
L = solve(Q,b3);
L = double(L); %%
F = (A_s(1).*p - q)./(A_s(1)-A_s(2));
R = solve(F,b3); %%
R = double(R);
```

```
if(min(L) < min(R))
   l = min(R);
else
   l = min(L);
end
if(max(L) < max(R))
   r = max(L);
else
   r = max(R);
end
l = double(l);
r = double(r);
err = abs(B_s(1));
b3 = l; %%
b1 = (A_s(1).*(B_s(1)+B_s(2)) + A_s(2).*b3 - k)./(A_s(1)-A_s(2)); %%
b2 = (k - A_s(2).*(B_s(1)+B_s(2)) - A_s(1).*b3)./(A_s(1)-A_s(2));
Q_value = double(eval(Q));
F_value = double(eval(F)); %%
g = eval(det_bb - b1*b5_sq - b2*(b6)^2 - b3*b4_sq+2*b4_sq*b5_sq*b6);
g = simplify(expand(g));
G = solve(g,b6);
ans = 1; %%
while(ans ~= 0 & b3 <= r)
   if(Q_value > 0 & F_value > 0)
      if(Q_value < F_value)
         m = Q_value;
```

```
else
   m = F_value;
end
for i=1:5
   w = double(G(i));
   if(isreal(w) & w > 0 & w < m)
      b6 = double(G(i));
      b4_sq = eval((A_s(1).*p - (A_s(1)-A_s(2)).*
                                  (b6)^2 - q)./A_s(1));
      b5_sq = eval((q - A_s(2).*(b6)^2)./A_s(1));
      if(isreal(b4_sq)&isreal(b5_sq) & b4_sq >= 0 & b5_sq >= 0)
         b4 = sqrt(b4_sq);
         b5 = sqrt(b5_sq);
         B = [b1 b4 b6; b4 b2 b5; b6 b5 b3];
         eigenB = eig(B);
         if(eigenB(1) == B_s(1) & eigenB(2) == B_s(2))
            disp('The matrix B is an exact solution.');
            ans = 0;
            flag = 0;
            break;
         else
            if(abs(eigenB(1)-B_s(1)) <= err &
               abs(eigenB(2)-B_s(2)) <= err)
               my_b1 = b1;
               my_b2 = b2;
               my_b3 = b3;
```

```
                  my_b4 = b4;
                  my_b5 = b5;
                  my_b6 = b6;
                  err = min(abs(eigenB(1)-B_s(1)),
                            abs(eigenB(2)-B_s(2)));
               end
            end
         end
      end
   end
end
if((r-1) < 5)
   b3 = b3 + rand(1)./100;
else
   b3 = b3 + (r-1)*rand(1)./100;
end
b1 = (A_s(1).*(B_s(1)+B_s(2)) + A_s(2).*b3 - k)./(A_s(1)-A_s(2));
b2 = (k - A_s(2).*(B_s(1)+B_s(2)) - A_s(1).*b3)./(A_s(1)-A_s(2));
Q_value = double(eval(Q));
F_value = double(eval(F));
syms b6
b4_sq = (A_s(1).*p - (A_s(1)-A_s(2)).*(b6)^2 - q)./A_s(1);
b5_sq = (q - A_s(2).*(b6)^2)./A_s(1);
g = eval(det_bb - b1*b5_sq - b2*(b6)^2 - b3*b4_sq + 2*b4_sq*b5_sq*b6);
g = simplify(expand(g));
G = solve(g,b6);
```

```
end
if(ans ~= 0)
   b1 = my_b1;
   b2 = my_b2;
   b3 = my_b3;
   b4 = my_b4;
   b5 = my_b5;
   b6 = my_b6;
end
if((b1 ~= 0 & abs(b1) < 1./1000000) |
   (b2 ~= 0 & abs(b2) < 1./1000000) |
   (b3 ~= 0 & abs(b3) < 1./1000000) |
   (b4 ~= 0 & abs(b4) < 1./1000000) |
   (b5 ~= 0 & abs(b5) < 1./1000000) |
   (b6 ~= 0 & abs(b6) < 1./1000000))
   disp('Note: "*" means 0.');
end
```

C.3 $n \times n$ case

There are 9 functions in this program.

The *main.m* function gets the spectra of two $n \times n$ matrices: A and B, and an arbitrary eigenvalue of the matrix $C = A + B$. It will display the matrices A and B if there exists a solution.

C.3.1 main.m

```
function [] = main()
```

```
reply = 'y';
while (reply == 'y')
   [A_spectrum, B_spectrum, gamma] = get_spectra;
   if (check_length(A_spectrum, B_spectrum, gamma))
      [A_spec, B_spec] = descend_sort(A_spectrum, B_spectrum);
      [bool, K] = Weyl_verify(A_spec, B_spec, gamma);
      if (bool & K)
         n = length(A_spec);
         if ((A_spec(1) == A_spec(n)) | (B_spec(1) == B_spec(n)))
            A = diag(A_spec); A
            B = diag(B_spec); B
            k_subscript_of_gamma = K
            eigA = eig(A)
            eigB = eig(B)
            eigAB = eig(A+B)
         else
            A_temp = zeros(1,n);
            B_temp = zeros(1,n);
            switch (K)
               case {1,n}
                  B1 = compute_B1(A_spec(1),A_spec(n), B_spec(1),
                                                    B_spec(n), gamma);
                  for i=2:n-1
                     A_temp(i+1) = A_spec(i);
                     B_temp(i+1) = B_spec(n+1-i);
                  end
```

```
      A_temp(1) = A_spec(1);
      A_temp(2) = A_spec(n);
      B_temp(1) = B1(1,1);
      B_temp(2) = B1(2,2);
      B = diag(B_temp);
      B(1,2) = B1(1,2);
      B(2,1) = B1(1,2);
      A = diag(A_temp)
      B
      k_subscript_of_gamma = K
      eigA = eig(A)
      eigB = eig(B)
      eigAB = eig(A+B)
   otherwise
      if (gamma >= (A_spec(n)+B_spec(1)))
         B1 = compute_B1(A_spec(K),A_spec(n), B_spec(1),
                                          B_spec(n), gamma);
         for i=1:K-1
            A_temp(i) = A_spec(i);
            B_temp(i) = B_spec(K+1-i);
         end
         for j=K+2:n
            A_temp(j) = A_spec(j-1);
            B_temp(j) = B_spec(n+K+1-j);
         end
         A_temp(K) = A_spec(K);
```

```
   A_temp(K+1) = A_spec(n);
   B_temp(K) = B1(1,1);
   B_temp(K+1) = B1(2,2);
   B = diag(B_temp);
   B(K,K+1) = B1(1,2);
   B(K+1,K) = B1(1,2);
   A = diag(A_temp)
   B
   k_subscript_of_gamma = K
   eigA = eig(A)
   eigB = eig(B)
   eigAB = eig(A+B)
else
   B1 = compute_B1(A_spec(1),A_spec(n), B_spec(1),
                                    B_spec(K), gamma);
   for i=1:K-2
      A_temp(i) = A_spec(i+1);
      B_temp(i) = B_spec(K-i);
   end
   for j=K+1:n
      A_temp(j) = A_spec(n+K-j);
      B_temp(j) = B_spec(j);
   end
   A_temp(K-1) = A_spec(1);
   A_temp(K) = A_spec(n);
   B_temp(K-1) = B1(1,1);
```

```
                        B_temp(K) = B1(2,2);
                        B = diag(B_temp);
                        B(K-1,K) = B1(1,2);
                        B(K,K-1) = B1(1,2);
                        A = diag(A_temp)
                        B
                        k_subscript_of_gamma = K
                        eigA = eig(A)
                        eigB = eig(B)
                        eigAB = eig(A+B)
                     end
                  end
               end
            else
               disp('Hence, there is no solution.');
            end
         end
      reply = input('\nDo you want more? y/n: ','s');
      if isempty(reply)
         reply = 'y';
      end
   end
```

The *get_spectra.m* function gets the spectra of the matrices A and B, and an arbitrary eigenvalue of the matrix $C = A + B$.

C.3.2 get_spectra.m

```
function[A_spec, B_spec, gamma_k] = get_spectra() %%
A = input('\nEnter A spectrum: ','s');
A_spec = str2num(A); %%
B = input('\nEnter B spectrum: ','s');
B_spec = str2num(B); %%
C = input('\nEnter "GAMMA(k)", an arbitrary eigenvalue of C: ','s');
gamma_k = str2num(C);
```

The *check_length.m* function checks whether the spectra of the matrices A and B have the same length. It also verifies whether only one γ_k is input.

C.3.3 check_length.m

```
function answer = check_length(A_spec, B_spec, gamma_k)
answer = 0;
if (length(A_spec) ~= 0 & length(B_spec) ~= 0 & length(gamma_k) == 1
    & length(A_spec) == length(B_spec))
   answer = 1;
else
   if (isempty(A_spec))
      disp('You did not enter any eigenvalue of the matrix A.');
   end
   if (isempty(B_spec))
      disp('You did not enter any eigenvalue of the matrix B.');
   end
   if (length(gamma_k) ~= 1)
```

```
      disp('Please enter one eigenvalue of matrix C.');
   end
   if (length(A_spec) ~= length(B_spec))
      disp('The matrices A and B do not have the same length.');
   end
end
```

The *descend_sort.m* function sorts the spectra of the matrices A and B in a decreasing order.

C.3.4 descend_sort.m

```
function[A_spec, B_spec] = descend_sort(A_spectrum, B_spectrum)
A_spec = -1 .* sort(-1 .* A_spectrum); %%
B_spec = -1 .* sort(-1 .* B_spectrum);
```

The *Weyl_verify.m* function verifies whether the arbitrary eigenvalue of the matrix $C = A + B$ satisfies Weyl's inequalities.

C.3.5 Weyl_verify.m

```
function [flag, k_index] = Weyl_verify(A_spectrum, B_spectrum,
                                                        gamma_k)
flag = 1; k_index = 0; sorry = Inf; k_indices = []; %%
n = length(A_spectrum); %%
if (gamma_k > (A_spectrum(1) + B_spectrum(1)) | gamma_k <
                                 (A_spectrum(n) + B_spectrum(n)))
   flag = 0;
   disp('GAMMA(k) does not satisfy the inequalities of Weyl.');
```

```
   disp('GAMMA(k) should be in the interval: ');
   sprintf('[%g, %g].', (A_spectrum(n) + B_spectrum(n)),
                        (A_spectrum(1) + B_spectrum(1)))
   sorry = 0;
else
   temp = zeros(1,n);
   for i=1:n
      temp(i) = A_spectrum(i) + B_spectrum(n+1-i);
   end
   if (gamma_k >= max(temp))
      k_indices = 1;
   end
   if (gamma_k <= min(temp) & n > 1)
      if (isempty(k_indices))
         k_indices = n;
      else
         k_indices = [k_indices, n];
      end
   end
   for k=2:n-1
      a = Inf;
      b = -Inf;
      for j=1:k
         temp1 = A_spectrum(j) + B_spectrum(k+1-j);
         if (a > temp1)
            a = temp1;
```

```
            end
         end
         for p=1:n-k+1
            temp2 = A_spectrum(p+k-1) + B_spectrum(n+1-p);
            if (b < temp2)
               b = temp2;
            end
         end
         if (gamma_k <= a & gamma_k >= b)
            if (isempty(k_indices))
               k_indices = k;
            else
               k_indices = [k_indices, k];
            end
         end
   end
end
if (~isempty(k_indices))
   sorry = 0;
   k_index = k_subscript_verify(A_spectrum, B_spectrum, k_indices,
                                                           gamma_k);
end
if (k_index == 0 & sorry == Inf)
    disp('GAMMA(k) does not satisfy the inequalities of Weyl.');
end
```

The *choose_k_index.m* function asks the users for choosing their desired k^{th} subscript of the eigenvalue γ_k.

C.3.6 choose_k_index.m

```
function k_index = choose_k_index(k_indices, subscripts_size) %%
again = 1; flag3 = 1;
if (subscripts_size < 1)
    k_index =0;
else
    disp('The possible k_subscript(s) of the eigenvalue GAMMA(k): ');
    for i=1:subscripts_size
        subscripts(i) = k_indices(i);
    end
    subscripts
    temp = input('\nChoose your desired k_subscript
                    (by the above value(s)): ','s');
    temp = str2num(temp);
    while (again)
        i= 1;
        while (i <= subscripts_size & flag3)
            if (temp == k_indices(i))
                flag3 = 0;
            end
            i = i+1;
        end
        if (flag3)
```

```
            disp('You did not choose any k_subscript in the list: ');
            for i=1:subscripts_size
                subscripts(i) = k_indices(i);
            end
            subscripts
            temp = input('\nPlease, choose your desired
                                   k_subscript again: ','s');
            temp = str2num(temp);
        else
            k_index = temp;
            again = 0;
        end
    end
end
```

The *k_subscript_verify.m* function verifies whether the eigenvalue γ_k gives a real solution when calculating the 2×2 matrix B_1. Notice that all k subscripts in the list *k_indices* satisfy Weyl's inequalities. However, some (or all) of them may not give a real solution.

C.3.7 k_subscript_verify.m

```
function k_index = k_subscript_verify(A_spectrum, B_spectrum,
                                                   k_indices, gamma_k)
k_index = 0; subscripts_size = 0; %%
n = length(A_spectrum);
if(length(k_indices) > 1)
```

```
flagB1 = 1;
subscripts_size = length(k_indices);
k_index = choose_k_index(k_indices, subscripts_size);
while (flagB1 & k_index > 1 & k_index < n)
   if ((A_spectrum(1) == A_spectrum(n)) | (B_spectrum(1) ==
                                            B_spectrum(n)))
      flagB1 = 0;
   else
      if (gamma_k >= (A_spectrum(n) + B_spectrum(1)))
         if (B1_Weyl_verify(A_spectrum(k_index), A_spectrum(n),
                       B_spectrum(1), B_spectrum(n), gamma_k))
            flagB1 = 0;
         else
            disp('Your chosen subscript of gamma satisfies the
                  inequalities of Weyl, but it does not satisfy
                  when computing the 2x2 matrix B1.');
            temp_index = k_indices(subscripts_size);
            i=1; stop = 1;
            while (i < subscripts_size & stop)
               if (k_index == k_indices(i))
                  k_indices(i) = temp_index;
                  k_indices(subscripts_size) = k_index;
                  stop = 0;
               end
               i = i+1;
            end
```

```
        subscripts_size = subscripts_size-1;
        if (subscripts_size > 0)
           disp('Please re-choose another subscript.');
        end
        k_index = choose_k_index(k_indices, subscripts_size);
     end
  else
     if (B1_Weyl_verify(A_spectrum(1), A_spectrum(k_index),
                     B_spectrum(1), B_spectrum(n), gamma_k))
        flagB1 = 0;
     else
        disp('Your chosen subscript of gamma satisfies the
              inequalities of Weyl, but it does not satisfy
              when computing the 2x2 matrix B1.');
        temp_index = k_indices(subscripts_size);
        i=1; stop = 1;
        while (i < subscripts_size & stop)
           if (k_index == k_indices(i))
              k_indices(i) = temp_index;
              k_indices(subscripts_size) = k_index;
              stop = 0;
           end
           i = i+1;
        end
        subscripts_size = subscripts_size-1;
        if (subscripts_size > 0)
```

```
                disp('Please re-choose another subscript.');
                end
                k_index = choose_k_index(k_indices, subscripts_size);
            end
        end
    end
end
else
    if ((A_spectrum(1) == A_spectrum(n)) | (B_spectrum(1) ==
                                            B_spectrum(n)))
        k_index = k_indices;
    else
        if (gamma_k >= (A_spectrum(n) + B_spectrum(1)))
            if (B1_Weyl_verify(A_spectrum(k_indices), A_spectrum(n),
                            B_spectrum(1), B_spectrum(n), gamma_k))
                k_index = k_indices;
            else
                disp('This eigenvalue of C satisfies the inequalities
                        of Weyl, but it does not satisfy when computing
                        the 2x2 matrix B1.');
            end
        else
            if (B1_Weyl_verify(A_spectrum(1), A_spectrum(k_indices),
                            B_spectrum(1), B_spectrum(n), gamma_k))
                k_index = k_indices;
            else
```

```
          disp('This eigenvalue of C satisfies the inequalities
                of Weyl, but it does not satisfy when computing
                the 2x2 matrix B1.');
        end
      end
   end
end
```

The *compute_B1.m* function computes the 2×2 matrix B_1.

C.3.8 compute_B1.m

```
function [B1] = compute_B1(alpha_1,alpha_n, beta_1, beta_n, gamma_1)
if (alpha_1 ~= alpha_n)
   gamma_n = (alpha_1+alpha_n+beta_1+beta_n) - gamma_1;
   b11 = (alpha_1.*(beta_1+beta_n) - gamma_1.*gamma_n + alpha_1.*
          alpha_n + beta_1.*beta_n)./(alpha_1-alpha_n);
   b22 = (-alpha_n.*(beta_1+beta_n) + gamma_1.*gamma_n - alpha_1.*
           alpha_n - beta_1.*beta_n)./(alpha_1-alpha_n);
   b12 = sqrt(b11.*b22 - beta_1.*beta_n);
   B1 = [b11 b12; b12 b22];
else
   B1 = [beta_1 0; 0 beta_n];
end
if((b11 ~= 0 & abs(b11) < 1./1000000) |
   (b12 ~= 0 & abs(b12) < 1./1000000) |
   (b22 ~= 0 & abs(b22) < 1./1000000))
```

```
   disp('Note: "*" means 0.');
end
```

The *B1_Weyl_verify.m* function verifies whether the spectra of the 2×2 matrices A_1, B_1, and $C_1 = A_1 + B_1$ satisfy Weyl's inequalities.

C.3.9 B1_Weyl_verify.m

```
function flag2 = B1_Weyl_verify(alpha_1, alpha_k, beta_1, beta_n,
                                                                gamma_k)
flag2 = 0;
if ((alpha_1+beta_n) <= (alpha_k+beta_1))
    min = alpha_1 + beta_n;
    max = alpha_k + beta_1;
else
    max = alpha_1 + beta_n;
    min = alpha_k + beta_1;
 end
 if (((max <= gamma_k) & (gamma_k <= alpha_1+beta_1)) |
     ((alpha_k+beta_n <= gamma_k) & (gamma_k <= min)))
    flag2 = 1;
end
```