CHAPTER
ONE
INTRODUCTION

Contents

1.1. Motivation

This thesis handles two closely connected problems in numerical linear algebra and scientific computing. First, we consider the generalized eigenvalue problem

$$Ax = \lambda Bx, \tag{1.1}$$

which is a common building block in many applications and analysis tasks. Second, we regard the generalized Sylvester equation

$$AXB + CXD = Y, \tag{1.2}$$

which is closely connected to the generalized eigenvalue problem [107], as well as many other applications in numerical linear algebra. For example, its special cases, like the Lyapunov equation, play an important role in system and control theory [12].

The generalized eigenvalue problem consists of finding all pairs of scalar values λ and vectors x fulfilling (1.1) for a given pair of matrices (A, B) of size $m \times m$. One of the most common strategies to solve this problem is based on the generalized Schur decomposition of the matrix pair (A, B)

$$(A, B) = \left(QTZ^{\mathsf{H}}, QSZ^{\mathsf{H}}\right). \tag{1.3}$$

The transformation matrices Q and Z in equation (1.3) are unitary and the newly emerged matrix pair (T, S) consists of upper triangular matrices of size $m \times m$ containing the same eigenvalues as (A, B). Since the matrices T and S are upper triangular, the eigenvalues of (T, S) can directly be obtained from the diagonals of T and S.

The determination of the generalized Schur decomposition (1.3) is a computational tough task normally done with the QZ algorithm developed by Moler and Stewart in 1973 [80, 155, 129]. The algorithm is an extension of the Francis QR algorithm [73], which was developed in 1961 to solve the standard eigenvalue problem, i.e. $B = I$ in (1.3). The QZ algorithm became part of the

EISPACK library [150] and later it was integrated into LAPACK [10]. Commonly used tools like MATLAB® [127] or GNU Octave [71] and libraries like SLICOT [149], SCIPY [99], or Trillinos [87] rely on this implementation. Since LAPACK wrappers exist for most common programming languages, the QZ algorithm can be interfaced from almost all mathematical software projects. Even newly developed programming languages and environments for scientific computing, like Julia [37], still provide an interface to this implementation. Hence, the QZ algorithm can be easily used from mathematical research projects as well as from industrial applications. Some acceleration techniques developed by Kågström et al. [104] are already included in LAPACK. An algorithmic improvement based on the aggressive early deflation and multi-shift ideas from the QR algorithm [46, 47] is transferred to the QZ algorithm by Kågström, Dackland, and Kressner [57, 103]. Since this algorithm requires a large set of properly selected tuning parameters, it was not yet integrated into standard software like LAPACK.

Although the QZ algorithm is known for decades and many improvements were developed, the algorithm does not benefit from multi-core CPUs, accelerator devices, or massively parallel computer systems very well. These bad utilization of current computers' capabilities leads to runtime issues when using the QZ algorithm. This can be illustrated with a real world example of order 20 209 coming from a finite element model of a heat equation on a steel profile [34]. Using a current high performance computing (HPC) server equipped with two eight core Intel® Xeon® Silver 4110 CPUs, it takes more than 1.4 days to obtain Q and Z with the LAPACK implementation. Even the optimized versions from Kågström, Dackland, and Kressner [57, 103] take more than 13 hours, compare the numerical results in Chapter 5. A possibly available accelerator cannot be used in most parts of the QZ algorithm.

In contrast to the runtime of the QZ algorithm, the computation of an LU decomposition of the same size on the same machine only takes 23.5 seconds without using an additional accelerator. This motivates us to develop a new class of algorithms, which avoids the operations that cause the bottlenecks in the QZ algorithm. Instead of these operations we integrate well performing operations, like the LU decomposition or the QR decomposition, which exist in highly optimized and parallelized variants for shared memory systems [67, 118, 40, 165, 51], GPU accelerated architectures [166, 167, 65], and distributed systems [44, 55]. Furthermore, we expect to easily integrate new computer architectures and accelerator devices due to the fact that optimized versions of these fundamental operations are mostly available soon after a new platform appears.

The first part of this thesis (Chapters 2 to 5) is driven by the derivation of fast algorithms to compute an approximation of the generalized Schur decomposition (1.3) with the help of these operations. Our developed algorithms and the involved optimizations allow to compute a sufficient approximation of the generalized Schur decomposition of the steel profile example [34] within 38 minutes on the same system.

After we handle the fast approximation of the generalized Schur decomposition, we regard the aforementioned generalized Sylvester equation

$$AXB + CXD = Y \tag{1.4}$$

in Chapters 6 and 7. Most of the direct solution strategies for Sylvester-type matrix equations require the computation of two generalized Schur decompositions first [19, 100, 101, 78, 107, 105, 79]. Another connection between the generalized Schur decomposition (1.3) and Sylvester-type matrix equation is given by the coupled generalized Sylvester equation [107, 102]

$$\begin{aligned} AR - LB &= C, \\ DR - LE &= F. \end{aligned} \tag{1.5}$$

It is used to swap single eigenvalues or blocks on the diagonal of the matrix pair (T, S). Thus it can be used to reorder eigenvalues after a generalized Schur decomposition is computed. Sylvester-type matrix equations have a large impact in systems and control theory as well, e.g. [12]. Hence, the Lyapunov equation, a symmetric variant of the generalized Sylvester equation, lies in the field of application of this thesis as well. With $D = A^{\mathsf{H}}$ and $B = C^{\mathsf{H}}$ in the Sylvester equation (1.4) we obtain

$$AXC^{\mathsf{H}} + CXA^{\mathsf{H}} = Y, \tag{1.6}$$

where Y must be a symmetric right-hand side. In contrast to the (coupled) generalized Sylvester equation it only requires the computation of one generalized Schur decomposition in a direct solution process [19, 85, 140].

The computation of the generalized Schur decomposition is one of the most time-consuming operations in the direct solution process of above mentioned equations (1.4) to (1.6). Hence, the improvements made by our algorithms to approximate the generalized Schur decomposition already result in a notable runtime reduction. With the implementation of the Bartels-Stewart algorithm [19] from SLICOT [149], which is the standard solver in MATLAB and GNU Octave, the solution of the Lyapunov equation, arising from the steel profile example, still takes around 11 hours after the generalized Schur decomposition is known. Even newer algorithms, like the recursive blocking approach [100, 101], struggle with the parallel capabilities of current computer architectures. Our goal is the development of an algorithmic framework, which results in fast direct solvers for Sylvester-type matrix equations. Thereby features of current computer architectures and the optimization techniques of the compilers are taken into account. Since our algorithms, derived in the first part of this thesis, only approximate the generalized Schur decomposition (1.3) and the coefficient matrices of the transformed equation get perturbed, we transfer the iterative refinement procedure from standard linear systems to the Sylvester-type matrix equations. The iterative refinement procedure is formulated in a way such that only one generalized Schur decomposition is required over all iteration steps. Using the optimization and parallelization ideas, introduce in Chapter 6, we reduce the overall runtime of solving the Lyapunov equation (1.6) for the steel profile example to 48 minutes instead of 1.86 days with the SLICOT implementation.

Looking into future hardware, the parallel capabilities of the HPC systems will increase more and more, and accelerator devices, like GPUs, will exist in almost all computational hardware. In this way, moving forward to new algorithms and adjust existing ones to support multi-core and accelerator based systems is the aim of the ideas and approaches presented in this thesis.

1.2. Outline of the Thesis

This thesis is organized in two parts. The first part comprises Chapters 2 to 5 and handles the approximation of the generalized Schur decomposition and the optimizations included in the algorithms. The second part, consisting of Chapters 6 and 7, covers the direct solution of the generalized Sylvester equation.

Chapter 2 In Chapter 2, basic properties of the generalized eigenvalue problem and the generalized Schur decomposition are introduced. The Deflation Theorem 2.6 [155, 80], which provides a way to split large eigenvalue problems into smaller ones, is presented as a key building block for our algorithms. Furthermore, we recall the eigenvalue reordering technique develop by Kågström [102]. The reordering strategy is also used later by the solvers for the Sylvester-type matrix equations in Section 6.3 to regularize the memory access patterns. Beside the fact that the spectrum of a generalized eigenvalue problem is invariant under two-sided transformation, as a result of the Deflation Theorem 2.6, we present eigenvalue transformations leaving the deflating subspaces untouched. These transformations are used by our algorithms to move, scale, and modify the spectrum of a matrix pair (A, B).

Chapter 3 The spectral divide-and-conquer approach, which is the main idea behind our algorithmic strategies to compute the generalized Schur decomposition, is derived in Chapter 3. It extends the known spectral division ideas [15, 162, 121] to a recursive scheme. Therefor, we develop a generic spectral division framework, which uses the Deflation Theorem as key building block. In the context of this generic framework we discuss how to extract unitary transformations from a given deflating subspace with the help of two QR decompositions [162]. This extraction method is also used to remove infinite eigenvalues. Since the generic framework requires the computation of a deflating subspace, we suggest two ways to perform this tasks. In Section 3.2, we recall the generalized matrix sign function

$$S = \operatorname{sign}(A, B). \tag{1.7}$$

It was introduced by Roberts [144] to solve the algebraic Riccati equation and extended by Gardiner and Laub [77] to its generalized case. The generalized matrix sign function can be used to compute a deflating subspace, which belongs either to the eigenvalues in the open left half-plane or in the open right half-plane. In this way, the generalized matrix sign function is used to split the spectrum at the imaginary axis. Since a recursive application of this generalized matrix sign function, and thus a divide-and-conquer scheme, is only possible if the imaginary axis is enclosed by the spectrum again, we develop a spectral shifting strategy. This results in the *Divide-Shift-and-Conquer* algorithm to compute the generalized Schur decomposition.

The second deflating subspace computation method, shown in Section 3.3, is based on the matrix disc function

$$[U,\ V] = \operatorname{disc}(A, B), \tag{1.8}$$

which was introduced by Roberts [144] for a matrix A. Its extension to a matrix pair (A, B) was developed by Malyshev [121], Bai et al. [15], and Benner [21]. The matrix disc function can be used to compute a deflating subspace, which either corresponds to the eigenvalues lying outside the unit circle or inside it. The recursive application of the matrix disc function leads to a divide-and-conquer scheme, which uses eigenvalue scaling to enclose the unit circle again. In this way we obtain the *Divide-Scale-and-Conquer* algorithm. The main advantage of the matrix disc function, over the generalized matrix sign function, is that its iterative computation scheme is free of solving linear systems and computing matrix inverses. A disadvantage is the higher computational cost.

Chapter 4 The implementation and optimization of both algorithms is subject of Chapter 4. Before details of the Divide-Shift-and-Conquer and the Divide-Scale-and-Conquer scheme are discussed, common operations for both algorithms are presented in Section 4.1. The main issue is the rearrangement of the recursive scheme into an iterative one. This enables us to limit the overall memory consumption and provides strategies to manage the available memory. Furthermore, we present a hardware oriented stopping criterion for the iteration [27, 26]. Finally, the section covers common operations like matrix-matrix products and gives an overview about the different QR decompositions used for deflating subspace computations [42, 43, 49, 143, 125].

The efficient computation of the generalized matrix sign function using a Newton method is in the focus of Section 4.2. After recalling how the naive approach [77] is implemented using routines from BLAS and LAPACK, we discuss two strategies to reduce the operation count. First, a QR or QL decomposition based approach by Benner and Quintana-Ortí [31] is evaluated and adjusted to available routines in BLAS and LAPACK. Second, we regard a bi-diagonal reduction approach by Sun and Quintana-Ortí [161], which turns out to have a very reduced operation count, but does not lead to a satisfying performance on current computer architectures, as examples in Section 4.2.3 show. Based on the second idea, we develop a band reduction approach, which leads to a similar low operation count and respects the capabilities of current computer architectures. All regarded algorithms to compute the generalized matrix sign function have in common, that they require the solution of linear systems with many right-hand sides, or the inverse of a matrix. Therefore, we recall the Gauss-Jordan Elimination [142] scheme as an alternative to the LU decomposition. We extend the Gauss-Jordan Elimination to solve linear systems directly and show how multiple GPUs can be used to accelerate this approach. Finally, the Gauss-Jordan Elimination procedure integrates in the different variants of the Newton method to compute the generalized matrix sign function.

The matrix disc function is computed using the inverse free iteration [121, 15, 21], where the QR decomposition is the main operation. In Section 4.3.1, we focus on a naive implementation. Thereby, we regard the influence of different QR decompositions, like the compact storage QR decomposition [40], or the Tile-QR decomposition [51], on the runtime. In order to reduce the number of required operations in the inverse free iteration, we regard the efficiency improving technique by Marqués et al. [124]. Although this approach reduces the operation count and involves level-3 BLAS operations, it breaks with the capabilities of current computer architectures. Using directed acyclic graph scheduling instead, we derive a well-scaling multi-core aware formulation of the Marqués et al. idea in Section 4.3.2. Since the graph scheduling approach is not applicable to GPU accelerators with common compiler features and the OpenMP 4 programming model, it is only used on multi-core CPUs. On GPU accelerated systems, we focus on acceleration of the compact storage QR decomposition instead. Thereby, different efficiency improving strategies, like the communication-avoiding QR (CAQR) decomposition [59] and the Householder vector reconstruction [17], play and important role to overcome the performance issues of the QR decomposition.

Chapter 5 In Chapter 5, the first part of the thesis is closed by numerical results for the Divide-Shift-and-Conquer and the Divide-Scale-and-Conquer scheme. Before the approximation of the generalized Schur decomposition is evaluated, all important building blocks, identified in Chapter 4, are benchmarked and the important runtime parameters are determined. In order to easily adjust these parameters, we integrate the embedded scripting language LUA [95] into the code. It is used to evaluate tuning parameters at runtime. A comparison to the QZ algorithm shows that we obtain a sufficiently good approximation to the generalized Schur decomposition in a fraction of the runtime. Without the usage of GPUs we gain a factor of up to 17 in the runtime with the Divide-Shift-and-Conquer algorithm compared to the QZ algorithm from LAPACK on the already mentioned Intel® Skylake system. Since the Divide-Scale-and-Conquer algorithm is computationally more demanding, we only achieve an acceleration of up to a factor of 10 in the runtime. If GPU accelerators are available we obtain a speed up of up to a factor of 55 in case of the Divide-Shift-and-Conquer algorithm and up to a factor of 28 in case of the Divide-Scale-and-Conquer scheme.

Chapter 6 Chapter 6 opens the second part of the thesis, which deals with the efficient direct solution of Sylvester-type matrix equations. Using the generalized Schur decomposition, the generalized Sylvester equation can be transformed such that all coefficient matrices are of (quasi) upper triangular shape. Therefore, it is sufficient to consider direct solvers for Sylvester equations with (quasi) upper triangular coefficient matrices.

Before we develop our solvers we recall existing algorithms, like the Bartels-Stewart algorithm [19], the Gardiner et al. approach [78], the one- and two-solve schemes by Sorensen and Zhou [152], and the recursive blocking by Jonsson and Kågström [100, 101]. Beside these direct solvers we briefly discuss a generalized matrix sign function based solver [31] due to its close connection to Chapters 3 and 4. High performance implementations for distributed memory systems of this algorithm are already presented by Benner and Quintana-Ortí [32, 33, 30]. On top of the ideas of the existing solvers we develop a level-3 BLAS algorithm. With an experiment-driven approach, we identify the critical points, where the level-3 approach needs special treatment to cover the capabilities of the hardware and the Fortran compilers. The included optimizations are designed to coincide with the capabilities of current computer architectures. Furthermore, the design of the code focuses on building blocks, which can be turned into highly efficient machine code by current Fortran compilers. Beside the level-3 BLAS approach we apply directed acyclic graph scheduling, also used in the QR decomposition for the matrix disc function before, to the Sylvester-type matrix equations as well. This results in an algorithm, whose performance is independent of the parallelization features of the used BLAS implementation. The connection between the developed level-3 approach and the existing solver is given by Theorem 6.8. It shows that many of the solvers that are built on top of the Bartels-Stewart approach [19] are equal to our level-3 approach with a special set of block size parameters.

Due to the fact that both divide-and-conquer algorithms to compute the generalized Schur decomposition only result in a sufficient approximation, the transformation to (quasi) upper triangular coefficient matrices leads to a slightly perturbed equation. This perturbation is minimized or compensated with the help of iterative refinement [174, 128, 156], known from standard linear systems. Thereby, the generalized Schur decomposition of the coefficient matrices is only required once.

Chapter 7 The numerical results for the Sylvester-type matrix equations close the thesis. First, we determine the optimal block size parameters for all covered equations. This includes the generalized Sylvester equation (1.4) and the generalized Lyapunov equation (1.6). This incorporates special cases where some of the coefficient matrices are set to the identity. Using these optimal parameters, we compare the performance of our implementation to RECSY [100, 101], which figured out to be the fastest one of the existing approaches. The parallel capabilities of the used BLAS implementation already results in a runtime gain of a factor between 1.86 and 2.42 on the Intel® Skylake system for generalized Sylvester equations with (quasi) triangular coefficient matrices. The explicitly parallelized algorithm with directed acyclic graph scheduling results in a runtime gain of a factor between 3.06 and 6.38. The iterative refinement procedure to reduce the influence of the approximated generalized Schur decomposition is evaluated with the help of the generalized Lyapunov equation (1.6). Solving the generalized Lyapunov equation corresponding to the aforementioned steel profile example [34] leads to the following results. We only need three steps in the refinement procedure and obtain an overall speed up of 21.25 with a non-GPU version of the Divide-Shift-and-Conquer algorithm compared to the existing implementation in SLICOT [149, 140]. With GPUs enabled, we obtain a speed up of 55.67. In case of the Divide-Scale-and-Conquer method, we only need two steps in the iterative refinement process. This results in a speed up of 9.87 without GPUs and 32.39 with GPUs. For both options we obtain comparable forward errors and relative residuals with respect to the implementation in SLICOT.

1.3. Target Computer Architectures

In the thesis, three different computer systems are used for the evaluations of the algorithms and ideas. The used systems reflect typical compute nodes of high performance clusters from 2014 to 2019 [168]. All systems are equipped with two multi-core CPUs, where both have their own memory controller. Furthermore, the systems are assembled with one or two GPU accelerators, since the portion of accelerator-enabled systems in the Top-500 systems increased from 3.4% in 2010 to 29% in 2019 [168].

In order to cover a wide range of architectures, three different CPUs are considered in this thesis: Intel® Xeon® CPUs with the Haswell and Skylake mirco-architecture and the IBM POWER8 CPUs.

All three architectures are common CPUs for the nodes in the Top-500 cluster systems as well as they are used as standalone compute servers in various kinds of research. Regarding the GPUs we only focus on Nvidia® accelerators, since they are used in 136 of the 145 accelerator-enabled systems in the Top-500 in 2019. Namely, we use Nvidia® K20 and Nvidia® P100 accelerators. The details of the used hardware are given in Section 5.1.1.

Since the typical workflow in the design of high performance algorithms is to have an algorithm running optimal on one node first [84], we only focus on shared memory parallelization techniques in this thesis. An extension of the algorithms to distributed memory systems, such as HPC clusters in the Top-500, should be part of future research.

Although we focus on compute servers and nodes of HPC systems, most of the results are valid for notebooks, desktop computers, and workstations as well. Many of these devices are equipped with CPUs having a similar or the exact micro-architecture as the one in the compute servers. Today, even consumer devices are mostly equipped with accelerator devices, mostly in terms of a real graphic processor. Furthermore, problems, that result from unfavorable memory accesses, affect these systems as well, since the origin of the problem is inherent to the used CPU and memory architectures. Basically, this does not change from small consumer devices to compute servers. The developed algorithms employ many tuning parameters such that they can be easily adjusted to a new architecture by running a set of provide benchmarks and setting the proper parameters. This benchmark driven auto-tuning approach is already known from other mathematical software projects like the ATLAS BLAS implementation [173].

1.4. Related own Publications

Previously developed ideas that lead to this thesis have been published before in the articles and proceedings listed below.

Chapter 3, in particular Section 3.2, extends

[27]: P. Benner, M. Köhler, and J. Saak, *Fast approximate solution of the non-symmetric generalized eigenvalue problem on multi-core architectures*, in Parallel Computing: Accelerating Computational Science and Engineering (CSE), M. Bader, A. B. H.-J. Bungartz, M. Gerndt, G. R. Joubert, and F. Peters, eds., vol. 25 of Advances in Parallel Computing, IOS Press, 2014, pp. 143–152.

Section 4.1.4 includes the ideas from

[26]: P. Benner, M. Köhler, and J. Saak, *A cache-aware implementation of the spectral divide-and-conquer approach for the non-symmetric generalized eigenvalue problem*, Proc. Appl. Math. Mech., 14 (2014), pp. 819–820.

Section 4.2.4 summarizes

[111]: M. Köhler, C. Penke, J. Saak, and P. Ezzatti, *Energy-aware solution of linear systems with many right hand sides*, Comput. Sci. Res. Dev., 31 (2016), pp. 215–223.

[114]: M. Köhler and J. Saak, *GPU Accelerated Gauss-Jordan Elimination on the OpenPOWER platform – A case study*, Proc. Appl. Math. Mech., 17 (2017), pp. 845–846.

[115]: M. Köhler and J. Saak, *Frequency scaling and energy efficiency regarding the gauss-jordan elimination scheme with application to the matrix-sign-function on openpower 8*, Concurrency and Comput.: Pract. Exper., (2018).

Chapter 6 has its origin in

[112]: M. Köhler and J. Saak, *On GPU acceleration of common solvers for (quasi-) triangular generalized Lyapunov equations*, Parallel Comput., 57 (2016), pp. 212–221.

[113]: M. Köhler and J. Saak, *On BLAS level-3 implementations of common solvers for (quasi-) triangular generalized Lyapunov equations*, ACM Trans. Math. Software, 43 (2016), pp. 3:1–3:23.

CHAPTER
TWO

MATHEMATICAL BASICS OF THE GENERALIZED EIGENVALUE PROBLEM

Contents

The introduction has already mentioned that the solution of a generalized eigenvalue problem is necessary for the direct solution of dense Sylvester-type matrix equations. Therefore, this chapter is intended to give a brief overview of the mathematical basics behind the generalized eigenvalue problem, which are necessary for the derivation of our algorithms in Chapter 3. For additional information, like error estimation or perturbation theory, we refer to the literature [88, 155, 129, 80].

The generalized eigenvalue problem used in this work is given by the following definition:

Definition 2.1 (Generalized Eigenvalue Problem):
*We call $(A, B) \in \mathbb{C}^{m\times m} \times \mathbb{C}^{m\times m}$ a **matrix pair of order** m. The **characteristic polynomial** $\chi_{(A,B)}$ of the matrix pair (A, B) for $\gamma \in \overline{\mathbb{C}}$ is given by*

$$\chi_{(A,B)}(\gamma) := \det(A - \gamma B) \tag{2.1}$$

*and the **spectrum**, i.e., the **set of eigenvalues**, $\Lambda(A, B)$ of the matrix pair (A, B) is defined as*

$$\Lambda(A, B) := \left\{\lambda \in \mathbb{C} \mid \chi_{(A,B)}(\lambda) = 0\right\}. \tag{2.2}$$

If $\lambda \in \Lambda(A, B)$ and $x \in \mathbb{C}^m \setminus \{0\}$ is a nontrivial solution of the equation

$$Ax = \lambda Bx, \tag{2.3}$$

*then x is called a **(right) eigenvector** of (A, B) corresponding to the eigenvalue λ. The pair (λ, x) is referred to as **(right) eigenpair** of (A, B). A vector $y \in \mathbb{C}^m \setminus \{0\}$, which satisfies*

$$y^{\mathsf{H}}A = \lambda y^{\mathsf{H}}B \tag{2.4}$$

is said to be a ***left eigenvector*** *of (A, B) corresponding to the eigenvalue λ and (λ, y) is called a* ***left eigenpair*** *of (A, B).*

The determination of $\Lambda(A, B)$ by computing the roots of $\chi_{(A,B)}$ is not possible in general because direct solution formulas can only be derived for polynomials of degree less or equal to four [36]. Even if we have obtained $\Lambda(A, B)$ from $\chi_{(A,B)}$, the computation of the corresponding eigenvector will require to find a nontrivial solution of

$$(A - \lambda B)x = 0 \tag{2.5}$$

for each eigenvalue $\lambda \in \Lambda(A, B)$. Thereby, $A - \lambda B$ is singular by the definition of λ. If (A, B) is not well-structured, for example A is upper Hessenberg and B is upper triangular, this becomes a numerically tough task as well. For example, the algorithms implemented in LAPACK [10] rely on this property to compute the generalized eigenvectors.

Depending on the properties of the characteristic polynomial $\chi_{(A,B)}(\gamma)$ we divide the matrix pairs (A, B) into two classes using the following definition:

Definition 2.2 (e.g. [14]):

Let (A, B) be a matrix pair of order m. If there exists at least one $\lambda \in \mathbb{C}$ such that

$$\chi_{(A,B)}(\lambda) = \det(A - \lambda B) \neq 0, \tag{2.6}$$

the matrix pair (A, B) is said to be ***regular****. If the characteristic polynomial instead fulfills*

$$\chi_{(A,B)}(\lambda) = 0 \quad \forall \lambda \in \mathbb{C}, \tag{2.7}$$

the matrix pair (A, B) is said to be ***singular****.*

If the matrix pair is singular, Van Dooren [169] and Wilkinson [175] showed that the problem can be handled by splitting the matrix pair into a singular and a regular one and treat both of them separately. Furthermore, having a singular matrix pairs means that A and B have a common nontrivial null space, they are nonsingular as well and the spectrum of (A, B) contains the whole complex plane. This violates the solution conditions for the Sylvester type matrix equations covered in Chapter 6. For these reasons, we focus only on regular matrix pairs in this work. The solution of the generalized eigenvalue problem for singular matrix pairs is a separate area of research [61, 60, 175, 169].

Remark 2.3:

If B is invertible, then the generalized eigenvalue problem

$$Ax = \lambda Bx$$

is equivalent to the standard eigenvalue problem

$$B^{-1}Ax = \lambda x, \tag{2.8}$$

and $\Lambda(A, B) = \Lambda\left(B^{-1}A\right)$.

2.1. Invariant Subspaces and Deflation

The basic definition of the generalized eigenvalue problem shows that if we find an eigenpair (λ, x), x spans a distinguished one dimensional subspace in $\mathbb{C}^m$. From the Equation (2.3) it follows:

$$\operatorname{span}(Ax) = \operatorname{span}(Bx), \tag{2.9}$$

which means that the images of x under A and B lie in the same one dimensional subspace. Especially, we can conclude that

$$\operatorname{span}(Ax + Bx) = \operatorname{span}(Ax). \tag{2.10}$$

The generalization of this special subspace to higher dimensional subspaces of $\mathbb{C}^m$ is defined as follows:

Definition 2.4:
*Let (A, B) be a matrix pair of order m and let $\mathcal{Z}$ and $\mathcal{Q}$ be p-dimensional subspaces of $\mathbb{C}^m$. Then $\mathcal{Z}$ is called **right deflating subspace** of (A, B) if*

$$\dim\left(A\mathcal{Z} + B\mathcal{Z}\right) \leq p \tag{2.11}$$

*and $\mathcal{Q}$ is called **left deflating subspace** if*

$$\dim\left(A^{\mathsf{H}}\mathcal{Q} + B^{\mathsf{H}}\mathcal{Q}\right) \leq p \tag{2.12}$$

holds.

Remark 2.5:
If the matrix pair (A, B) is regular, equality holds in (2.11) *and* (2.12) *[155, 129].*

Based on this definition, Stewart [155] proved the following theorem concerning the connection between a given right deflating subspace and a block decomposition of the corresponding eigenvalue problem.

Theorem 2.6 (Deflation-Theorem [155, Theorem 2.1]):
Let (A, B) be a matrix pair of order m and $\mathcal{Z}$ be a p-dimensional right deflating subspace of (A, B). Then there exist unitary matrices Q and Z such that the first p columns of Z span the subspace $\mathcal{Z}$ and

$$Q^{\mathsf{H}}AZ = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} \quad \text{and} \quad Q^{\mathsf{H}}BZ = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} \tag{2.13}$$

holds, where A_{11} and B_{11} are of order p. The right eigenvectors $x \in \mathbb{C}^m \setminus \{0\}$ belonging to $\lambda \in \Lambda\left(A_{11}, B_{11}\right)$ fulfill

$$x \in \mathcal{Z}. \tag{2.14}$$

Corollary 2.7 ([162, Theorem 2.3, Corollary 2.4, Corollary 2.5]):
Let (A, B), Q, Z, and $\mathcal{Z}$ be given as in Theorem 2.6. Then we conclude:

1. *The first p columns of Q span a p-dimensional left deflating subspace $\mathcal{Q}$ corresponding to $\Lambda\left(A_{11}, B_{11}\right)$.*
2. *The subspace spanned by the last $(m-p)$ columns of Q is the orthogonal complement of $A\mathcal{Z} + B\mathcal{Z}$.*
3. *The set of eigenvalues $\Lambda\left(A, B\right)$ deflates into*

$$\Lambda\left(A, B\right) = \Lambda\left(A_{11}, B_{11}\right) \cup \Lambda\left(A_{22}, B_{22}\right). \tag{2.15}$$

4. *$\Lambda\left(A, B\right)$ is independent of (A_{12}, B_{12}).*

From the previous corollary together with Theorem 2.6 we conclude that the solution of the generalized eigenvalue problem (2.2) can be replaced by two smaller problems (2.15) if a deflating subspace $\mathcal{Z}$ is known. This introduces additional costs of only four matrix-matrix products (2.13). This fundamental result is used in Chapter 3 to derive a family of divide-and-conquer algorithms.