2017

Modifying Some Iterative Methods for Solving Quadratic Eigenvalue Problems

Ali Hasan Ali
Wright State University

MODIFYING SOME ITERATIVE METHODS FOR SOLVING QUADRATIC EIGENVALUE PROBLEMS

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science

By

ALI HASAN ALI
B.Sc.Ed., University of Mosul, 2011

2017
Wright State University

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

December 13, 2017

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Ali Hasan Ali ENTITLED Modifying Some Iterative Methods for Solving Quadratic Eigenvalue Problems BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Sara Pollock, Ph.D.
Thesis Director

Ayse Sahin, Ph.D.
Chair, Department of
Mathematics and Statistics

Committee on
Final Examination

Sara Pollock, Ph.D.

Yuqing Chen, Ph.D.

Weifu Fang, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Ali, Ali Hasan. M.S. Department of Mathematics and Statistics, Wright State University, 2017. *Modifying Some Iterative Methods for Solving Quadratic Eigenvalue Problems.*

In this thesis, we are investigating the solutions λ of a typical quadratic eigenvalue problem (QEP). Indeed, solutions λ of a QEP of the form $Q(\lambda) = \lambda^2 M + \lambda D + S$ that satisfy $Q(\lambda) = 0$, can be obtained iteratively and without linearizing the problem. However, many iterative methods can only find some of the solutions λ. Therefore, we are going to modify a method based on Newton iterations in order to find all of the solutions λ, that are known also as the eigenvalues of the QEP. In addition, we will investigate how the proposed method compares with standard iterative methods from the literature. Moreover, we will provide a method for finding an upper bound for the number of the eigenvalues of the QEP, and apply this in our method for the purpose of finding all solutions λ.

List of Symbols and Acronyms

Introduction

QEP	Quadratic Eigenvalue Problem
NLEP	Nonlinear Eigenvalue Problem
MPP	Matrix Polynomial Problem
SMPP	Symmetric Matrix Polynomial Problem
MMPP	Monic Matrix Polynomial Problem
NMMPP	Nonmonic Matrix Polynomial Problem
SAMPP	Self-Adjoint Matrix Polynomial Problem
GEP	Generalized Eigenvalue Problem
SEP	Standard Eigenvalue Problem

Literature Review

JD	Jacobi-Davidson
SOAR	Second Order Arnoldi Method
$\mathcal{K}$	Krylov Subspace
THQEP	Tri-diagonal Hyperbolic Quadratic Eigenvalue Problem
EAI	Ehrlich-Aberth Iteration

Methodology

NTI	Newton Trace Iteration
NII	Newton Inverse Iteration
NMM	Newton Maehly Method
NMI	Newton Maehly Iteration
$\mathcal{C}$	Newton Correction

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgment

I would like to extend my thanks to Dr. Sara Pollock for advising my thesis with great encouragement, patience, and insightful guidance. Words cannot express my gratitude for what you have done for me including all the time and energy that you have dedicated during three semesters. I am grateful to you for all the extra work you have done to make this process painless. Thank you for always having a smile on your face!

Special thanks to the Higher Committee for Educational Development in Iraq (HCED) for giving me the opportunity to complete my study abroad. In fact, my study would not have been possible without the financial support of the HCED.

I would also like to say a special word of thanks to the committee members Dr. Yuqing Chen and Dr. Weifu Fang for their time, expertise, and guidance throughout this thesis.

Also, I would like to take this opportunity to express my gratefulness to everyone in the LEAP program at Wright State University for their wonderful ways of promoting and improving my English skills and for the amazing and unforgettable activities and experiences we had during my journey in learning English.

Finally, thanks to all my family and to all friends who supported me in this work and to everyone I did not mention above. Your effort is still being acknowledged and remembered here. Thanks a ton.

Dedication

This thesis is lovingly dedicated to my parents. Your support and prayers have sustained me throughout my life.

Introduction

1.1 Quadratic Eigenvalue Problem (QEP)

Quadratic eigenvalue problems (QEPs) arise in many applications, such as dynamic systems, building designs, and vibrating systems. Many other applications, such as perturbation and dynamic analysis are described in [22]. QEPs are known by many different names, like quadratic matrix polynomials, nonlinear eigenvalue problems, quadratic pencils, and matrix pencils. These different names are used to describe a variety of similar problems. However, the general name is the nonlinear eigenvalue problem that can be a QEP if we have three matrices and one scalar λ of degree two at most. In addition, it is called a linear matrix pencil when we linearize the QEP to a general eigenvalue problem of the form $(A - \lambda B)v = 0$.

The standard QEP takes the form

$$Q(\lambda)v \coloneqq (\lambda^2 M + \lambda D + S)v = 0, \tag{1.1}$$

where M (the mass matrix), D (the damping matrix), and S (the stiffness matrix) are $n \times n$ matrices. Those matrices can have different meaning in other applications like the fluid dynamic system and the vibrating system. The entire QEP can be completely solved by finding the scalar $\lambda \in \mathbb{C}$ and some nonzero vectors $v \in \mathbb{C}^n$ satisfying (1.1). The scalar λ is called the eigenvalue of the QEP, and the nonzero vector v is called the right eigenvector

of the QEP that corresponds to λ. In addition the nonzero eigenvector $u \in \mathbb{C}^n$ that satisfies

$$uQ(\lambda) = u(\lambda^2 M + \lambda D + S) = 0, \tag{1.2}$$

is called the left eigenvector that corresponds to λ.

1.2 Solving the QEP

Solving the QEP can be done by linearizing the entire problem, which is the simplest approach and the classical way of solving QEPs. In addition, it can be solved by dealing with the nonlinear system as we will see in the next chapters.

Linearizing the QEP of type (1.1) means we will deal with a $2n \times 2n$ system. In other words, the linearization of a QEP means doubling the entire problem to transform it into a generalized eigenvalue problem (GEP) in order to deal with it linearly.

Consider the QEP of type (1.1), and let $n = 8$. This means, M, D, and S are 8×8 matrices. Therefore, the QEP (1.1) can be written in the form

$$\begin{bmatrix} 0 & I \\ -S & -D \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \lambda \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \tag{1.3}$$

where 0 denotes to the 8×8 block of zeros, and I denotes to the 8×8 identity block. It is clear that (1.3) is a generalized eigenvalue problem of the form $Av = \lambda Bv$, where A and B are 16×16 matrices. Therefore, the problem (1.3) can be solved either by transforming it into a standard eigenvalue problem (A or B must be non-singular), or directly as a generalized eigenvalue problem. In either case, high dimensional problems can be solved numerically by an iterative method [24].

1.2.1 Linearization of QEPs

As we mentioned in section 1.2, the most common way for solving QEPs is the linearization method, which is transforming the QEP into a linear eigenvalue problem. It is worth mentioning that the representation (1.3) is not unique. In other words, we can reformulate the representation (1.3) by the following approaches since each is equivalent to the problem $(A - \lambda B)v = 0$.

Representation I :

This is basically from the formulation (1.3), which is the most common one.

$$\begin{bmatrix} 0 & I \\ -S & -D \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} = \lambda \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} . \tag{1.4}$$

Representation II :

This approach is gotten by multiplying the first row of (1.4) by D, and the second row by -1, which is called also the symmetric formulation[1].

$$\begin{bmatrix} 0 & S \\ S & D \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} = \lambda \begin{bmatrix} S & 0 \\ 0 & -M \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} . \tag{1.5}$$

Representation III :

Another approach can be formulated by swapping the vectors v_i in Equation (1.3).

$$\begin{bmatrix} I & 0 \\ 0 & -S \end{bmatrix} \begin{bmatrix} \lambda v \\ v \end{bmatrix} = \lambda \begin{bmatrix} 0 & I \\ M & D \end{bmatrix} \begin{bmatrix} \lambda v \\ v \end{bmatrix} . \tag{1.6}$$

[1] Having a symmetric formulation can make the problem scaled appropriately to be solved by some special methods instead of general methods[7].

Representation IV :

The representation III can be symmetrized by multiplying the first row by M.

$$\begin{bmatrix} M & 0 \\ 0 & -S \end{bmatrix} \begin{bmatrix} \lambda v \\ v \end{bmatrix} = \lambda \begin{bmatrix} 0 & M \\ M & D \end{bmatrix} \begin{bmatrix} \lambda v \\ v \end{bmatrix}. \tag{1.7}$$

1.2.2 Additional linearizations

Representation V [22]:

This linearization is also equivalent to Equation (1.3), where H_1 is any non-singular matrix.

$$\begin{bmatrix} 0 & H_1 \\ -S & -D \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} = \lambda \begin{bmatrix} H_1 & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix}. \tag{1.8}$$

Representation VI [22]:

Another approach is equivalent to the formulation (1.3), where H_2 is any non-singular matrix.

$$\begin{bmatrix} S & 0 \\ 0 & H_2 \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix} = \lambda \begin{bmatrix} -D & -M \\ H_2 & 0 \end{bmatrix} \begin{bmatrix} v \\ \lambda v \end{bmatrix}. \tag{1.9}$$

Representation VII [13]:

This linearization is valid if the matrix M is non-singular and well-conditioned. The idea of this linearization is to assume that $L_0 = M^{-1}S$ and $L_1 = M^{-1}D$, and we have I and 0 are the identity matrix and the null matrix respectively. The eigenvalues of the matrix C below are the roots of the QEP (1.1).

$$C = \begin{bmatrix} 0 & I \\ -L_0 & -L_1 \end{bmatrix}. \tag{1.10}$$

1.3 QEPs as nonlinear eigenvalue problems (NLEPs)

As mentioned earlier, QEPs are considered a class of NLEPs, where the unknowns are of degree at most two. In a general NLEP, some nonlinear functions of λ such as $\cos(\lambda)$, $\sin(\lambda)$, e^{λ}, $\ln(\lambda)$, $\sqrt[3]{\lambda}$, ...etc. can be placed instead of λ in Equation (1.1). In addition, placing a higher degree term of the same pattern in Equation (1.1) will change the class of the problem to a matrix polynomial problem (MPP) [13], which is more general than a QEP. MPPs can be linearized by the mentioned approach in (1.10), but in generalized form as shown below. Consider the MPP

$$\bar{M}(\lambda)v = (\lambda^k M_k + \cdots\cdots + \lambda M_1 + M_0)v = 0, \tag{1.11}$$

the roots of the MPP (1.11) are the eigenvalues of the matrix C below, where I and 0 are the identity and the null matrices, and $L_i = M_k^{-1}M_i, \;\; 0 \leq i \leq k$.

$$C = \begin{bmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & I \\ -L_0 & -L_1 & -L_2 & \cdots & -L_{k-1} \end{bmatrix}. \tag{1.12}$$

It is important to mention that (1.10) is a special case of (1.12) with $k = 2$. Furthermore, the eigenvector that corresponds to the eigenvalue λ of this type of problem has a special pattern, which is

$$\begin{bmatrix} v & \lambda v & \lambda^2 v & \cdots & \lambda^{k-1} v \end{bmatrix}^T. \tag{1.13}$$

The matrix (M_k), which is called the leading matrix should be nonsingular and well conditioned to follow this approach.

To illustrate the relation among QEPs and the other problems such as NLEP, MPP, GEP, ...etc., we created this graph to demonstrate the classification structure.

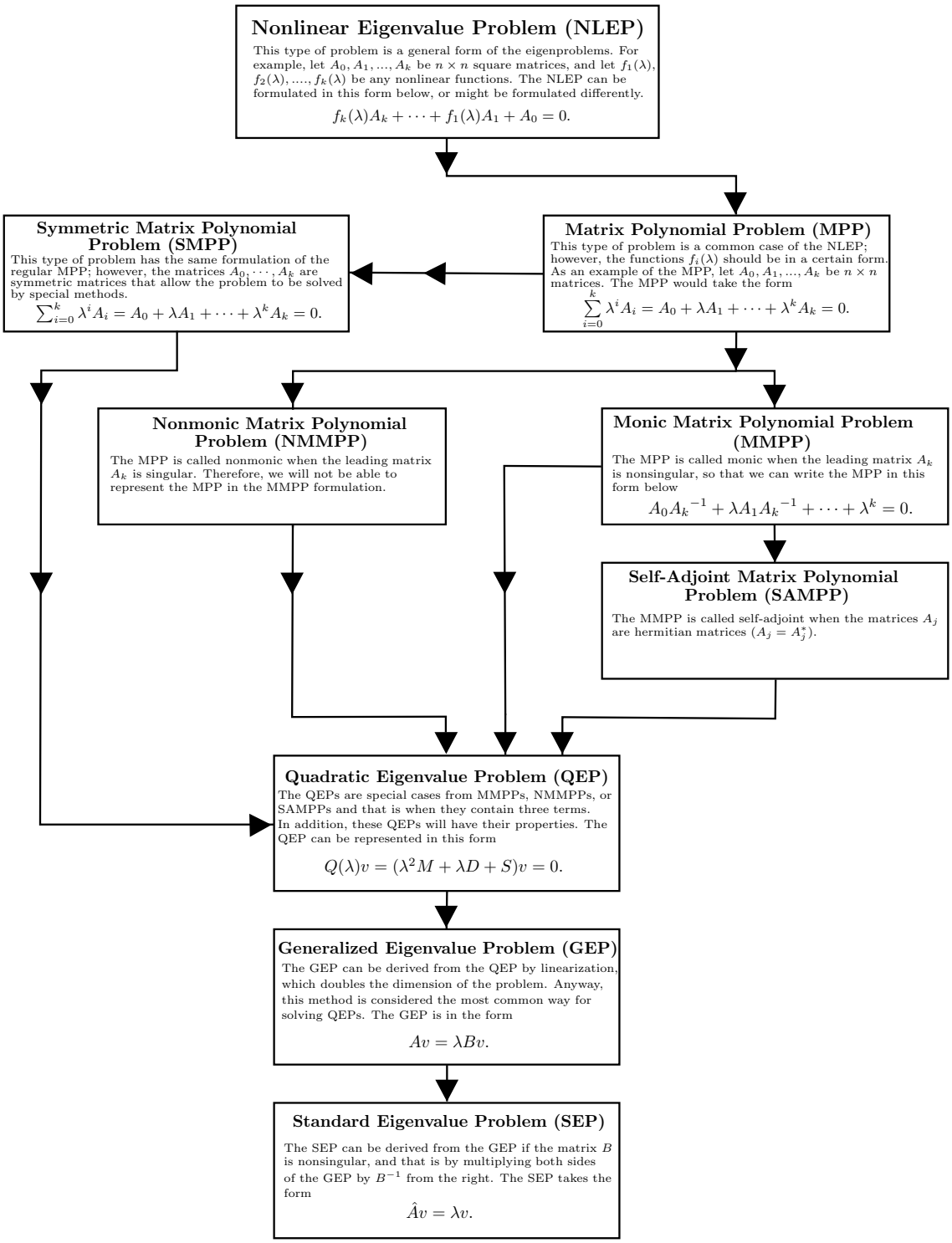

Figure 1.1: The classification of the nonlinear and linear eigenproblems.

Literature Review

2.1 Benefit of iterative methods

Linearizing QEPs does not always work, and might lead to a singular matrix in the GEPs. In other words, the singularity might appear in the linearized QEP as a result of certain choices of the matrices $M, D,$ and S in the original QEP. For instance, consider the example below.

Example 2.1. *Let* $M = \begin{bmatrix} 8 & 6 \\ 4 & 3 \end{bmatrix}$, $D = \begin{bmatrix} 2 & -1 \\ 22 & 4 \end{bmatrix}$, *and* $S = \begin{bmatrix} -2 & 7 \\ -9 & 11 \end{bmatrix}$. *Transforming the QEP of the form* (1.1) *into a GEP of the form* (1.3) *would give,*

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & -7 & -2 & 1 \\ 9 & -11 & -22 & -4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \lambda \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 8 & 6 \\ 0 & 0 & 4 & 3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

which is a GEP of the form $Av = \lambda Bv$, *with a singular matrix* B. *Therefore, it is not possible to transform it into a SEP by multiplying both sides by* B^{-1}. *Consequently, we have to use another way of solving the QEP, which is using the iterative methods.*

2.2 Types of iterative methods

Solving the QEP by iterative methods can be done either by directly dealing with the three matrices M, D, and S, or by transforming them into a GEP. Dealing with the QEP directly without transforming the problem into a GEP will reduce the computational cost and the time of getting the solution. In particular, avoiding the transformation will allow us to deal with $n \times n$ dimension instead of $2n \times 2n$. However, most of the methods that deal with QEPs directly are able to find only one eigenvalue at a time. In some cases, several of the eigenvalues can be recovered, but not all of them. On the other hand, the iterative methods that deal with GEPs are mostly able to recover all the eigenvalues of the QEPs that are the same eigenvalues of the GEPs. In the coming sections, we will describe some of these iterative methods.

2.3 Jacobi-Davidson method

The Jacobi-Davidson method is constructed to solve GEPs of the form $Av = \lambda Bv$. A significant results have been achieved by applying this method on QEPs that are from acoustic models [20]. The idea of this method is projecting QEPs onto some sub-spaces that have low dimensions. In other words, we will have projected QEPs of reasonable dimensions that can be solved by many methods after expanding the subspace by the correction equation of Jacobi-Davidson. Moreover, this method can be applied on MPPs of the form (1.11) and no linearization will be needed [6]. The projected QEP in Jacobi-Davidson takes the form

$$(\theta^2 V^* M V + \theta V^* D V + V^* S V)u = 0, \tag{2.1}$$

where V is an $n \times m$ orthonormal matrix, θ is the selected eigenvalue, and u is the eigenvector that is associated with θ. The algorithm of the Jacobi-Davidson method for QEPs is given below.

Algorithm 2.1: Jacobi-Davidson for a QEP of the form $\lambda^2 Q_2 + \lambda Q_1 + Q_0 = 0$.

Input : V is an orthonormal matrix of $n \times m$ dimension.

for *i=0, 1, 2* **do**

  calculate $R_i = Q_i V$ and $P_i = V^* R_i$

end

while *the convergence is not achieved,* **do**

  calculate the eigenvalue θ and the eigenvector u of $(\theta^2 P_2 + \theta P_1 + P_0)u = 0$.

  pick an eigenvalue θ with its associated eigenvector u where $\|u\|_2 = 1$.

  calculate $s = Vu$, $k = \Psi'(\theta)s$, $h = \Psi(\theta)s$.

  if $(\|h\|_2 < \epsilon), \lambda = \theta, x = s,$ **then**

    STOP

  else

    calculate $r \perp s$ from $\left(I - \dfrac{k\ s^*}{s^*\ k} \right) \Psi(\theta)(I - s\ s^*)r = -h$

    orthogonalize r against V, $b = \dfrac{r}{\|r\|_2}$

  end

  for *i=0, 1, 2* **do**

    calculate $\omega_i = Q_i b$

    $M_i = \begin{bmatrix} P_i & V^*\omega_i \\ b^* R_i & b^*\omega_i \end{bmatrix}$, $R_i = \begin{bmatrix} R_i & \omega_i \end{bmatrix}$

  end

  expand $V = \begin{bmatrix} V & b \end{bmatrix}$

end

2.4 The second order Arnoldi method (SOAR)

The main advantage of this method is dealing with large-scale QEPs by producing an orthonormal basis. This generated basis is based on a generalized Krylov subspace that is induced by two matrices K_1 and K_2 and a nonzero vector s. In general, the Krylov subspace method is highly beneficial when we deal with large-scale matrices approximately [1]. Indeed, most of the methods that are based on Krylov subspace are very efficient in generating orthonormal bases. The Arnoldi method and all other methods that are based on Krylov subspaces of the first order cannot be applied directly on QEPs. In fact, we need to go through two further steps before applying those methods. The first step is transforming the QEP into a GEP. The second step is reducing the GEP to a SEP where we assume that the leading matrix M in the QEP is non-singular. The disadvantage in these methods is the dimension is doubled in the first step. In contrast, the method of Jacobi-Davidson in section 2.3 does not need to go through the two mentioned steps. Instead, it deals directly with the QEP by projecting it to a low-dimensional projected problem [1]. The Krylov subspace is based on a square matrix K and a nonzero vector s, and it is defined as

$$\mathcal{K}(K; s) = \text{span}\{s, Ks, K^2 s, \ldots, K^{q-1} s\}, \tag{2.2}$$

where $K \in \mathbb{R}^{n\times n}$, and $s \in \mathbb{R}^n$ is called the starting vector. Moreover, the second order Krylov subspace is defined as

$$\mathcal{K}(K_1, K_2; s) = \text{span}\{r_0, r_1 \ldots, r_{q-1}\}, \tag{2.3}$$

where $r_0 = s, \quad r_1 = K_1 s, \quad r_i = K_1 r_{i-1} + K_2 r_{i-2}, \quad K_1, K_2 \in \mathbb{R}^{n\times n}$, and $s \in \mathbb{R}^n$.

To avoid the steps of transforming the QEP into a SEP, we need to have a second order Krylov subspace as well as the orthogonal projection technique of Rayleigh Ritz which allows us to apply the SOAR directly on the QEPs. The algorithm of the improved SOAR

method with deflation is given below. More SOAR algorithms are available in [1].

Algorithm 2.2: SOAR method for QEPs.

$y_1 = s/\|s\|_2$
$x_1 = 0$
for *j=1, 2. . . , n* **do**
  $r = K_1 y_j + K_2 x_j$
  $h = y_j$
  for *i=1, 2, . . . , j* **do**
    $b_{ij} = y_i^T r$
    $r := r - y_i b_{ij}$
    $h := h - x_i b_{ij}$
  end
  $b_{j+1\ j} = \|r\|_2$
  if $b_{j+1\ j} = 0$ **then**
    if $h \in \text{span}\{x_i \mid i :\ y_i = 0, 1 \le i \le j\}$ **then**
      STOP
    else *% deflation*
      reset $b_{j+1\ j} = 1$
      $y_{j+1} = 0$
      $x_{j+1} = h$
    end
  else *% regular case*
    $y_{j+1} = r/b_{j+1\ j}$
    $x_{j+1} = h/b_{j+1\ j}$
  end
end

2.5 Methods for tri-diagonal hyperbolic QEPs (THQEPs)

The tri-diagonal QEP is a special type of QEP of the form (1.1). In this type of QEP, the three matrices M, D, and S are tri-diagonal. The QEP (1.1) is called hyperbolic if the matrix M is positive definite and

$$(v^*Dv)^2 > 4(v^*Mv)(v^*Sv). \tag{2.4}$$

If the matrices M, D, and S in (1.1) are tri-diagonal and symmetric, M is positive definite, and (2.4) holds, then we call (1.1) a tri-diagonal hyperbolic QEP (THQEP) [17]. The two methods below focus on this type of QEP.

2.5.1 Ehrlich-Aberth iteration (EAI)

The method of Ehrlich-Aberth was first developed in 1967 by Louis W. Ehrlich and Oliver Aberth for finding zeros of regular polynomials. Later, many attempts have been made to modify this method in order to work on different classes of polynomials such as QEPs and NLEPs. In fact, a great modification was done by Dario A. Bini, and Vanni Noferini in [5]. They were able to make the method of EAI able to deal not only with THQEP, but also with NLEPs with significant accuracy and efficiency. However, their method is way technical, so we are going to illustrate here the one that deals with only THQEPs.

The Ehrlich-Aberth iteration takes an initial approximation $u^{(0)} \in \mathbb{C}^{2n}$ and creates a sequence $u^{(k)} \in \mathbb{C}^{2n}$ that will eventually approach the eigenvalue of the THQEP. The form of the EAI for THQEP is given by

$$u_k^{(m+1)} = u_k^{(m)} - \frac{\mathcal{N}\left(u_k^{(m)}\right)}{1 - \mathcal{N}\left(u_k^{(m)}\right) \sum\limits_{\substack{i=1 \\ i \neq k}}^{2n} \frac{1}{u_k^{(m)} - u_i^{(m)}}} \tag{2.5}$$

for $i = 1, \ldots, 2n$. In addition, there is another formulation of EAI in the style of Gauss-Seidel that will result in giving a cubic convergence rate and also is slightly faster than (2.5) [5]. This formulation is given by

$$u_k^{(m+1)} = u_k^{(m)} - \frac{\mathcal{N}\left(f(u_k^{(m)})\right)}{1 - \mathcal{N}\left(f(u_k^{(m)})\right)\left(\sum_{i=1}^{k-1} \frac{1}{u_k^{(m)} - u_i^{(m+1)}} + \sum_{i=k+1}^{2n} \frac{1}{u_k^{(m)} - u_i^{(m)}}\right)}, \tag{2.6}$$

where $\mathcal{N}(u) = f(u)/f'(u)$ is called the Newton correction, and $f(\lambda) = \det(Q(\lambda))$ is called the scalar polynomial.

2.5.2 Durand-Kerner method

This method is similar to the EAI method in the way of approaching a single eigenvalue. More specifically, the Durand-Kerner method generates a sequence $u^{(k)} \in \mathbb{C}^{2n}$ that will eventually converge to the eigenvalue of the THQEP. However, the scalar polynomials of this method and the EAI method are not the same. The equation of this method has different formulation[1] from EAI method and it is given by

$$u_k^{(m+1)} = u_k^{(m)} - \frac{g(u_k^{(m)})}{\prod_{\substack{i=1 \\ i \neq k}}^{2n} (u_k^{(m)} - u_i^{(m)})}, \tag{2.7}$$

with the scalar polynomial

$$g(\lambda) = \frac{1}{\det(M)} \det(Q(\lambda)). \tag{2.8}$$

[1] More formulations, such as Jacobi style and Gauss-Seidel style for better and faster convergence are available in [17].

Methodology

3.1 Properties of QEPs eigenvalues

Determining the number and the properties of the problem's eigenvalues, is very important to increase the efficiency of the related iterative methods. For instance, the number of the finite eigenvalues of the QEP (1.1) is at most $2n$. In other words, for certain choice of the matrices $M, D,$ and S we might get less than $2n$ eigenvalues for the entire problem. Therefore, doing $2n$ computations by using methods that can find only one eigenvalue at each time is not a good idea. On the other hand, if the leading matrix M is singular, we will surely get some infinite eigenvalues. The table below illustrates some situations of different choices for the matrices $M, D,$ and S [22].

Table 3.1: The eigenvalues properties of the QEP $Q(\lambda)v := (\lambda^2 M + \lambda D + S)v = 0$.

Matrices	**Eigenvalues**
The leading matrix M is non-singular	The number of finite eigenvalues $= 2n$
The leading matrix M is singular	The number of finite eigenvalues $< 2n$
$M = M^*$ and is positive definite, D, S are positive semi-definite and Hermitian	The real part of the eigenvalues ≤ 0
The matrices $M, D,$ and S are real or Hermitian	Real eigenvalues or come in the form $(\lambda, \bar{\lambda})$
M, D are positive definite and symmetric, S is positive semi-definite and symmetric, $\min\{(v^*Dv)^2 - 4(v^*Mv)(v^*Sv)\} > 0$	All the eigenvalues are real and negative, and there is a gap between the half largest and the half smallest eigenvalues
$M = M^*$ and is positive definite, $D = -D^*$, $S = S^*$	All the eigenvalues are pure imaginary, or come in the form $(\lambda, -\bar{\lambda})$
M, S are positive definite, symmetric, and real, $D = -D^T$	All the eigenvalues are pure imaginary

Example 3.1. *Consider the QEP*

$$Q(\lambda) = \lambda^2 M + \lambda D + S = 0, \tag{3.1}$$

where the matrices M, D, and S are given as follows

$$M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 4 \end{bmatrix}, \; D = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, S = \begin{bmatrix} 1 & 2 & 2 \\ 2 & -1 & 2 \\ 2 & 2 & 1 \end{bmatrix}.$$

It is clear that the matrices above have the following properties

- *The matrix M is nonsingular, positive definite, and Hermitian $(M = M^*)$.*
- *The matrix D is equal to its own negative conjugate transpose $(D = -D^*)$.*
- *The matrix S is Hermitian $(S = S^*)$.*

Therefore, we expect to get six finite eigenvalues that are pure imaginary or have the form $(\lambda, -\bar{\lambda})$ as mentioned in Table 3.1. In fact, some of the eigenvalues of the QEP (3.1) are pure imaginary and some are in the form $(\lambda, -\bar{\lambda})$ as shown in Table 3.2.

Table 3.2: The eigenvalues of the QEP (3.1).

λ	$-\bar{\lambda}$ **or** $-\lambda$
$0.0000 + 1.6055i$	$-0.0000 - 1.6055i$
$0.5731 + 0.0000i$	$-0.5731 + 0.0000i$
$1.2745 + 0.0000i$	$-1.2745 + 0.0000i$

3.2 Determining the number of eigenvalues

The number of a QEP's eigenvalues can be determined if we know the degree of the polynomial in the QEP's determinant. Calculating the numerical determinants can be done by applying many mathematical methods. Some of these methods are valid to be applied on the symbolic determinant. However, the determinants that contain polynomials as components have some different issues to deal with. More specifically, we are dealing here with components that contain both numeric and symbolic entries. The determinant can contain different combinations of numerical and symbolic entries. In other words, it can contain polynomials of variant degrees, constants, and zeros. The matrix below is an example of a matrix of polynomials.

$$C = \begin{bmatrix} x^2 - 26 & x^2 - 3x & -3x & \cdots & 5 \\ 4x & 7x^3 - 4x^2 + 2 & 43 & \cdots & x - 3 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ -x^3 & 0 & x - 4 & \cdots & x^3 - 5 \\ -x + 2 & x^2 + 2 & x^4 + 3x & \cdots & x^5 \end{bmatrix}. \tag{3.2}$$

The methods of finding the determinant of the matrix above differ in terms of efficiency and complexity. Many studies have been done in order to get a simple and fast way of calculating these types of determinants. Expanding these types of determinants can be done by the interpolation technique [8], which is finding the degree of every term, then finding their coefficients. In addition, it can be expanded by the regular way of determinant calculation that requires a high computational cost compared with the previous mentioned method. However, the regular way of finding the determinant can be applied directly and done easily if we have some zero components in certain positions in the original matrix.

3.2.1 Calculating the degree of polynomial determinants

As mentioned in section 3.2, the degree of a polynomial is a good indication of the number of roots, or the eigenvalues when we deal with QEPs. The method of calculating the degree of a polynomial determinant we are presenting here can be applied on any $n \times n$ determinant. In addition, determinants with multivariate components can be calculated by this method each variable separately. In other words, the method can provide the degree of each variable in the polynomial determinant. It is worth mentioning that this method does not always give the exact degree of the polynomial determinant. Sometimes it only gives an upper bound of the exact degree [19].

The first step in this method is transforming the symbolic matrix we have to a numerical matrix by placing the variable degree in place of the component that has the same position in the matrix.

$$C = \begin{bmatrix} 3t^3 & -4t & 5 & 2t \\ -t^2 & 6t^3 & 22 & 8t^2 \\ 11t^2 & 9 & t^2 & 32t^4 \\ -3t & 5t & 3 & 7t^2 \end{bmatrix} \longrightarrow N = \begin{bmatrix} 3 & 1 & 0 & 1 \\ 2 & 3 & 0 & 2 \\ 2 & 0 & 2 & 4 \\ 1 & 1 & 0 & 2 \end{bmatrix}. \tag{3.3}$$

After getting the numerical matrix N, we start re-indexing the $n \times n$ matrix by assigning a new representation N_0 with indices running from 0 to $n-1$. That is, the component $N(1,1)$ will be the component $N_0(0,0)$. Next, we form the matrix N_1, which is one less column and row than N_0, and so on until we reach the 1×1 matrix N_{n-1} which is the initial degree of the polynomial determinant. The last step is unwinding the initial degree when the original matrix dimension is more than 2×2. A detailed explanation is provided in Algorithm 3.1.

Algorithm 3.1: Calculating the degree of polynomial determinants.

Input : C is a symbolic matrix of $n \times n$ dimension.

Transform the symbolic matrix C to a numerical matrix N

Set $N_0 = N$, where N_0 is indexed from 0 to $n-1$

if $n = 1$ **then**

  set the MaxDegree $= N_0(0,0)$

  STOP

end

if $n = 2$ **then**

  set the MaxDegree $N_1 = \max\Big(N_0(1,1) + N_0(0,0), N_0(1,0) + N_0(0,1)\Big)$

  STOP

end

for *i=1:n-1* **do**

  for *j=1:n-1* **do**

    $N_k(i,j) = \max\Big(N_{k-1}(i,j) + N_{k-1}(0,0), N_{k-1}(i,0) + N_{k-1}(0,j)\Big)$

  end

end

MaxDegree $= N_{n-1}$

for *i=1:n-2* **do**

  MaxDegree $=$ MaxDegree $- i * N_{n-2-i}(0,0)$

end

$N_{n-1} =$ MaxDegree

Output: N_{n-1} is the maximum degree.

Example 3.2. *Consider the symbolic matrix C and its numerical matrix N in (3.3). Applying Algorithm 3.1 would give:*

$$N_0 = \begin{bmatrix} 3 & 1 & 0 & 1 \\ 2 & 3 & 0 & 2 \\ 2 & 0 & 2 & 4 \\ 1 & 1 & 0 & 2 \end{bmatrix} \longrightarrow N_1 = \begin{bmatrix} \max(3+3, 2+1) & \max(0+3, 2+0) & \max(2+3, 2+1) \\ \max(0+3, 2+1) & \max(2+3, 2+0) & \max(4+3, 2+1) \\ \max(1+3, 1+1) & \max(0+3, 1+0) & \max(2+3, 1+1) \end{bmatrix}$$

$$N_1 = \begin{bmatrix} 6 & 3 & 5 \\ 3 & 5 & 7 \\ 4 & 3 & 5 \end{bmatrix} \longrightarrow N_2 = \begin{bmatrix} \max(5+6, 3+3) & \max(7+6, 3+5) \\ \max(3+6, 4+3) & \max(5+6, 4+5) \end{bmatrix}$$

$$N_2 = \begin{bmatrix} 11 & 13 \\ 9 & 11 \end{bmatrix} \longrightarrow N_3 = \max\Big(11+11, 9+13\Big) \longrightarrow \boxed{N_3 = 22}$$

Now, $MaxDegree = N_3 - N_1(1,1) - 2N_0(1,1) = 22 - 6 - 2 \times 3 = \boxed{10}$. This tells us that the degree of the polynomial should not be more than 10. In fact, the polynomial degree is exactly 10 as shown below.

$$\Big|C\Big| = -1602t^{10} + 7560t^8 - 1118t^7 - 7966t^6 - 7030t^5 - 2735t^4 + 1134t^3 - 1188t^2$$

.

To illustrate how this method works with more than one variable, we consider the three variable matrix that is shown in Example 3.3.

Example 3.3. *Let* $C = \begin{bmatrix} x+y & xz & x-1 & y^2 \\ x^3+yz & 2z^2 & x & 22x^2 \\ x^2+2x-1 & 2xy & 0 & 1-z^3 \\ y-z & -3 & x^2-9 & x+y^3-z^2 \end{bmatrix}$.

Applying Algorithm 3.1 on the three variables $x, y,$ *and* z *would give:*

$$\textit{Solving for } x \longrightarrow \begin{bmatrix} 1 & 1 & 1 & 0 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 4 & 4 & 3 \\ 3 & 3 & 2 \\ 1 & 3 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 7 & 6 \\ 7 & 6 \end{bmatrix} \longrightarrow 13 \longrightarrow \boxed{7}.$$

$$\textit{Solving for } y \longrightarrow \begin{bmatrix} 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 3 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 3 \\ 2 & 1 & 2 \\ 1 & 1 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 5 \\ 2 & 5 \end{bmatrix} \longrightarrow 8 \longrightarrow \boxed{5}.$$

$$\textit{Solving for } z \longrightarrow \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 1 & 0 & 0 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 1 & 1 \\ 1 & 0 & 3 \\ 2 & 1 & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 5 \\ 3 & 4 \end{bmatrix} \longrightarrow 8 \longrightarrow \boxed{6}.$$

Calculating the exact determinant[1] *of* C *would give:*

$$-22x^7z + \cdots + 2x^2y^5z - 2x^2y^5 - 2xy^5z + \cdots + 2xz^6 - 2z^6 + \ldots.$$

[1]The exact determinant was found by using **(GNU Octave)** [9].

3.3 QEP applications and real-life models

A great deal of work collecting NLEP models has been done in [4]. The authors were able to collect more than 50 models of real-life problems. In addition, more than 40 of those problems are QEPs. The purpose of mentioning these models is going to be discussed in detail in the next chapter, in what those models are used to test the methods of solving QEPs. Here are some notable examples of models using QEPs.

3.3.1 The spring problem

The illustrated graph in Figure 3.1 represents a connected system of damped masses and springs. In this system, m_1 to m_n are the masses that are joined together by dampers with constants d_1 to d_{n-1} and springs with constants s_1 to s_{n-1}. In addition, the dampers τ_1 to τ_n and the springs κ_1 to κ_n connect the masses m_1 to m_n to the ground.

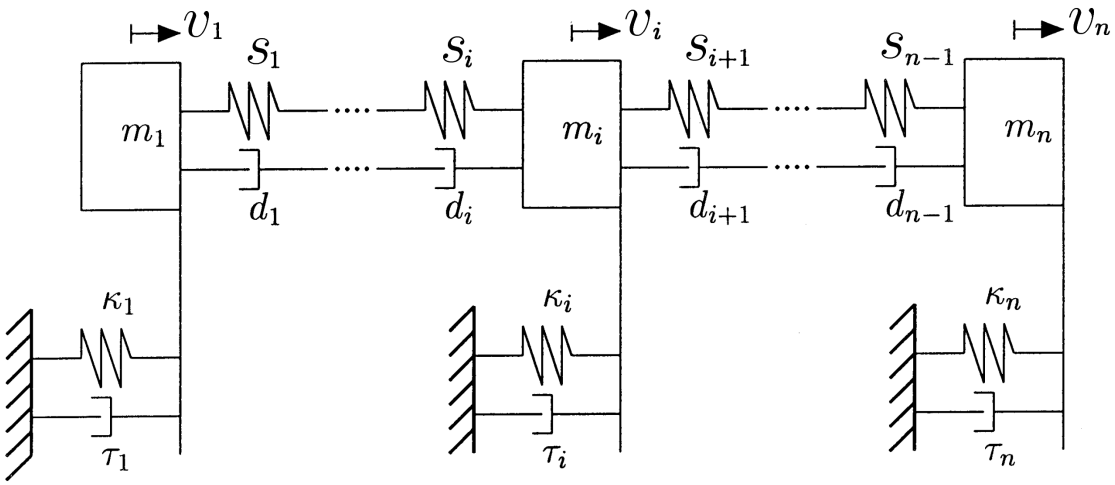

Figure 3.1: A damped mass-spring system of degree n [21].

The QEP in this system is the differential equation of second order that governs the system's vibration. The differential equation of this system is defined as

$$M\frac{d^2}{dt^2}v + D\frac{d}{dt}v + Sv = 0, \tag{3.4}$$

where the $n \times n$ mass matrix M, damping matrix D, and the stiffness matrix S as given in [4] are:

$$M = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}, D = \begin{bmatrix} 30 & -10 & & \\ -10 & \ddots & \ddots & \\ & \ddots & \ddots & -10 \\ & & -10 & 30 \end{bmatrix}, S = \begin{bmatrix} 15 & -5 & & \\ -5 & \ddots & \ddots & \\ & \ddots & \ddots & -5 \\ & & -5 & 15 \end{bmatrix}.$$

3.3.2 The bicycle problem

The motion of the steer and the lean angles δ and ϕ, of a bicycle, can be represented together as a second order differential equation with the coefficients M as the mass matrix, $D = uD_0$ as the damping matrix, and $S = gS_1 + u^2S_2$ as the stiffness matrix.

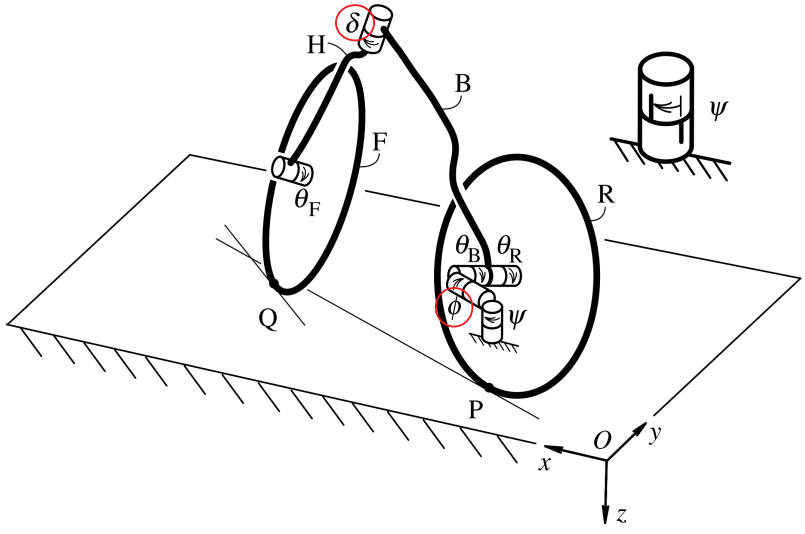

Figure 3.2: The steer and the lean angles in the bicycle problem [15].

The equation of the bicycle model can be written as

$$\lambda^2 Mv + \lambda u D_0 v + (gS_1 + u^2 S_2)v = f, \tag{3.5}$$

where $v = [\phi, \delta]^T$ and $f = [T_\phi, T_\delta]^T$ are time-dependent quantities, u is the forward speed, and g is the gravitational acceleration. In addition, the values of the coefficient matrices are given in [4] as:

$$M = \begin{bmatrix} -794.1195 & 1889.4323 \\ -25.5012 & 58.4775 \end{bmatrix}, D = \begin{bmatrix} 0 & 169.332 \\ -4.2517 & 8.427 \end{bmatrix}, S = \begin{bmatrix} 80.8172 & 2.3194 \\ 2.3194 & 0.2978 \end{bmatrix}.$$

3.3.3 The population of bilbies problem

A QEP arises in a quasi-birth-death model for the population of the bilby, which is an endangered animal that is shown in Figure 3.3. In this problem, (i, j) is considered as the state, where i indicates the population, and $j - 1$ takes the values 0 to 4 and indicates the number of the previous bad seasons. For example, $j = 1$ means the previous season was not a bad season. In addition, g indicates the probability of having a good season, u_j and d_j are the probabilities of the population increasing and decreasing, respectively.

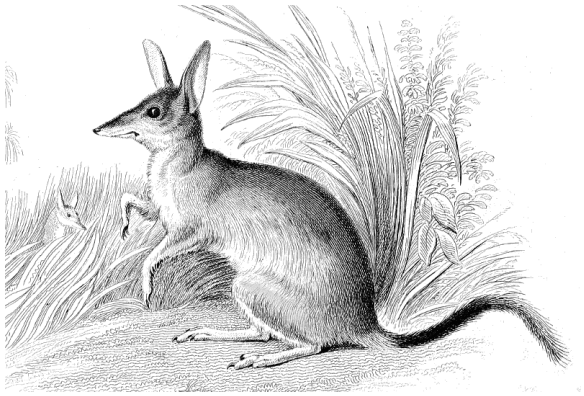

Figure 3.3: The Australian bilby.

Define k_j by $k_j = 1 - u_j - d_j$, and define the matrix C by

$$C(g,z) = \begin{bmatrix} gz_1 & (1-g)z_1 & 0 & 0 & 0 \\ gz_2 & 0 & (1-g)z_2 & 0 & 0 \\ gz_3 & 0 & 0 & (1-g)z_3 & 0 \\ gz_4 & 0 & 0 & 0 & (1-g)z_4 \\ gz_5 & 0 & 0 & 0 & (1-g)z_5 \end{bmatrix},$$

then the corresponding matrices $M, D,$ and S of the QEP of this model can be written as

$$M = vH_2{}^T,\ D = vH_1{}^T - I,\ S = vH_0{}^T,$$

where

$$H_0 = C(g,u),\ \ H_1 = C(g,k),\ \ H_2 = C(g,d).$$

Taking the same values as in [3], gives

$$M = \begin{bmatrix} 0.1 & 0.04 & 0.025 & 0.01 & 0 \\ 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0.16 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0.04 & 0 \end{bmatrix},\ D = \begin{bmatrix} -1 & 0.01 & 0.02 & 0.01 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0.04 & -1 & 0 & 0 \\ 0 & 0 & 0.08 & -1 & 0 \\ 0 & 0 & 0 & 0.04 & -1 \end{bmatrix},$$

$$S = \begin{bmatrix} 0 & 0.05 & 0.055 & 0.08 & 0.1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0.22 & 0 & 0 \\ 0 & 0 & 0 & 0.32 & 0.4 \end{bmatrix}.$$

3.3.4 QEPs with singular leading matrix

In the following two acoustic-modeling problems from [22], the 3×3 leading matrices are singular. Therefore, from Table 3.1 we will have less than 6 finite eigenvalues in each problem.

3.3.4.1 Problem 1

$$Q(\lambda)v = (\lambda^2 M + \lambda D + S)v = 0, \tag{3.6}$$

where

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \; D = \begin{bmatrix} -2 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

3.3.4.2 Problem 2

$$Q(\lambda)v = (\lambda^2 M + \lambda D + S)v = 0, \tag{3.7}$$

where

$$M = \begin{bmatrix} 0 & 6 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; D = \begin{bmatrix} 1 & -6 & 0 \\ 2 & -7 & 0 \\ 0 & 0 & 0 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

3.4 Numerical methods based on Newton iteration

Newton iteration methods play an important role in solving nonlinear problems numerically. In addition, they provide accurate and efficient approximations most of the time. Unlike the mentioned methods in Chapter 2, Newton methods need an initial guess at the beginning. This initial guess is very important in determining the efficiency and the convergence properties of the solution.

Suppose that we have the polynomial $P(\lambda) = 0$, and the initial guess for the root λ is $\lambda^{(0)}$, then the Newton iteration method for finding the solution λ is given as

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{P(\lambda^{(i)})}{P'(\lambda^{(i)})}, \quad i = 0, 1, \ldots. \tag{3.8}$$

Many methods from [23] based on Equation (3.8) have been modified in [11] in order to make them deal with NLEPs and QEPs. However, the modified methods cannot find all the solutions λ, or the eigenvalues of QEPs. These methods also differ in terms of accuracy and efficiency as we will see in the next chapter. Before we start listing Newton methods, it is important to mention that Equation (3.9) below is equivalent to Equation (1.1).

$$Q(\lambda)v = \begin{bmatrix} \lambda^2 m_{11} + \lambda d_{11} + s_{11} & \ldots\ldots & \lambda^2 m_{1n} + \lambda d_{1n} + s_{1n} \\ \lambda^2 m_{21} + \lambda d_{21} + s_{21} & \ldots\ldots & \lambda^2 m_{2n} + \lambda d_{2n} + s_{2n} \\ \vdots & \ddots & \vdots \\ \lambda^2 m_{n1} + \lambda d_{n1} + s_{n1} & \ldots\ldots & \lambda^2 m_{nn} + \lambda d_{nn} + s_{nn} \end{bmatrix} v = 0 \tag{3.9}$$

$Q(\lambda)$ is called the polynomial matrix of the QEP, $P(\lambda) = \det(Q(\lambda))$ is the scalar polynomial of the QEP.

3.4.1 Newton-trace iteration (NTI)

Since the polynomial matrix $Q(\lambda)$ in (3.9) has only three types of components, and those components are differentiable functions of λ, then we will have

$$P'(\lambda) = P(\lambda)\,\text{trace}\Big(Q^{-1}(\lambda)Q'(\lambda)\Big). \tag{3.10}$$

Therefore, according to Theorem 5.1 in [12], Equation (3.8) can be rewritten as

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{1}{\text{trace}\Big(Q^{-1}(\lambda^{(i)})Q'(\lambda^{(i)})\Big)}, \quad i = 0, 1, \ldots. \tag{3.11}$$

Equation (3.11) is called Newton-trace iteration (NTI), and the algorithm is given as:

Algorithm 3.2: Newton-trace algorithm [12].

Input : $n \times n$ polynomial matrix $Q(\lambda)$, initial guess $\lambda^{(0)}$, iterations number k .

Calculate $Q'(\lambda)$

for *i=1:k* **do**

  LU decomposition for $Q(\lambda^{(i)})$

  if $\dfrac{\text{abs}\Big(\prod_{j=1}^{n} u_{jj}\Big)}{\|Q(\lambda^{(i)})\|_F} < \epsilon$ **then**

    STOP

  end

  Solve $LX = Q'(\lambda^{(i)})$ for X

  Solve $UY = X$ for Y

  $\lambda^{(i+1)} = \lambda^{(i)} - \dfrac{1}{\text{trace}(Y)}$

end

Output: eigenvalue λ.

3.4.2 Newton inverse iteration (NII)

The Newton inverse iteration can be applied directly to the QEPs, without need to compute the scalar polynomial $\det(Q(\lambda))$. A unit-normalized vector v is needed in this method to initialize the eigenvector approximation. In addition, a nonzero vector u is needed to initialize the iteration for $\lambda^{(i+1)}$ as shown in (3.12)

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{u^T v^{(i)}}{u^T v^{(i+1)}}. \tag{3.12}$$

The choice of the vector u can affect the convergence properties of the eigenvalue λ, and there are various ways as discussed in [11] to choose the vector u. Moreover, the normalization of the vector v is very important to prevent any numerical underflow or overflow. The steps of this method are shown in Algorithm 3.3 below.

Algorithm 3.3: Newton inverse iteration [11].

Input : polynomial matrix $Q(\lambda)$, initial guess $\lambda^{(0)}$, iterations number k, initial normalized vector $v^{(0)}$, nonzero vector u .

for *i=1:k* **do**
  if $\left\|Q(\lambda^{(i)})v^{(i)}\right\|_2 < \epsilon$ **then**
    STOP
  end
  Solve $Q(\lambda^{(i)})v^{(i+1)} = Q'(\lambda^{(i)})v^{(i)}$ for $v^{(i+1)}$
  $\lambda^{(i+1)} = \lambda^{(i)} - \dfrac{u^T v^{(i)}}{u^T v^{(i+1)}}$
  Normalize $v^{(i+1)} = \dfrac{v^{(i+1)}}{\left\|v^{(i+1)}\right\|_2}$
end

Output: eigenvalue λ.

3.4.3 Newton Maehly method (NMM)

Approaching a root of a polynomial $P(\lambda)$ from a specific point can be done by drawing a tangent from an initial point P_1 through the graph until touching the X axis at the point X_2. Next, starting from the point X_2 that we ended up in and going back to the point P_2 in the graph again as shown in Figure 3.4. Repeating the first step by drawing another tangent from P_2 to X_3, and the second step by going back again to the graph until we have a very small difference between two of the previous mentioned iterations [16].

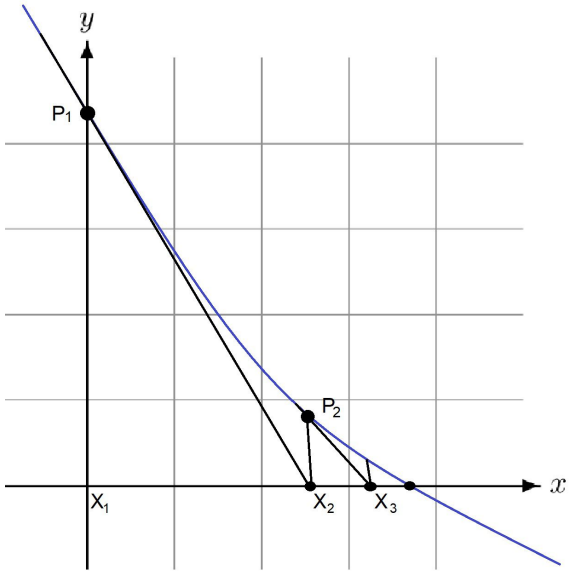

Figure 3.4: Newton iteration technique.

Mathematically, the mentioned steps can be interpreted as:

$$P(\lambda_1) = (\lambda_1 - \lambda_2)P'(\lambda_1), \tag{3.13}$$

or

$$\lambda_2 = \lambda_1 - \frac{P(\lambda)}{P'(\lambda)}, \tag{3.14}$$

which is the Newton iteration mentioned previously in Equation (3.8).

Using Equation (3.14) will result in finding only one root depending on the chosen initial value $\lambda^{(0)}$. Therefore, a significant modification has been done in [2] in order to find all the roots of a given polynomial. To illustrate the idea of NMM in [2], we consider a polynomial of three roots α_1, α_2, and α_3. Let

$$P_3(\lambda) = (\lambda - \alpha_1)(\lambda - \alpha_2)(\lambda - \alpha_3). \tag{3.15}$$

Differentiating (3.15) would give

$$P'_3(\lambda) = (\lambda - \alpha_1)(\lambda - \alpha_2) + (\lambda - \alpha_1)(\lambda - \alpha_3) + (\lambda - \alpha_2)(\lambda - \alpha_3). \tag{3.16}$$

The fraction $\frac{P'_3(\lambda)}{P_3(\lambda)}$ becomes

$$\frac{P'_3(\lambda)}{P_3(\lambda)} = \frac{(\lambda - \alpha_1)(\lambda - \alpha_2) + (\lambda - \alpha_1)(\lambda - \alpha_3) + (\lambda - \alpha_2)(\lambda - \alpha_3)}{(\lambda - \alpha_1)(\lambda - \alpha_2)(\lambda - \alpha_3)}, \tag{3.17}$$

$$\Rightarrow \frac{P'_3(\lambda)}{P_3(\lambda)} = \frac{1}{(\lambda - \alpha_1)} + \frac{1}{(\lambda - \alpha_2)} + \frac{1}{(\lambda - \alpha_3)} = \sum_{j=1}^{3} \frac{1}{(\lambda - \alpha_j)}, \tag{3.18}$$

and Equation (3.8) can be rewritten as

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{1}{\frac{P'(\lambda^{(i)})}{P(\lambda^{(i)})}} = \lambda^{(i)} - \frac{1}{\sum_{j=1}^{k} \frac{1}{(\lambda^{(i)} - \alpha_j)}}, \quad i = 0, 1, \ldots. \tag{3.19}$$

To avoid evaluating the recomputed roots, the modified formula by Maehly [2] that is given below is used.

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{P_{n-k}(\lambda^{(i)})}{P'_{n-k}(\lambda^{(i)})}, \tag{3.20}$$

where

$$P_{n-k}(\lambda) = \frac{P_n(\lambda)}{(\lambda - \alpha_1)(\lambda - \alpha_2) \ldots \ldots (\lambda - \alpha_k)}, \tag{3.21}$$

and the derivative $P'_{n-k}(\lambda)$ is

$$P'_{n-k}(\lambda) = \frac{P'_n(\lambda)}{(\lambda - \alpha_1) \ldots (\lambda - \alpha_k)} - \frac{P_n(\lambda)}{(\lambda - \alpha_1) \ldots (\lambda - \alpha_k)} \sum_{j=1}^{k} \frac{1}{(\lambda - \alpha_j)}. \tag{3.22}$$

Therefore, Equation (3.20) can be rewritten as

$$\lambda^{(i+1)} = \lambda^{(i)} - \frac{1}{\frac{P'(\lambda^{(i)})}{P(\lambda^{(i)})} - \sum_{j=1}^{k} \frac{1}{(\lambda^{(i)} - \alpha_j)}}. \tag{3.23}$$

Equation (3.23) is called Newton Maehly iteration (NMI), which finds a new root and suppresses all the computed roots. In other words, computing a new root with an implicit deflation of the evaluated roots.

3.4.3.1 QEPs by NMM

NMM can be used to evaluate QEPs by evaluating the determinant of the matrix in Equation (3.9) without explicitly computing the scalar polynomial. The Newton correction $\frac{P(\lambda)}{P'(\lambda)}$ is needed in order to progress the iteration of finding every root. To avoid the overflow of evaluating the determinants $P(\lambda) = \det(Q(\lambda))$ and $P'(\lambda) = \det(Q'(\lambda))$ for a given λ during the calculation, it is better to use Gaussian elimination and initialize $\sigma := \log(P(\lambda))$ to compute the Newton correction $\frac{P(\lambda)}{P'(\lambda)}$ as follows.

Deriving $\log(P(\lambda))$

$$\sigma' := \frac{d}{d\lambda} \log(P(\lambda)) = \frac{P'(\lambda)}{P(\lambda)}. \tag{3.24}$$

Equation (3.24) is equivalent to the inverse of the Newton correction $\frac{P(\lambda)}{P'(\lambda)}$. Therefore, it is better to update the steps of evaluating the determinant as below [10].

$$\sigma = \sigma + \log(q_{ii}) \tag{3.25}$$

instead of

$$P(\lambda) = P(\lambda) \times q_{ii}, \tag{3.26}$$

or even better to compute

$$\sigma' = \sigma' + \frac{q'_{ii}}{q_{ii}}, \tag{3.27}$$

where q_{ii} and q'_{ii} are the diagonal entries of $Q(\lambda)$ and $Q'(\lambda)$ respectively.

Finally, it is important to mention that finding all the eigenvalues of a QEP by NMM requires knowing the degree of the scalar polynomial $P(\lambda)$. Therefore, we use Algorithm 3.1 in Section 3.2 to calculate the degree of $P(\lambda)$. More details about the method of Newton Maehly are provided in Algorithm 3.4.

Algorithm 3.4: Modified NMM for finding all eigenvalues of a QEP.

Input : polynomial matrix $Q(\lambda)$.

Calculate $Q'(\lambda)$

Calculate n, the degree of the scalar polynomial $P(\lambda)$ using Algorithm 3.1

for *i=1:n* **do**

  Initialize, λ the initial guess, $\mathcal{C}$ Newton correction, and ϵ tolerance

  while abs $(\mathcal{C}) > \epsilon$ **do**

    Calculate the correction $\mathcal{C} = \frac{P_n(\lambda)}{P'_n(\lambda)}$ using Gaussian elimination and evaluating determinants using the updating in Equation (3.27)

    if *i > 1* **then** *% suppression after finding the first eigenvalue*

      $$\mathcal{S} = \sum_{j=1}^{i-1} \frac{1}{(\lambda - \alpha_j)}$$

    end

    $$\lambda_{new} = \lambda - \frac{\mathcal{C}}{1 - \mathcal{C}\,\mathcal{S}}$$

  end

  $\alpha(i) = \lambda_{new}$

end

Output: all eigenvalues λ.

Results and Discussion

4.1 Overview

In this chapter, we are going to use MATLAB® to test and compare the numerical methods that are based on the Newton iteration in Section 3.4. In addition, we will implement some of the standard methods from Chapter 2 and compare them with the modified Newton Maehly method that is described in Algorithm 3.4. Moreover, we will use the commands **eig**[1] and **polyeig**[2] from MATLAB® and compare them with the mentioned iterative methods to check the accuracy and the efficiency. The real life problems from Section 3.3 as well as some other problems will be tested in these experiments.

4.1.1 Experimental environment

The coming implementations and results have been performed using MATLAB® R2017a (Version 9.2). In addition, all the experiments have been done on a computer with 4GB RAM, CPU Intel® Core™ i5-6300U 2.50GHz, and 64-bit Widows operating system.

[1] **eig**: gives the eigenvalues of a certain SEP or GEP. More information in [14], page 2877.
[2] **polyeig**: gives the eigenvalues of a certain QEP or MMP. More information in [14], page 9408.

4.2 Results

In this section, the eigenvalues of the problems from Section 3.3 are shown in tables using the commands **polyeig** and **eig** from MATLAB® in the first and second columns respectively. The third column contains the eigenvalues using the modified NMM as explained in Algorithm 3.4. The fourth and fifth columns are the eigenvalues using JD and SOAR methods from the literature in Chapter 2. The last column is the symbolic solution that gives an exact or a very accurate approximation using MATLAB®. Moreover, the graph of the eigenvalues and the elapsed time that each method needed to compute all the eigenvalues are provided as well.

4.2.1 The results of the spring problem

4.2.1.1 The eigenvalues

We considered the spring problem 3.3.1 with 8×8 matrices. In this problem, we have only real eigenvalues that are shown in Table 4.1.

Table 4.1: The eigenvalues of the spring problem.

polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
-48.2886210448492	-48.2886210448493	-48.2886210448492	-48.2886210448492	-48.2886210448493	-48.2886210448492
-44.8152474632154	-44.8152474632154	-44.8152474632154	-44.8152474632154	-44.8152474632154	-44.8152474632154
-39.4935886896179	-39.4935886896180	-39.4935886896179	-39.4935886896179	-39.4935886896179	-39.4935886896179
-32.9652630150776	-32.9652630150777	-32.9652630150776	-32.9652630150776	-32.9652630150776	-32.9652630150776
-26.0172391485077	-26.0172391485076	-26.0172391485076	-26.0172391485076	-26.0172391485076	-26.0172391485076
-19.4868329805051	-19.4868329805051	-19.4868329805051	-19.4868329805051	-19.4868329805051	-19.4868329805051
-14.1608106130703	-14.1608106130703	-14.1608106130703	-14.1608106130703	-14.1608106130702	-14.1608106130702
-10.6815934709485	-10.6815934709485	-10.6815934709485	-10.6815934709485	-10.6815934709485	-10.6815934709485
-0.524554113333373	-0.524554113333373	-0.524554113333373	-0.524554113333373	-0.524554113333373	-0.524554113333373
-0.518300524550191	-0.518300524550193	-0.518300524550191	-0.518300524550191	-0.518300524550191	-0.518300524550191
-0.513167019494862	-0.513167019494863	-0.513167019494862	-0.513167019494862	-0.513167019494862	-0.513167019494862
-0.509797298153810	-0.509797298153811	-0.509797298153810	-0.509797298153811	-0.509797298153811	-0.509797298153810
-0.507700538260968	-0.507700538260969	-0.507700538260968	-0.507700538260969	-0.507700538260969	-0.507700538260968
-0.506411310382072	-0.506411310382072	-0.506411310382072	-0.506411310382071	-0.506411310382072	-0.506411310382072
-0.505641399164193	-0.505641399164194	-0.505641399164193	-0.505641399164193	-0.505641399164193	-0.505641399164193
-0.505231370868922	-0.505231370868923	-0.505231370868922	-0.505231370868923	-0.505231370868923	-0.505231370868922

4.2.1.2 The graph of the eigenvalues

Figure 4.1 shows the real eigenvalues of the spring problem 3.3.1.

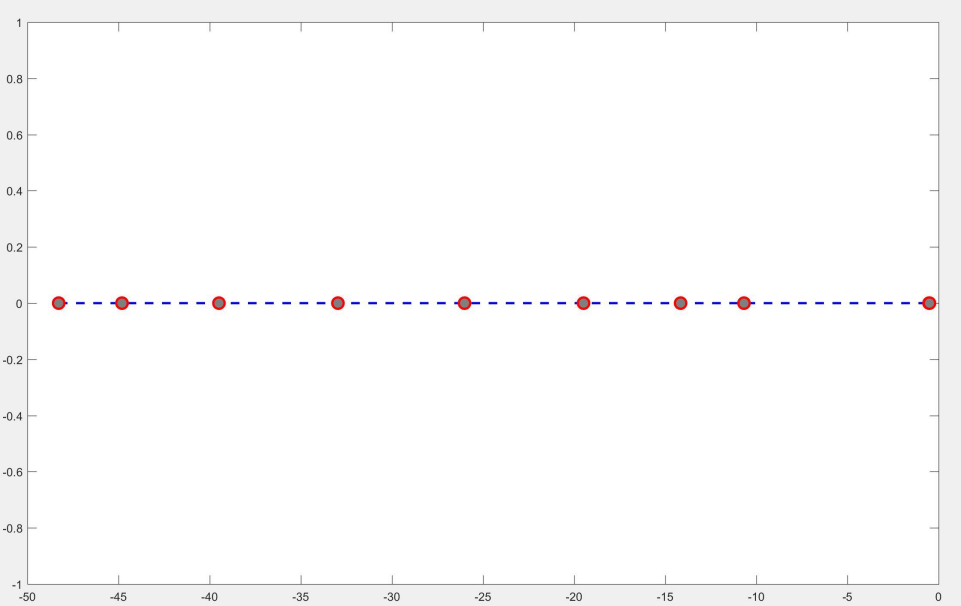

Figure 4.1: The eigenvalues of the spring problem.

4.2.1.3 The elapsed time

Table 4.2 shows the elapsed time of each method using the **tic toc** command in MATLAB®.

Table 4.2: The elapsed time of the spring problem.

Method	Elapsed time (second)
polyeig	0.000516102140634
eig	0.000169025502338
Modified NMM	0.292592752744408
JD method	0.231284323048761
SOAR method	0.10914534634228
Symbolic	2.04444057892055

4.2.2 The results of the bicycle problem

4.2.2.1 The eigenvalues

We considered the same matrices that are mentioned in the bicycle problem 3.3.2. In this problem, we have real and complex eigenvalues.

Table 4.3: The eigenvalues of the bicycle problem.

polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
-3.09728782115114 + 0.00000000000000i	-3.09728782115086 + 0.00000000000000i	-3.09728782115081 - 6.94197596594490e-29i	-3.09728782115084 + 1.27553265615061e-15i	-3.09728782115078 - 9.67773252890883e-21i	-3.09728782115081 + 0.00000000000000i
-0.0710186656720219 + 0.00000000000000i	-0.0710186656720246 + 0.00000000000000i	-0.0710186656720231 - 1.33487886937879e-25i	-0.0710186656720285 + 1.52075556671689e-15i	-0.0710186656720225 - 4.06960560614274e-17i	-0.0710186656720231 + 0.00000000000000i
-0.0377667384512559 - 0.217415070367874i	-0.0377667384512537 - 0.217415070367873i	-0.0377667384512544 - 0.217415070367873i	-0.0377667384512546 - 0.217415070367873i	-0.0377667384512556 - 0.217415070367872i	-0.0377667384512545 - 0.217415070367873i
-0.0377667384512559 + 0.217415070367874i	-0.0377667384512537 + 0.217415070367873i	-0.0377667384512546 + 0.217415070367873i	-0.0377667384512560 + 0.217415070367839i	-0.0377667384512564 + 0.217415070367874i	-0.0377667384512545 + 0.217415070367873i

4.2.2.2 The graph of the eigenvalues

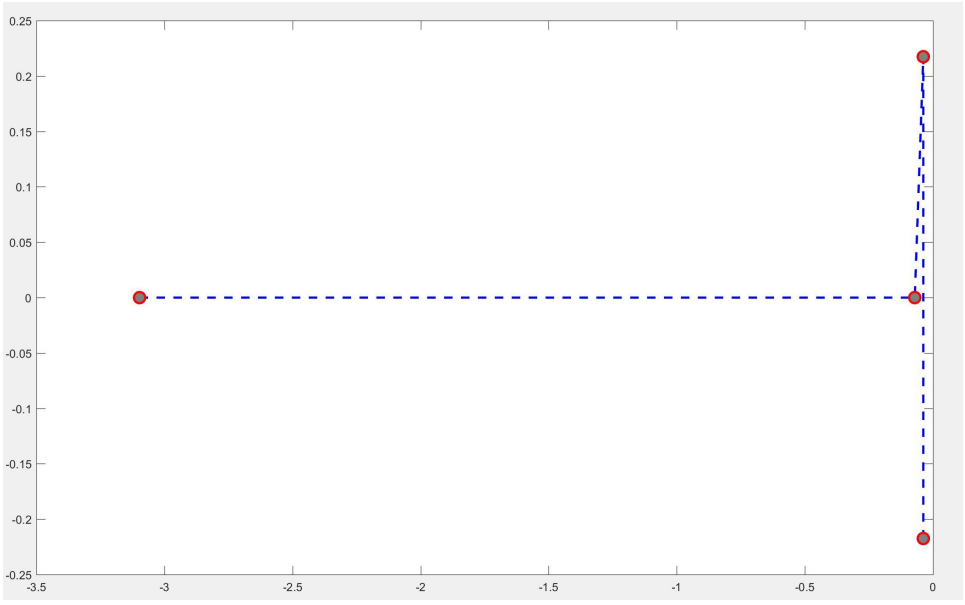

Figure 4.2: The eigenvalues of the bicycle problem.

4.2.2.3 The elapsed time

Table 4.4: The elapsed time of the bicycle problem.

Method	Elapsed time (second)
polyeig	0.000718358384937
eig	0.000252717741360
Modified NMM	0.002534557099851
JD method	0.214376603588428
SOAR method	0.069751129640099
Symbolic	0.617959698084350

4.2.3 The results of the bilby problem

4.2.3.1 The eigenvalues

We considered the same matrices that are mentioned in the bilby problem 3.3.3. In this problem, we have real, zero, complex, and infinite eigenvalues.

Table 4.5: The eigenvalues of the bilby problem.

polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
-19.2093356780573 + 0.000000000000000i	-19.2093356780572 + 0.000000000000000i	-19.2093356780573 + 3.94430452610506e-31i	-19.2093356780573 - 3.60211353777817e-15i	-19.2093356780572 - 1.97959084074126e-14i	-19.2093356780573
-6.26221095987339 - 12.0847655599571i	-6.26221095987339 - 12.0847655599571i	-6.26221095987338 - 12.0847655599571i	-6.26221095987340 - 12.0847655599571i	-6.26221095987338 - 12.0847655599572i	-6.26221095987338 - 12.0847655599571i
-6.26221095987339 + 12.0847655599571i	-6.26221095987339 + 12.0847655599571i	-6.26221095987338 + 12.0847655599571i	-6.26221095987338 + 12.0847655599571i	-6.26221095987338 + 12.0847655599572i	-6.26221095987338 + 12.0847655599571i
-8.04828785593925e-08 + 0.000000000000000i	-8.96777373388142e-08 + 0.000000000000000i	-6.78216289760305e-15 - 8.04640630034406e-16i	-8.57744797910509e-08 + 6.45412745371875e-11i	-6.04903210180670e-08 - 2.21328706211966e-08i	0
-2.47433749573842e-17 + 0.000000000000000i	0.000000000000000 + 0.000000000000000i	3.49464019031050e-15 + 1.31690313126443e-16i	1.46152804072020e-16 + 7.40600115646023e-18i	-4.42278532088848e-17 - 5.46836154223283e-17i	0
8.04901750288605e-08 + 0.000000000000000i	8.96867600970848e-08 + 0.000000000000000i	9.70397148346409e-15 - 1.22000045018037e-16i	8.57827457694782e-08 - 6.45537911025004e-11i	6.04939571947770e-08 + 2.21358875368946e-08i	0
0.000890079219420621 + 0.000000000000000i	0.000890079217694443 + 0.000000000000000i	0.000890079226717209 - 7.08586316108145e-19i	0.000890079218451498 + 1.21723555731509e-14i	0.000890079223081409 - 3.01708389710887e-12i	0.000890079226717208
0.405995438967173 + 0.000000000000000i	0.405995438967173 + 0.000000000000000i	0.405995438967173 - 1.54074395550979e-33i	0.405995438967173 + 4.28581579171165e-16i	0.405995438967173 - 6.35767522722572e-17i	0.405995438967173
4.27687207961013 + 0.000000000000000i	4.27687207961013 + 0.000000000000000i	4.27687207961013 - 2.11904447449192e-24i	4.27687207961013 + 4.03955008677683e-16i	4.27687207961013 - 2.47055975940458e-17i	4.27687207961013
∞	∞	∞	∞	∞	∞

4.2.3.2 The graph of the eigenvalues

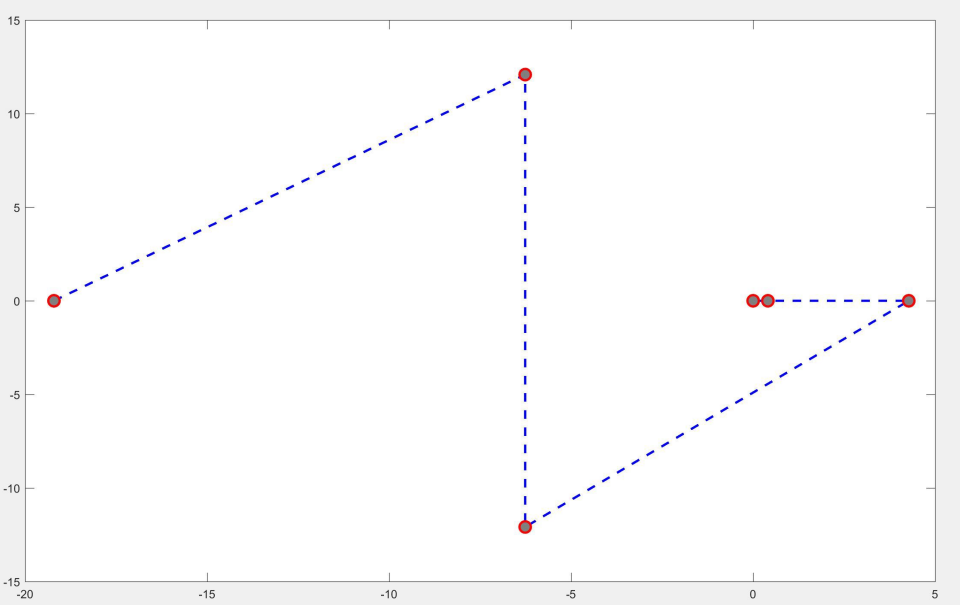

Figure 4.3: The eigenvalues of the bilby problem.

4.2.3.3 The elapsed time

Table 4.6: The elapsed time of the bilby problem.

Method	Elapsed time (second)
polyeig	0.000495076598911509
eig	0.000151794747245336
Modified NMM	0.0210491847801561
JD method	0.560473386278247
SOAR method	0.088906979358376
Symbolic	1.13579361165652

4.2.4 The results of the acoustic-modeling problem 1

4.2.4.1 The eigenvalues

We considered the same matrices that are mentioned in the first acoustic-modeling problem 3.3.4.1. In this problem, we have real and infinite eigenvalues.

Table 4.7: The eigenvalues of the acoustic-modeling problem 1.

polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
-1.00000000000000 + 0.00000000000000i	-1.00000000000000 + 0.00000000000000i	-1.00000000000000 + 0.00000000000000i	-1.00000000112532 + 0.00000000000000i	-1.00000000000000 + 0.00000000000000i	-1
0.999999989049928 + 0.00000000000000i	1.00000000000000 + 0.00000000000000i	0.999999999274104 + 2.74880930903236e-25i	1.00000000121028 + 0.00000000000000i	0.999997392033132 + 0.00000000000000i	1
1.00000000000000 + 0.00000000000000i	1.00000000000000 + 0.00000000000000i	1.00000000000000 - 7.56884830468748e-09i	1.00000001703073 + 0.00000000000000i	1.00000130398344 - 2.25858357700408e-06i	1
1.00000001095007 + 0.00000000000000i	1.00000000000000 + 0.00000000000000i	1.00000000000000 + 7.70476461097103e-09i	0.999999982969270 + 0.00000000000000i	1.00000130398344 + 2.25858357700408e-06i	1
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
∞	∞	∞	∞	∞	∞

4.2.4.2 The graph of the eigenvalues

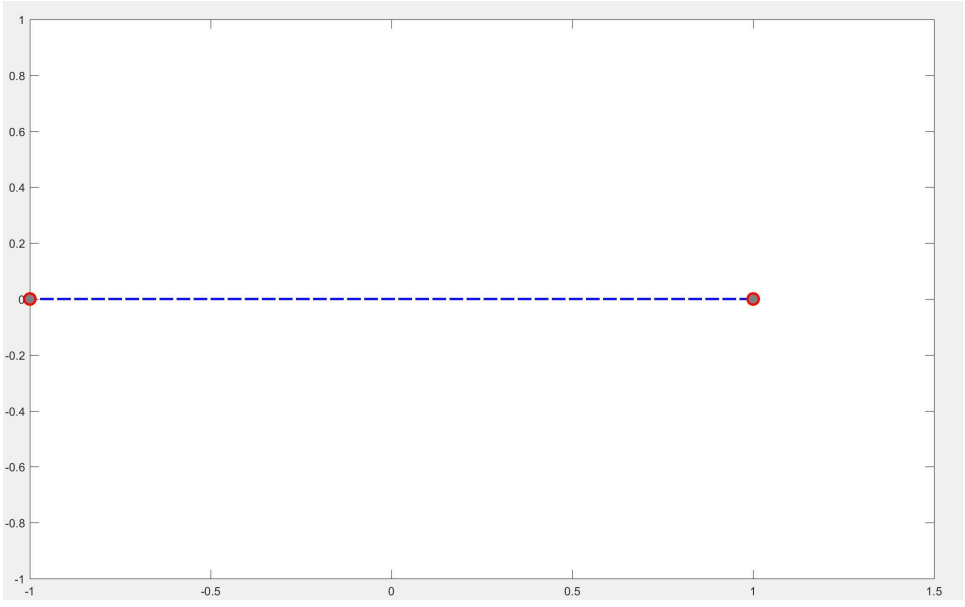

Figure 4.4: The eigenvalues of the acoustic-modeling problem 1.

4.2.4.3 The elapsed time

Table 4.8: The elapsed time of the acoustic-modeling problem 1.

Method	Elapsed time (second)
polyeig	0.000164922941601689
eig	4.34871438054205e-05
Modified NMM	0.00187692153688489
JD method	0.542664243967800
SOAR method	0.0747092720334178
Symbolic	0.611516298243037

4.2.5 The results of the acoustic-modeling problem 2

4.2.5.1 The eigenvalues

We considered the same matrices that are mentioned in the second acoustic-modeling problem 3.3.4.2. In this problem, we have real, pure imaginary, and infinite eigenvalues.

Table 4.9: The eigenvalues of the acoustic-modeling problem 2.

polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
0.000000000000000 - 1.000000000000000i	0.000000000000000 - 1.000000000000000i	0.000000000000000 - 1.000000000000000i	0.000000000000000 - 1.000000000000000i	0.000000000000000 - 1.000000000000000i	-1i
0.000000000000000 + 1.000000000000000i	0.000000000000000 + 1.000000000000000i	0.000000000000000 + 1.000000000000000i	0.000000000000000 + 1.000000000000000i	0.000000000000000 + 1.000000000000000i	+ 1i
0.333333333333336 + 0.000000000000000i	0.333333333333333 + 0.000000000000000i	0.333333333333333 + 1.97302823776934e-42i	0.333333333333331 + 3.36469448676242e-17i	0.333333333333330 + 2.18068397039413e-16i	1/3
0.499999999999995 + 0.000000000000000i	0.500000000000002 + 0.000000000000000i	0.500000000000000 + 5.17689969051289e-31i	0.500000000000004 - 7.19677987715788e-18i	0.500000000000003 + 4.44522679167366e-16i	1/2
1.000000000000000 + 0.000000000000000i	1.000000000000000 + 0.000000000000000i	1.000000000000000 + 0.000000000000000i	1.000000000000000 - 1.92693779236989e-18i	1.000000000000000 + 2.14685988811526e-16i	1
∞	∞	∞	∞	∞	∞

4.2.5.2 The graph of the eigenvalues

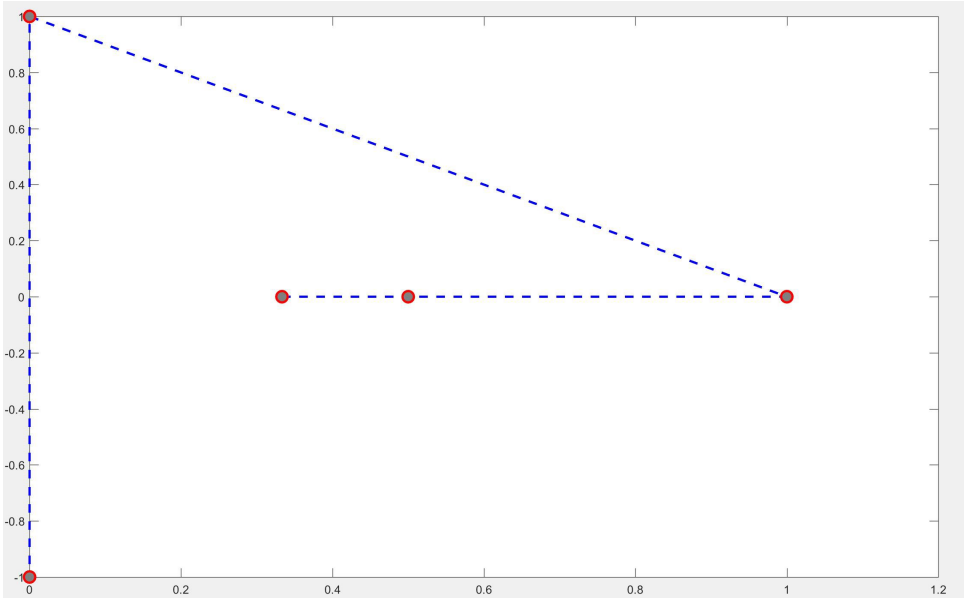

Figure 4.5: The eigenvalues of the acoustic-modeling problem 2.

4.2.5.3 The elapsed time

Table 4.10: The elapsed time of the acoustic-modeling problem 2.

Method	Elapsed time (second)
polyeig	0.000407343419615
eig	0.000109599860841
Modified NMM	0.0021634854043197
JD method	0.406944814814511
SOAR method	0.072359325243631
Symbolic	0.673750913845404

4.3 Discussion

4.3.1 The accuracy

The accuracy of the modified NMM depends on a given convergence tolerance that is tested after every iteration. For a given stopping criterion we used in this method, the results were slightly more accurate than the command of MATLAB® and the methods from the literature. In fact, there is not that notable big difference between the modified NMM and the other methods in most of the QEPs that we solved; however, the modified NMM has a slightly better accuracy in the problems with singular leading matrices as well as the bicycle problem 3.3.2.

Table 4.11: The accuracy of the bicycle problem.

polyeig	eig	Modified NMM	JD method	SOAR method	Exact
-3.097287821151 14 + 0.000000000000000i	-3.09728782115086 + 0.000000000000000i	-3.09728782115081 - 6.94197596594490e-29i	-3.0972878211508 4 + 1.27553265615061e-15i	-3.09728782115078 - 9.67773252890883e-21i	-3.09728782115081 + 0.000000000000000i
-0.071018665672021 9 + 0.000000000000000i	-0.071018665672024 6 + 0.000000000000000i	-0.0710186656720231 - 1.33487886937879e-25i	-0.071018665672028 5 + 1.52075556671689e-15i	-0.071018665672022 5 - 4.06960560614274e-17i	-0.0710186656720231 + 0.000000000000000i
-0.037766738451255 9 - 0.217415070367874i	-0.037766738451253 7 - 0.217415070367873i	-0.037766738451254 4 - 0.217415070367873i	-0.037766738451254 6 - 0.217415070367873i	-0.037766738451255 6 - 0.217415070367872i	-0.0377667384512545 - 0.217415070367873i
-0.037766738451255 9 + 0.217415070367874i	-0.037766738451253 7 + 0.217415070367873i	-0.037766738451254 6 + 0.217415070367873i	-0.037766738451256 0 + 0.217415070367839i	-0.037766738451256 4 + 0.217415070367874i	-0.0377667384512545 + 0.217415070367873i

Table 4.12: The accuracy of the acoustic-modeling problem 1.

polyeig	eig	Modified NMM	JD method	SOAR method	Exact
-1.000000000000000 + 0.000000000000000i	-1.000000000000000 + 0.000000000000000i	-1.000000000000000 + 0.000000000000000i	-1.000000001125 32 + 0.000000000000000i	-1.000000000000000 + 0.000000000000000i	-1
0.999999989049928 + 0.000000000000000i	1.000000000000000 + 0.000000000000000i	0.999999999274104 + 2.74880930903236e-25i	1.000000001210 28 + 0.000000000000000i	0.999997392033132 + 0.000000000000000i	1
1.000000000000000 + 0.000000000000000i	1.000000000000000 + 0.000000000000000i	1.000000000000000 - 7.56884830468748e-09i	1.000000017030 73 + 0.000000000000000i	1.000001130398344 - 2.25858357700408e-06i	1
1.000000010950 07 + 0.000000000000000i	1.000000000000000 + 0.000000000000000i	1.000000000000000 + 7.70476461097103e-09i	0.999999982969270 + 0.000000000000000i	1.000001130398344 + 2.25858357700408e-06i	1
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
∞	∞	∞	∞	∞	∞

4.3.2 The efficiency

Based on the elapsed time tables in Section 4.2, the modified NMM performs faster than Jacobi-Davidson method in all the problems of dimension less than 8×8, and faster than SOAR method in all the problems of dimension less than 7×7, and obviously, the symbolic method is the slowest in every problem. On the other hand, the modified NMM starts to perform slower than the other iterative methods when the dimension of the matrices gets bigger than 6×6 for the QEPs with non-singular leading matrices, and 10×10 for the QEPs with singular leading matrices.

In Jacobi-Davidson method, there are two stopping criteria. The first criterion is the convergence tolerance, and the second one is the number of iterations. The number of iterations is considered as a criterion when the method fails to determine the eigenvalue depending on the given convergence tolerance. In other words, the criterion of the iteration number is considered when the problem contains infinite roots. This issue is avoided in the modified NMM because we ignore calculating the infinite roots by giving the number of the finite eigenvalues at the beginning of the algorithm.

The efficiency of the iterative methods in MATLAB® can be affected by many issues that might happen to the computer during the experiment. In addition, choosing a random initial guess in every iteration using the command **rand**[3] can also affect the efficiency of the algorithm. Consequently, repeating the experiment might lead to ending up with different elapsed time for the small QEPs. Therefore, for more reliable results, the way of computing the elapsed time of all the problems in this thesis was by doing every experiment for fifty times and taking the average of the entire elapsed time. Table 4.13 and Figure 4.6 show the elapsed time of the spring problem 3.3.1 with 10 different dimensions.

[3] **rand**: gives a random number between 0 and 1. More information in [14], page 10187.

Table 4.13: The elapsed time of the spring problem with 10 different sizes.

Method / Size	polyeig	eig	Modified NMM	JD method	SOAR method	Symbolic
2×2	0.000443	7.96e-05	0.00229	0.203611	0.072749	0.598963
3×3	0.000458	9.93e-05	0.00431	0.208553	0.074351	0.836125
4×4	0.000471	0.000122	0.009631	0.223261	0.079641	1.053089
5×5	0.000485	0.00013	0.021504	0.227067	0.085794	1.314622
6×6	0.00049	0.000133	0.049482	0.228645	0.097949	1.649801
7×7	0.000499	0.000149	0.111957	0.229063	0.103299	2.00808
8×8	0.000516	0.000169	0.292593	0.231284	0.109145	2.044441
9×9	0.000518	0.000215	0.366352	0.240307	0.116326	2.482973
10×10	0.000523	0.000241	0.417741	0.24454	0.125054	2.595643
11×11	0.000538	0.000242	0.492477	0.251765	0.127997	2.932986

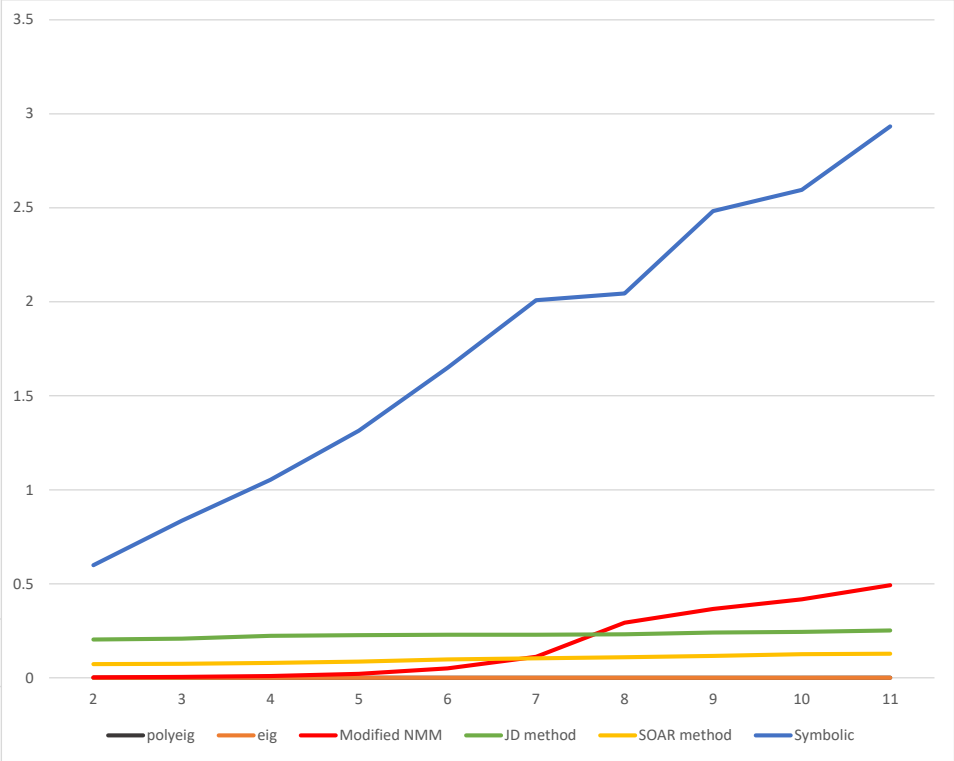

Figure 4.6: The efficiency of the methods for 10 different dimensions. The x-axis represents the dimension of the matrices, and the y-axis represents the elapsed time.

Figures 4.7 and 4.8 show the elapsed time of the spring problem 3.3.1 up to dimension of 40×40 in regular scale and logarithmic scale respectively. The x-axis represents the dimension of the matrices, and the y-axis represents the elapsed time.

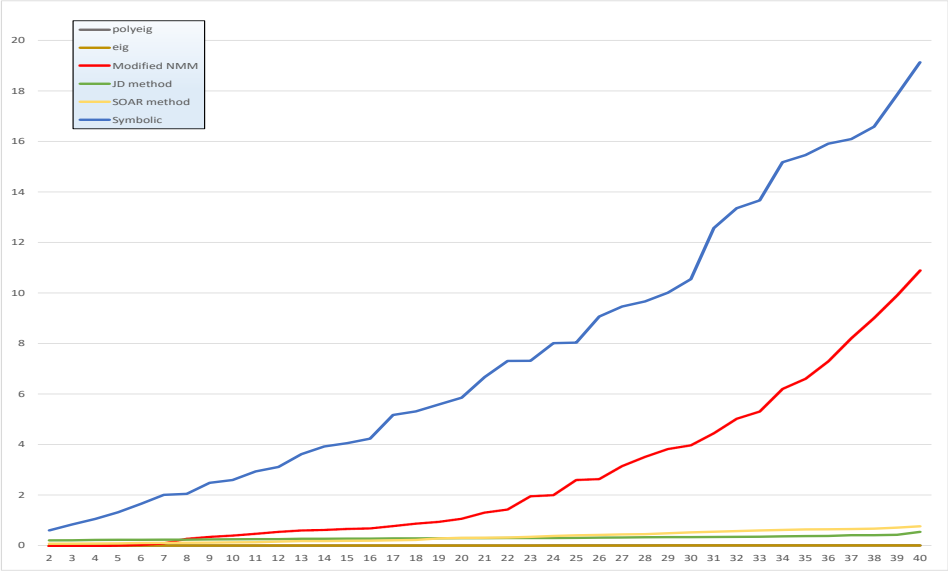

Figure 4.7: The efficiency of the methods.

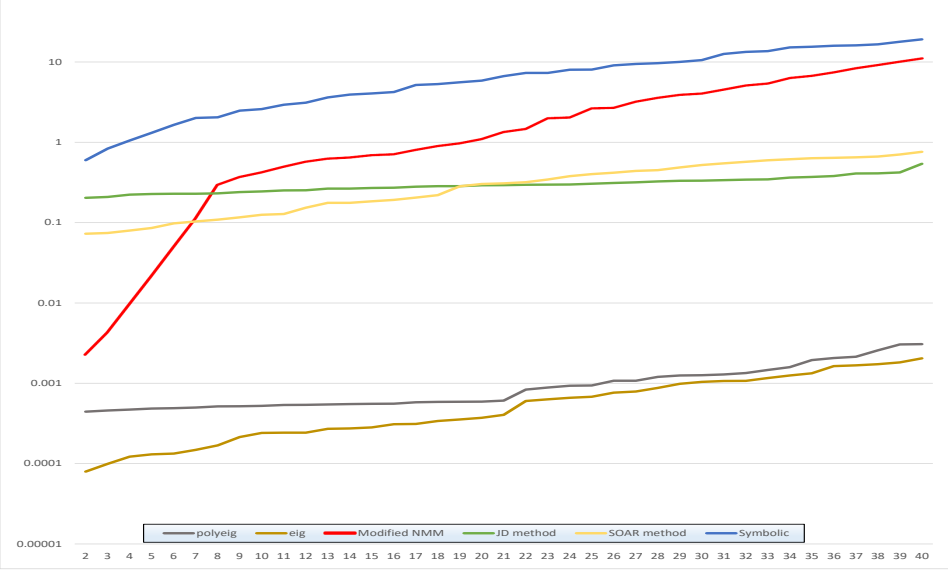

Figure 4.8: The efficiency of the methods in logarithmic scale.

4.4 Conclusion and future work

QEPs can be solved by using NMM that was originally structured to find zeros of scalar polynomials. The implicit deflation can be applied on NMM in order to find all the solutions λ of any QEP after determining the number of the eigenvalues. The advantage of using NMM is to deal directly with the QEP without linearizing the problem to a GEP. Consequently, in solving QEPs, one does not need to worry about the singularity of the leading matrix of the QEP. In addition, determining the number of the eigenvalues using Algorithm 3.1 will save the time of calculating the infinite eigenvalues in case we have a singular leading matrix. Moreover, NMM is an efficient method when we deal with small and singular matrices; however, the method starts to get slower when we deal with high dimensional matrices.

We suggest a similar modification on the Newton methods that are mentioned in Section 3.4 and in [11] to improve the efficiency. In Section 3.4, the three methods that are based on Newton iteration perform differently in terms of efficiency when we hold the same tolerance for some problems. For example, in the bicycle problem 3.3.2, NMM is slightly faster than NTI, whereas NII performs significantly the slowest. In fact, NII needed 48 iterations to converge to one of the eigenvalues of Problem 3.3.2, while NMM and NTI converged to the same eigenvalue with 9 and 10 iterations respectively. On the other hand, the three methods performed almost the same with 10 iterations for both NMM and NTI, and 9 iterations for NII to converge to one of the eigenvalues of the bilby problem 3.3.3. In addition, we suggest working on more general problems since NMM can deal with MPP and NLEP as long as the functions that are placed instead of λ in a QEP are differentiable. Finally, we suggest doing these modifications in more controlled environment or enhancing the MATLAB® issues that we mentioned earlier in Section 4.3.2.

Bibliography

[1] Zhaojun Bai and Yangfeng Su. Soar: A second-order arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM Journal on Matrix Analysis and Applications*, 26(3):640–659, 2005.

[2] F. L. Bauer and J. Stoer. Algorithm 105: Newton maehly. *Commun. ACM*, 5(7):387–388, July 1962.

[3] NG Bean, L Bright, Guy Latouche, CEM Pearce, PK Pollett, and Peter G Taylor. The quasi-stationary behavior of quasi-birth-and-death processes. *The Annals of Applied Probability*, pages 134–155, 1997.

[4] Timo Betcke, Nicholas J Higham, Volker Mehrmann, Christian Schröder, and Françoise Tisseur. Nlevp: A collection of nonlinear eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):7, 2013.

[5] Dario A Bini and Vanni Noferini. Solving polynomial eigenvalue problems by means of the ehrlich–aberth method. *Linear Algebra and its Applications*, 439(4):1130–1149, 2013.

[6] Philippe G Ciarlet and Jacques-Louis Lions. *Handbook of numerical analysis*, volume 8. Gulf Professional Publishing, 1990.

[7] Timothy F. Walsh David M. Day. Quadratic eigenvalue problems. *Sandia Report, SAND2007-2072*, 2007.

[8] Brent M Dingle. Symbolic determinants: Calculating the degree. Technical report, unpublished–Tech Report Anticipated, 2004.

[9] John W. Eaton, David Bateman, Søren Hauberg, and Rik Wehbring. *GNU Octave version 4.2.1 manual: a high-level interactive language for numerical computations*, 2017.

[10] Walter Gander. Zeros of determinants of λ-matrices. *PAMM*, 8(1):10811–10814, 2008.

[11] Stefan Güttel and Françoise Tisseur. The nonlinear eigenvalue problem. *Acta Numerica*, 26:1–94, 2017.

[12] Peter Lancaster. *Lambda-matrices and vibrating systems*. Courier Corporation, 2002.

[13] Dinesh Manocha. Solving systems of polynomial equations. *IEEE Computer Graphics and Applications*, 14(2):46–55, 1994.

[14] MATLAB. *MATLAB: the language of technical computing: MATLAB function reference*. MathWorks, Inc., Natick, Massachusetts, 2000.

[15] Jaap P Meijaard, Jim M Papadopoulos, Andy Ruina, and Arend L Schwab. Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 463, pages 1955–1982. The Royal Society, 2007.

[16] H.P. Moser. Newton maehly. http://www.mosismath.com/NewtonMaehly/NewtonMaehly.html, 2010.

[17] Bor Plestenjak. Numerical methods for the tridiagonal hyperbolic quadratic eigenvalue problem. *SIAM Journal on Matrix Analysis and Applications*, 28(4):1157–1172, 2006.

[18] Jiang Qian and Wen-Wei Lin. A numerical method for quadratic eigenvalue problems of gyroscopic systems. *Journal of sound and vibration*, 306(1):284–296, 2007.

[19] XIAOLIN QIN, ZHI SUN, TUO LENG, and YONG FENG. Computing the determinant of a matrix with polynomial entries by approximation. *arXiv preprint arXiv:1408.5879*, 2014.

[20] Gerard LG Sleijpen, Albert GL Booten, Diederik R Fokkema, and Henk A Van der Vorst. Jacobi-davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT Numerical Mathematics*, 36(3):595–633, 1996.

[21] Françoise Tisseur. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra and its Applications*, 309(1-3):339–361, 2000.

[22] Françoise Tisseur and Karl Meerbergen. The quadratic eigenvalue problem. *SIAM review*, 43(2):235–286, 2001.

[23] J. H. Wilkinson. *The algebraic eigenvalue problem*. Clarendon Press, Oxford, 1965.

[24] Ping Zhang. Iterative methods for computing eigenvalues and exponentials of large matrices, 2009.

Appendix

A.1 Newton-trace iteration (NTI)

```
clc;clear;close all;
% Newton trace iteration computes one eigenvalue of a QEP.
% NTI requires: (Tolerance, iterations number, initial guess
% polynomial matrix Q(lambda) and Q'(lambda)).
% Ali Hasan Ali .... Wright State University .... December 2017
eps=input('eps='); % Tolerance (10^(-8))
ite_num = input('ite_num='); %Number of iterations (25)
int_guss = input('int_guss='); % Initial guess (real+imaginary)
Q=input('Q='); % The polynomial matrix
Q_prime=diff(Q); % Calculating Q'
Q=inline(Q); Q_prime= inline(Q_prime);
for i = 0:ite_num
    [Lt,Ut]=lu(Q(int_guss)); % LU decomposition for Q
    Accc=diag(Ut); Acc=prod(Accc); Ac=abs(Acc);
    Bc=norm(Q(int_guss),'fro');
    if Ac/Bc<eps % The stopping criterion
        break
    end
    X=Lt\Q_prime(int_guss); %Solve LX=Q for X
    Y=Ut\X; %Solve UY=X for Y
    int_guss=int_guss-1/trace(Y); %Updating the eigenvalue
end
The_eigenvalue = int_guss %The required eigenvalue
```

A.2 Newton inverse iteration (NII)

```
clc;clear;close all;
% Newton inverse iteration computes one eigenvalue of a QEP.
% NII requires:
% (Tolerance, iterations number, initial guess,
% initial normalized vector, non-zero vector,
% polynomial matrix Q(lambda) and Q'(lambda)).
% Ali Hasan Ali .... Wright State University .... December 2017
eps=input('eps='); % Tolerance (10^(-8))
ite_num = input('ite_num='); %Number of iterations (25)
int_guss = input('int_guss='); % Initial guess (real+imaginary)
Q=input('Q='); % The polynomial matrix
Q_prime=diff(Q); % Calculating Q'
Q=inline(Q);
Q_prime= inline(Q_prime);
v = input('v=');
u = input('u='); %non_zero vector
v = v/norm(v); %Normalizing the vector v
for i = 0:ite_num
    if norm(Q(int_guss)*v)<eps % The stopping criterion
        break
    end
    nv = Q(int_guss)\Q_prime(int_guss)*v; %Solve Qnv=Q'v for nv
    Ac=u'*v;
    Bc=u'*nv;
    int_guss=int_guss-Ac/Bc; %Updating the eigenvalue
    v = nv/norm(nv); %Normalizing the new vector v
end
The_eigenvalue=int_guss  %The required eigenvalue
```

A.3 The modified Newton Maehly method (NMM)

```
clc;clear;close all;
% Modified Newton Maehlycoputes all eigenvalues of a QEP.
% Modified NMM requires:
% (Tolerance, initial guess, initial Newton correction
% Number of eigenvalues, polynomial matrix Q(lambda), Q'(lambda)).
% Ali Hasan Ali .... Wright State University .... December 2017
Q=input('Q='); % The polynomial matrix
Q_prime=diff(Q); % Calculating Q'
Q=inline(Q); Q_prime= inline(Q_prime);
Q1=Q; Q1_prime=Q_prime;
d=input('Number_of_eigenvalues='); %Number of eigenvalues ...
    (Algorithm 3.1)
eps=input('eps='); % Tolerance (10^(-8))
for n=1:d %This loop for the eigenvalues of the entire problem
    t = rand*1i; %Initial complex random number for t
    CC = 1; %Initilizing the correction of the newton iterration
    while abs(CC)>eps  % The stopping criterion
        Q=Q1(t);
        Q_prime=Q1_prime(t);
        CC = correction(Q,Q_prime); % Calculating the newton
                                    % correction with an external
                                    % function
        g = 0;
        if n>1 % Suppression after finding the first eigenvalue
             g = sum(1./(t-root(1:n-1)));
        end
        t = t-CC/(1-CC*g); % Updating the eigenvalue
    end
    root(n) = t;
end
[root'] %The required eigenvalues
```

A.4 Newton correction

```
% The external function correction.m to calculate the Newton ...
    orrection
% The Newton correction= P_n(lambda)/P'_n(lambda)
% Ali Hasan Ali .... Wright State University .... December 2017
function CCs = correction(Q,Q_prime)
k = length(Q);
Cs = 0;
for n = 1:k
    [CM,KM]= max(abs(Q(n:k,n)));
    if CM == 0
        CCs = 0;
        return
    end
    KM = KM+n-1;
    if KM ≠ n
        h = Q(n,:); Q(n,:) = Q(KM,:); Q(KM,:) = h;
        h = Q_prime(KM,:); Q_prime(KM,:) = Q_prime(n,:);
        Q_prime(n,:) = h;
    end
    Cs = Cs+Q_prime(n,n)/Q(n,n);
    % Gaussian elimination with the modification in equation (3.27)
    Q_prime(n+1:k,n)=(Q_prime(n+1:k,n)*Q(n,n)- Q_prime(n,n) ...
        *Q(n+1:k,n))/Q(n,n)^2;
    Q(n+1:k,n)=Q(n+1:k,n)/Q(n,n);
    Q_prime(n+1:k,n+1:k)=Q_prime(n+1:k,n+1:k)- Q_prime(n+1:k,n) ...
        *Q(n,n+1:k)-Q(n+1:k,n)*Q_prime(n,n+1:k);
    Q(n+1:k,n+1:k)=Q(n+1:k,n+1:k)-Q(n+1:k,n)*Q(n,n+1:k);
end
CCs = 1/Cs;
```

A.5 Calculating the degree of polynomial determinants

```
clc;clear;close all;
% The number of the eigenvalues of a QEP can be determined
% if we know the degree of the scalar polynomial det(Q(lambda))
% This Matlab code, computes  the degree of det(Q(lambda))
% without dealing with the determinant symbolicly
% This code requires only a polynomial matrix Q(lambda).
% Ali Hasan Ali .... Wright State University .... December 2017
Q=input('Q='); n=length(Q);
% Transforming the symbolic matrix to a numerical matrix
for i=1:n
    for j=1:n
        Nx(i,j)=feval(symengine, 'degree', Q(i,j), x);
    end
end
NumMat=Nx; Un=zeros(n,1); Un(1,1)=NumMat(1,1);
if n==1 % Starting to reduce the main numerical matrix
    Deg=Nx(1,1);
elseif n==2
    Deg=max(Nx(2,2)+Nx(1,1),Nx(2,1)+Nx(1,2));
else
 for k=1:n-1
     for i=1:n-k
         for j=1:n-k
          Nx1(i,j)=max(Nx(i+1,j+1)+Nx(1,1),Nx(i+1,1) +Nx(1,j+1));
         end
     end
     Nx=Nx1; Un(k+1)=Nx1(1,1); Nx; Nx1=0;
 end
 Deg=Nx;
 for s=1:n-2
     Deg=Deg-s*Un(n-1-s);  % Unwinding the initial degree
 end; end
```