UMEÅ UNIVERSITY

Improving the Efficiency of Eigenvector-Related Computations

Angelika Beatrix Schwarz

Doctoral Thesis, May 2021
Department of Computing Science
Umeå University
Sweden

Dissertation for PhD

ISBN 978-91-7855-576-5 (print)
ISBN 978-91-7855-577-2 (pdf)
ISSN 0348-0542
UMINF 21.05

Printed by Cityprint i Norr AB, Umeå, 2021.

Dies diem docet.

—Publilius Syrus, *Sententiae*

Abstract

An effective strategy in dense linear algebra is the design of algorithms as tiled algorithms. Tiled algorithms that express the bulk of the computation as matrix–matrix operations (level-3 BLAS) have proven successful in achieving high performance on cache-based architectures. At the same time, tiled algorithms interoperate with dynamic data-driven execution models such as task parallelism and promise good parallel scalability.

This thesis applies the concept of tiled algorithms and task-centric execution to algorithms related to the computation of eigenvectors for the dense, non-symmetric eigenvalue problem. First, a standard algorithm for computing eigenvectors from the Schur form is recast such that all computational steps are rich in matrix–matrix operations. Second, inverse iteration on the Hessenberg matrix as an alternative approach to computing eigenvectors is addressed. An existing algorithm is revised to express the computationally most expensive step with matrix–matrix operations. Third, a task-parallel, tiled triangular Sylvester equation solver is amended to solve a larger class of problems. All algorithms have an enhanced performance, which is demonstrated through numerical experiments.

Enkel sammanfattning på svenska

Egenvärden och egenvektorer spelar en roll i ett stort antal discipliner som till exempel finansiell matematik, mekanik, statik och reglerteknik. Kunskap om egenvärden och egenvektorer kan ge en bättre förståelse av applikationernas fysikaliska egenskaper. Egenvärden och egenvektorer gör det möjligt att exempelvis analysera dynamiska system. För att konstruera jordbävningssäkra byggnader används ofta databaserad egenvärdesanalys. Ett praktiskt angreppssätt är därför att skapa matematiska modeller, ställa upp ekvationssystem och analysera matrisens egenvärden och egenvektorer. Beräkning av egenvärden och egenvektorer är ett klassiskt problem i numerisk linjär algebra.

Denna avhandling består av två delar. Den första delen vidareutvecklar och förbättrar två existerande algoritmer för numerisk beräkning av egenvektorer när matrisen är reellvärd och osymmetrisk. De reviderade algoritmerna beräknar egenvektorerna på ett mer effektivt, parallellt sätt på datorsystem med en minneshierarki. Den första algoritmen beräknar egenvektorerna av en matris utgående från dess reella Schurfaktorisering. Algoritmen uttrycker alla beräkningssteg som blockoperationer, i synnerhet matrismultiplikationer, och når därmed en förbättrad effektivitet och skalbarhet. Den andra algoritmen utgår från en Hessenbergmatris. Givet att egenvärdena är kända kan de associerade egenvektorerna approximeras med inversiteration. I de flesta fall når inversiteration konvergens redan efter en iteration, så att ett skiftat linjärt Hessenbergsystem löses för varje egenvektor. Den nya algoritmen reviderar den så kallade RQ-metoden för att lösa ett skiftat Hessenbergsystem så att den beräkningsintensiva delen utförs för flera egenvektorer samtidigt och med blockoperationer. Förbättringarna jämfört med de ursprungliga algoritmerna visas med numeriska experiment.

Den andra delen behandlar den triangulära Sylvesterekvationen. Den triangulära Sylvesterekvationen är ett delproblem när algoritmen av Bartels och Stewart används för att lösa den allmänna Sylvesterekvationen. Sylvesterekvationer spelar en roll exempelvis inom reglerteorin. Denna avhandling förbättrar en existerande parallell blockbaserad lösare för den triangulära Sylvesterekvationen genom åtgärder som undviker flyttalsöversvämning. Detta medför att klassen av de lösbara problemen förstoras. Numeriska experiment visar att den nya lösaren når ungefär samma effektivitet som den ursprungliga implementationen.

Populärwissenschaftliche Zusammenfassung

Eigenwerte und Eigenvektoren haben eine Bedeutung in einer Vielzahl von Gebieten wie der Finanzmathematik, der Mechanik, der Statik oder der Regelungstechnik. Die Analyse der Eigenwerte und Eigenvektoren kann Aufschluss über physikalische Eigenschaften der Anwendung geben. So spielen Eigenwerte und Eigenvektoren beispielsweise eine Rolle bei der Stabilitätsanalyse dynamischer Systeme oder der Analyse von Gebäuden bezüglich ihrer Erdbebensicherheit. Eine etablierte Vorgehensweise ist daher die Erstellung eines mathematischen Modells und die Analyse dessen mit Hilfe von Eigenwerten und Eigenvektoren. Die Berechnung der Eigenwerte und Eigenvektoren ist ein klassisches Problem der numerischen linearen Algebra. Diese Promotionsschrift beschäftigt sich mit dem speziellen Fall, dass die Systemmatrix des mathematischen Modells reellwertig und nicht symmetrisch ist.

Der erste Beitrag dieser Arbeit ist die Verbesserung zweier existierender Algorithmen zur numerischen Berechnung von Eigenvektoren. Die überarbeiteten Algorithmen ermöglichen eine effizientere, parallele Berechnung auf Rechnersystemen mit einer Speicherhierarchie. Der erste Algorithmus berechnet die Eigenvektoren einer Matrix ausgehend von deren reellen Schur-Zerlegung. Er erreicht eine höhere Performanz und bessere Skalierbarkeit, indem alle Berechnungsschritte als Blockoperationen, insbesondere Matrizenmultiplikationen, ausgedrückt werden. Der zweite Algorithmus geht von der Hessenbergmatrix aus. Unter der Annahme, dass die Eigenwerte bekannt sind, können die zugehörigen Eigenvektoren durch inverse Iteration approximiert werden. In den meisten Fällen erreicht das Verfahren der inversen Iteration Konvergenz nach einer Iteration, sodass pro Eigenvektor lediglich ein lineares Hessenbergsystem mit Shift gelöst wird. Der neue Algorithmus revidiert den RQ-Ansatz für das Lösen eines linearen Hessenbergsystems mit Shift, sodass trotz Shift-spezifischer RQ-Zerlegungen der berechnungsintensive Teil des Algorithmus für mehrere Eigenvektoren gleichzeitig und mit Blockoperationen berechnet werden kann. Die Verbesserung gegenüber den originalen Algorithmen wird durch numerische Experimente nachgewiesen.

Der zweite Beitrag dieser Arbeit beschäftigt sich mit der triangulären Sylvestergleichung. Diese ist ein Teilproblem, welches in der Lösung der generellen Sylvestergleichung mittels des Bartels-Stewart-Algorithmus erscheint. Sylvestergleichungen haben Bedeutung zum Beispiel in der Kontrolltheorie. Diese Arbeit erweitert einen parallen, blockbasierten Löser für die trianguläre Sylvestergleichung, sodass die Klasse lösbarer Probleme vergrößert wird. Numerische Experimente weisen nach, dass die Kosten dieser Erweiterung vernachlässigbar sind und die Performanz des originalen Ansatzes nicht signifikant beeinträchtigen.

Preface

This thesis is based on the following papers.

Paper I	Carl Christian Kjelgaard Mikkelsen, Angelika Beatrix Schwarz and Lars Karlsson. Parallel robust solution of triangular linear systems. In *Concurrency and Computation: Practice and Experience*, e5064, Wiley Online Library, 2018.
Paper II	Angelika Schwarz and Lars Karlsson. Scalable Eigenvector Computation for the Non-Symmetric Eigenvalue Problem. In *Parallel Computing*, volume 85, 131–140, Elsevier, 2019.
Paper III	Angelika Schwarz, Carl Christian Kjelgaard Mikkelsen and Lars Karlsson. Robust parallel eigenvector computation for non-symmetric eigenvalue problem. In *Parallel Computing*, volume 100, 102707, Elsevier, 2020.
Paper IV	Angelika Schwarz. Robust level-3 BLAS Inverse Iteration from the Hessenberg Matrix. Technical report arXiv:2101.05063, 2021.
Paper V	Angelika Schwarz, Carl Christian Kjelgaard Mikkelsen. Robust Task-Parallel Solution of the Triangular Sylvester Equation. In *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, vol 12043, Springer, 2020.

Funding and Resources

Financial support for the papers I, II, III and V is provided by the European Union's Horizon 2020 research and innovation programme under grant agreement No 671633 (NLAFET). Support has also been received by eSSENCE, a collaborative e-Science programme funded by the Swedish Government via the Swedish Research Council (VR). The author is grateful to the Department of Computing Science at Umeå University for providing financial support during the work on Paper IV.

The computational experiments were enabled by resources by the Swedish National Infrastructure for Computing (SNIC) at High Performance Computing Center North (HPC2N), Umeå, partially funded by the Swedish Research Council through grant agreement no. 2018-05973. SNIC provided compute hours under the grants SNIC 2017/1-617 (Paper II), SNIC 2018/5-41 (Papers I, III, V), SNIC 2018/3-624 (Paper I, III), SNIC 2018/5-6 (Paper I), SNIC 2019/5-29 (Paper IV) and SNIC 2019/3-311 (Paper IV).

Acknowledgements

Finally, my doctoral studies under the supervision of Lars Karlsson, Bo Kågström and Carl Christian Kjelgaard Mikkelsen end. It has been a long journey. I am grateful to all of you who provided encouragement, constructive criticism, or friendly advice. I thank my reference person Martin Berggren for his guidance during my studies.

This final stage would not have been reached without the assistance by numerous people. I thank my colleagues for creating an amazing working environment. Without you, my time in Umeå would have looked totally differently. I will smile when I think back of the cheerful moments with you. Special thanks go to Eddie Wadbro for providing unconditional support and invaluable advice.

I thank my parents for their patience and immeasurable support throughout my studies.

Contents

CHAPTER 1

A Motivational Example

Eigenproblems occur in numerous fields including mechanics, finance, data science and control theory. This very first chapter aims at highlighting the value and insights that eigenvalues and eigenvectors can provide with a concrete example, a damped vibrating system.

We study the mechanical vibrations of the mass-spring-damper system depicted in Figure 1.1. The system comprises two masses m_1 and m_2. The mass m_2 is connected by a spring k_2 and a damper c_2 to the mass m_1. The mass m_1 is attached to a fixed point by a spring k_1 and a damper c_1. The masses are assumed to move without friction only in the horizontal direction. Hence, the (time-dependent) position coordinates of the two masses x_1 and x_2 fully describe the motion of the system and are the two degrees of freedom of this system. The analysis of this example loosely follows [18, Sec. 6.7.2, Sec. 6.8.2].

The equations of motion can be obtained by applying Newton's second law of motion to each of the masses. The corresponding free body diagram is shown in Figure 1.2. The spring k_1 is under tension for a positive value of x_1; the spring

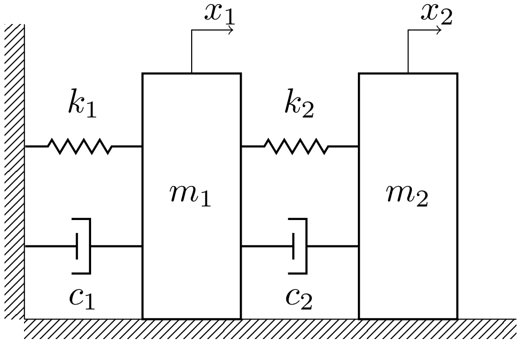


Figure 1.1: Simple model of a mass-spring-damper system with two degrees of freedom (x_1, x_2).

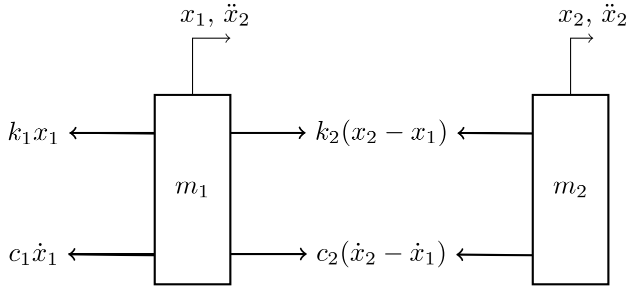


Figure 1.2: Free body diagram corresponding to Figure 1.1.

k_2 is under tension whenever $(x_2 - x_1)$ is positive. The resulting equations are

$$\begin{aligned} m_1\ddot{x}_1 &= k_2(x_2 - x_1) + c_2(\dot{x}_2 - \dot{x}_1) - k_1 x_1 - c_1\dot{x}_1 \\ m_2\ddot{x}_2 &= -k_2(x_2 - x_1) - c_2(\dot{x}_2 - \dot{x}_1). \end{aligned}$$

Here $x_j = x_j(t)$ is the displacement, $\dot{x}_j = \dot{x}_j(t) = dx_j/dt$ is the velocity and $\ddot{x}_j = \ddot{x}_j(t) = d^2x_j/dt^2$ is the acceleration for $j = 1, 2$.

This set of two second-order ordinary differential equations is converted into a first-order system. We define $u = \begin{bmatrix} x_1 & \dot{x}_1 & x_2 & \dot{x}_2 \end{bmatrix}^T$ and obtain

$$\underbrace{\begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \\ \dot{u}_4 \end{bmatrix}}_{\dot{u}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ -(k_1+k_2)/m_1 & -(c_1+c_2)/m_1 & k_2/m_1 & c_2/m_1 \\ 0 & 0 & 0 & 1 \\ k_2/m_2 & c_2/m_2 & -k_2/m_2 & -c_2/m_2 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{u}. \tag{1.1}$$

The assumption that a solution to (1.1) has the form $u = ve^{\lambda t}$, where λ is an eigenvalue and $v \neq 0$ is an eigenvector of the matrix A, leads to the standard non-symmetric eigenvalue problem $Av = \lambda v$. Eigenvalues and eigenvectors give insight into properties of the mass-spring-damper system as we will demonstrate with a concrete example.

Suppose $m_1 = 10\,\text{kg}$, $m_2 = 2.5\,\text{kg}$, $k_1 = 400\,\text{N/m}$, $k_2 = 200\,\text{N/m}$, $c_1 = 10\,\text{kg/s}$, $c_2 = 5\,\text{kg/s}$. Moreover, the initial displacements are $x_1(0) = 0.5\,\text{m}$ and $x_2(0) = 1\,\text{m}$, and the initial velocities are $\dot{x}_1(0) = \dot{x}_2(0) = 0\,\text{m/s}$. The corresponding matrix is

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -60 & -1.5 & 20 & 0.5 \\ 0 & 0 & 0 & 1 \\ 80 & 2 & -80 & -2 \end{bmatrix}.$$

Using the Matlab command `eigs(A)`, we compute the eigenvalues and eigenvectors of A. We observe that the four eigenvalues of A occur in two complex conjugate pairs. If we confine us to those eigenvalues with a positive

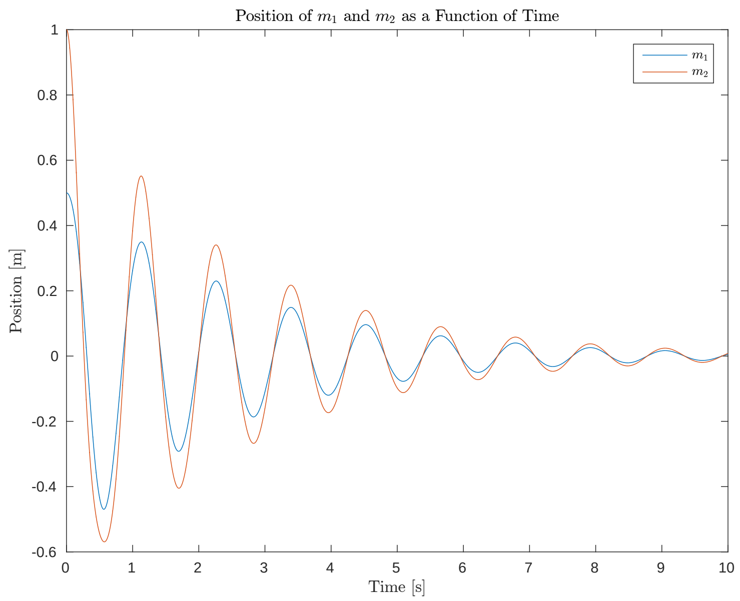


Figure 1.3: Displacement of the masses m_1 and m_2 over time.

imaginary part $\lambda_j = -a_j + ib_j$, $j = 1, 2$, we obtain the eigenvalue matrix $\Lambda = \operatorname{diag}(-0.3596 + 5.3516i, -1.3904 + 10.4546i)$ and the eigenvector matrix

$$V = \begin{bmatrix} -0.0066 & 0.0045 \\ 0.5302 & -0.3620 \\ -0.0103 & -0.0116 \\ 0.8279 & 0.9274 \end{bmatrix} + \begin{bmatrix} -0.0986 & 0.0340 \\ 0 & 0 \\ -0.1540 & -0.0872 \\ 0 & 0 \end{bmatrix} i$$

satisfying

$$AV = V\Lambda. \tag{1.2}$$

The eigenvalues give insight into the oscillatory motion of the mass-spring-damper system. The imaginary part b_j provides information about the damped natural frequency. For our system, there are two damped natural frequencies, 5.3516 rad/s (0.8517 Hz) and 10.4546 rad/s (1.6639 Hz). The real part $-a_j$ represents the decay rate and leads to the damping ratio $\zeta_j = -a_j/\omega_j$, where $\omega_j = |\lambda_j|$. The damping ratios are 0.0670 and 0.1318, respectively, and reveal that the system is underdamped. This is reflected in Figure 1.3, where the system dissipates energy with every overshoot such that the oscillations tend to zero.

The eigenvectors provide information about the movement of the masses. Recall that the first and the third component of u are the displacements of

m_1 and m_2. Hence, we focus on the first and the third row of the eigenvector matrix V. The comparison of the real part and the imaginary part of $v_{11} = -0.0066-0.0986i$ and $v_{31} = -0.00103-0.1540i$ (corresponding to the eigenvalue $-0.3596 + 5.3516i$) shows that the real parts have identical signs and that the imaginary part is norm-wise larger than the real part. The physical meaning is that the masses move in phase. Based on visual inspection, this oscillation mode is excited in Figure 1.3. Regarding the eigenvalue $-1.3904+10.4546i$, the eigenvector components $v_{12} = 0.0045 + 0.0340i$ and $v_{32} = -0.0116 - 0.0872i$ have opposite signs. This suggests that the masses move out of phase.

This example has demonstrated how eigenvalues and eigenvectors can provide insight into the properties of a model system. Naturally, mass-spring-damper systems with more components generate larger matrices, which can be analyzed by means of the underlying eigenvalues and eigenvectors. Larger matrices imply an increased computational demand, which motivates the usage of efficient and parallel routines.

In this example, the eigenproblem was solved with the Matlab command `eigs`, which hides how the eigenproblem is solved internally. This thesis focuses on algorithms related to the eigenvector computation. The presented algorithms could be used for obtaining the eigenvectors as one of the subproblems that `eigs` solves.

CHAPTER 2

High-Performance Linear Algebra Software

The previous chapter showed that eigenvalues and eigenvectors can provide valuable information to domain experts. Domain experts can, in addition, benefit from library software, which in the previous chapter was hidden under the command `eigs`. Library software promises the use of well-optimized, reliable routines as back end. As a result, domain experts can focus on the underlying science rather than spending time on developing and tuning all code by themselves.

The concept of library software is established in the domain of numerical linear algebra. Numerous libraries like for instance LAPACK [2], SLICOT [5], and libflame [44] aim at providing high-quality implementations of linear algebra routines. A library targeting the solution of non-symmetric eigenproblems is StarNEig [38].

BLAS. An important library in dense linear algebra is the BLAS (Basic Linear Algebra Subprograms). Originally, the BLAS were a collection of vector-vector operations (level-1 BLAS) [36]. Later on, the BLAS was extended to matrix–vector operations (level-2 BLAS) [14] and matrix–matrix operations (level-3 BLAS) [13, 6]. The use of level-3 BLAS promises attaining high performance. Level-3 BLAS operations process $\mathcal{O}(n^3)$ operations on $\mathcal{O}(n^2)$ data. The $\mathcal{O}(n)$ ratio of computation/data allows level-3 BLAS operations to be implemented such that data is reused. By reusing data in the cache, the latency of main memory accesses can be hidden. A particularly important level-3 BLAS operation is the matrix–matrix product (`xGEMM`). An effective strategy to increase the performance has been the redesign of algorithms such that the bulk of the computation corresponds to matrix–matrix multiplications [33]. As a consequence, the algorithms can benefit from the performance of highly optimized implementations of `xGEMM` available in, for example, OpenBLAS [39], oneMKL [28] and Atlas [47].

Parallel Computing. A desirable property of linear algebra library software is the support of parallelism and, naturally accompanying, good parallel performance [9]. Parallel computing has become increasingly important since the uni-processor design faces the so-called power wall, the difficulty of increasing the clock frequency due to power dissipation limits [3, p. 1]. The power wall suggests that (future) performance gains primarily arise from an increase of the number of processors rather than a performance improvement of a single processor [3, p. 3]. A parallel execution uses multiple processing units for solving a problem. For this purpose, the problem is divided into smaller chunks that can be executed concurrently such that the total execution time is reduced. This leads to parallel performance as a measure of how efficiently software runs in parallel. In the ideal case, the parallel speedup is proportional to the amount of processing units in use.

Parallel computations require the involved processing units to cooperate. The cooperation includes the exchange of information between processing units. This exchange of information can be realized with shared memory or message passing. In shared memory, the processing units share the view on global memory. All processing units asynchronously read and write to the shared memory. Hence, there is no explicit communication of data. Mechanisms such as locks are needed to control memory accesses, for example, to ensure the correct memory access order when the outcome requires a specific order of read and write accesses. In message passing, the processing units have their own local data. Data is exchanged explicitly through passing messages. The processing units can reside on the same compute component or can be distributed across physically separated compute components. This parallel programming model is also known as a distributed memory model.

In parallel computations, the processing units may not be equal. This is particularly true for a CPU–GPU system. The involved processing units have different capabilities in processing certain tasks. Preferably, a parallel computation takes into account the different compute capabilities. An efficient hybrid computation requires load balancing and minimizes synchronization and communication overhead. A numerical library that targets heterogeneous CPU–GPU systems is MAGMA [12, 43].

Parallelism and BLAS in numerical libraries. Many numerical libraries have reformulated algorithms in a loop-based, recursive or hybrid way such that the algorithms are rich in matrix–matrix multiplications [15]. A common pattern, for example encountered in LAPACK, is the alternating repetition of a panel factorization and a trailing matrix update. The panel factorization processes a small part of the matrix, the so-called panel, and often relies on level-2 BLAS operations. The transformations computed during the panel factorization are accumulated. The trailing matrix update then applies the accumulated transformations to the remaining matrix using level-3 BLAS operations. Overall, the computation is rich in level-3 BLAS operations. The panel computation can be difficult to parallelize efficiently. The usage of parallel BLAS, however, trivially introduces parallelism to the trailing matrix updates. Then the computation alternates sequential panel factorizations and parallel trailing matrix updates in a so-called fork-join execution pattern.

Since the panel factorization can be a bottleneck in a parallel computation, libraries such as PLASMA [10] and libflame [44, Sec. 5.5] increase the level of parallelism by partitioning the matrices into tiles[1]. Tasks define chunks of work that operate on tiles. The computation is expressed as a Directed Acyclic Graph (DAG), whose nodes represent tasks and whose edges represent dependences between tasks. The DAG is processed task by task. Tasks with satisfied input dependences can be executed in parallel and asynchronously. The order in which tasks are executed is up to the scheduler of the executing runtime system. Based on the scheduling decisions, the runtime system handles the actual execution of the tasks and maintains data consistency.

The partitioning of a matrix into tiles leads to the notion of task granularity, the computational load per task. The task granularity influences the scheduling overhead, the efficiency at which each single task can be executed, and how many tasks are available for a parallel execution. Suited task granularities can be found by tuning. Tuning promises attaining high parallel performance while maintaining portability across different computer systems.

This thesis contributes algorithms related to the computation of eigenvectors. All these algorithms aim at rearranging the computation such that computation benefits from more efficient usage of the available computational resources.

[1] A synonym used in literature is *block*. This thesis calls larger contiguous submatrices related to matrix partitionings *tiles* in order to avoid confusion with 1-by-1 or 2-by-2 blocks occurring as structure in the algorithms in Section 3.1 and onwards.

Chapter 3

Eigenvector-Related Computations

In Chapter 1 eigenvalues and eigenvectors were computed with the Matlab command `eigs`. This chapter details algorithms that can be used for the computation of the eigenvectors of the standard eigenvalue problem.

3.1 Eigenvectors for the Standard Eigenvalue Problem

The standard eigenvalue problem concerns finding eigenpairs (λ_ℓ, x_ℓ) that satisfy

$$Ax_\ell = \lambda_\ell x_\ell \tag{3.1}$$

for a general matrix A. In the case that $A \in \mathbb{R}^{n\times n}$ is non-symmetric, the eigenvalues λ_ℓ are real or complex. An eigenvector $x_\ell \neq 0$ corresponding to λ_ℓ is, in accordance, real or complex. Despite potentially complex eigenvalues and eigenvectors, solutions to the standard eigenvalue problem can be computed using solely real arithmetic. Real arithmetic saves memory and floating-point operations (flops) compared to complex arithmetic, but makes the computation more complicated.

Equation (3.1) concerns only right eigenvectors. A left eigenvector satisfies $y_\ell^T A = \lambda_\ell y_\ell^T$. Hence, a left eigenvector of A is same as the transpose of a right eigenvector of A^T and therefore $A^T y_\ell = \lambda_\ell y_\ell$.

When many eigenpairs are sought, (3.1) can be written as the matrix equation

$$AX = X\Lambda. \tag{3.2}$$

In complex arithmetic, Λ is a diagonal eigenvalue matrix, in other words, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots)$. The columns of the complex matrix X are the corresponding right eigenvectors. If the computation is executed in real arithmetic, the matrix Λ is block diagonal with 1-by-1 or 2-by-2 blocks on the diagonal. A 1-by-1 block corresponds to a real eigenvalue; a 2-by-2 block corresponds to a complex conjugate pair of eigenvalues. The eigenvector matrix can be stored using real numbers if the complex eigenvectors are stored in interleaved storage, i.e., the real and the imaginary parts of a complex vector are stored as two adjacent columns.

Example: Eigenvalue equation in real arithmetic

Equation (1.2) from Chapter 1 can be expressed in real arithmetic as

$$A\begin{bmatrix} -0.0066 & -0.0986 & 0.0045 & 0.0340 \\ 0.5302 & 0 & -0.3620 & 0 \\ -0.0103 & -0.1540 & -0.0116 & -0.0872 \\ 0.8279 & 0 & 0.9274 & 0 \end{bmatrix} =$$

$$\begin{bmatrix} -0.0066 & -0.0986 & 0.0045 & 0.0340 \\ 0.5302 & 0 & -0.3620 & 0 \\ -0.0103 & -0.1540 & -0.0116 & -0.0872 \\ 0.8279 & 0 & 0.9274 & 0 \end{bmatrix}\begin{bmatrix} -0.3596 & 5.3516 & & \\ -5.352 & -0.3596 & & \\ & & -1.390 & 10.45 \\ & & -10.45 & -1.390 \end{bmatrix}.$$

Each complex eigenvalue appears as a 2-by-2 block in the block diagonal matrix. The first two columns of the eigenvector matrix correspond to the real and the imaginary part of the first eigenvector. The other two columns interleave the real and the imaginary parts of the second eigenvector.

We focus on the case when the non-symmetric A is dense. A standard method for computing all eigenvalues of A is the QR algorithm. In real arithmetic, the QR algorithm reduces A to the real Schur form $T = Q^TAQ$. Here, the matrix T is upper quasi-triangular, i.e., it is block upper triangular with 1-by-1 or 2-by-2 blocks on the diagonal. The matrix Q is orthogonal. LAPACK 3.9.0 computes the real Schur decomposition through a two-step procedure. In the first step, A is reduced to upper Hessenberg form $H = Q_0^TAQ_0$ with an orthogonal matrix Q_0. In the second step, H is reduced to the real Schur form $T = Q_1^THQ_1$ using the QR algorithm. Here, the matrix Q_1 accumulates all orthogonal similarity transformations applied in the iterative QR algorithm. The QR algorithm has been revised several times; a modern implementation is the multi-shift QR algorithm with aggressive early deflation [7, 8]. Multi-shift iterations and aggressive early deflation are two techniques that improve the convergence speed of the QR algorithm.

In the following, we address the computation of eigenvectors. There are two standard approaches, namely the computation of eigenvectors from the Schur form and the computation of eigenvectors from the Hessenberg matrix. Both approaches involve backward substitution, which is known to be prone to floating-point overflow. Hence, a mechanism to avoid overflow is necessary and discussed first. Afterwards, the eigenvector computations are presented.

Overflow. A common floating-point format for numerical computations is the binary 64 bit floating-point representation ("doubles") defined in the IEEE-754-2008 standard. The 64 bits decompose into 1 bit for the sign, 11 bit for the

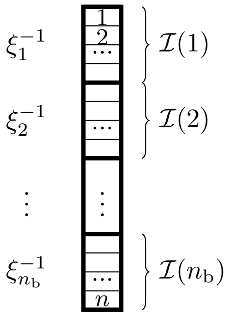


Figure 3.1: Mapping between index ranges $\mathcal{I}$ defined by tile indices $1, \ldots, n_\mathrm{b}$ and global indices $1, \ldots, n$. The vector segment with indices $\mathcal{I}(j)$ is associated with a local scaling factor ξ_j.

exponent and 52 bits for the mantissa. A normalized number is of the form

$$(-1)^{sign}(1.b_{52} \ldots b_1)_2 \cdot 2^{e-1023},$$

where the smallest biased exponent is $e = 1$ and the largest biased exponent is $e = 2046$. The largest biased exponent determines the overflow threshold as $\Omega = 2^{1023}$. Overflow occurs when an exponent is too large to be represented in the exponent field. In this case, the number is replaced with infinity (`inf`). As a consequence, the exact value is lost.

The solution of triangular systems $Tx = b$ is a key component in the computation of eigenvectors. Here T is a triangular matrix and b is a vector. Scaling both sides does not guarantee an overflow-free computation as demonstrated in [11, p. 9]. Anderson [1] therefore approaches an overflow-free computation through the solution of the scaled triangular linear system $Tx = \gamma b$. By virtue of the scaling factor $\gamma \in (0, 1]$ a computation that would otherwise overflow can be executed without infinities. This leads to robust algorithms. We call an algorithm robust if overflow cannot occur at any point during the computation. A robust algorithm for the solution of triangular systems computes the solution as the scalar γ and the vector x representing $\gamma^{-1}x$ such that no entry in x is `inf`. For this purpose, the vector x is scaled whenever an operation could exceed the overflow threshold. Once rescaled, the operation can be executed safely.

Anderson's approach to the robust solution of a triangular system with one right-hand side is realized in the LAPACK 3.9.0 routine `xLATRS`. The routine `xLATRS` evaluates based on upper bounds if overflow cannot occur during the computation. If so, the problem is reduced to the solution of $Tx = b$ and solved with `xTRSV`, the non-robust counterpart of `xLATRS`. If, however, overflow can occur, $Tx = \gamma b$ is solved such that overflow is avoided by dynamic scaling. The routine `xLATRS` is slower than `xTRSV`, but guarantees a result free of `inf`.

Example: Overflow (Paper I Sec. 8)

The system

$$\begin{bmatrix} 1 & -2 & & & \\ & 1 & -2 & & \\ & & \ddots & \ddots & \\ & & & 1 & -2 \\ & & & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

has the exact solution

$$x = \begin{bmatrix} 2^{m-1} \\ 2^{m-2} \\ \vdots \\ 2^1 \\ 2^0 \end{bmatrix}. \tag{3.4}$$

The component-wise relative condition of a linear system is given by Skeel's condition number [27], which is $2m - 1$ for (3.3). If $m = 1025$, the representation of the solution in double-precision arithmetic is

$$x = \begin{bmatrix} \texttt{inf} \\ 2^{1023} \\ \vdots \\ 2^1 \\ 2^0 \end{bmatrix}. \tag{3.5}$$

The first entry of x overflows. For (3.5), a scaled, overflow-free representation is

$$x = \left(\frac{1}{2}\right)^{-1} \begin{bmatrix} 2^{1023} \\ 2^{1022} \\ \vdots \\ 2^0 \\ 2^{-1} \end{bmatrix}. \tag{3.6}$$

Kjelgaard Mikkelsen and Karlsson [34] extend the robust solution of triangular systems to many right-hand sides. Each right-hand side is associated with a scaling factor. Hence, the scaled system $TX = B\Gamma$ is solved, where Γ is a diagonal matrix of scaling factors. The algorithm is designed such that the level-3 BLAS potential is preserved and the overhead from scaling events due to overflow avoidance is reduced. For this, each right-hand side is partitioned into segments. Each segment is associated with a local scaling factor. Figure 3.1

illustrates a tiled vector with local scaling factors. The usage of local scaling factors delimits scaling due to overflow avoidance. Scaling is applied only to those tiles that are involved in an operation rather than to the entire column. The global scaling Γ is computed in a postprocessing step. The global scaling factor γ for a solution x corresponds to the smallest local scaling factor. The vector segments are rescaled with respect to this global scaling factor, which yields a consistently scaled solution.

Example: Consistency scaling

The consistency scaling transforms the segment-wise scaled vector into a consistently scaled final solution.

$$\begin{array}{|c|c|c|}
\hline
\left(\frac{1}{4}\right)^{-1} & \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} & \\
\hline
\left(\frac{1}{2}\right)^{-1} & \begin{matrix} x_4 \\ x_5 \\ x_6 \end{matrix} & \\
\hline
\left(\frac{1}{1}\right)^{-1} & \begin{matrix} x_7 \\ x_8 \\ x_9 \end{matrix} & \\
\hline
\end{array}
\Rightarrow \left(\frac{1}{4}\right)^{-1}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \frac{1}{2}x_4 \\ \frac{1}{2}x_5 \\ \frac{1}{2}x_6 \\ \frac{1}{4}x_7 \\ \frac{1}{4}x_8 \\ \frac{1}{4}x_9 \end{bmatrix}$$

Overflow avoidance is required for the robust computation of eigenvectors. The next paragraphs discuss two standard approaches to computing the eigenvectors of a non-symmetric, dense matrix. For clarity, the ideas of the algorithms are discussed assuming complex arithmetic. Details relevant for executing the computation in real arithmetic are given separately.

Eigenvectors from the Schur form. Eigenvectors can be computed from the Schur decomposition $A = QTQ^H$ through a backward substitution followed with a backtransform [20, Sec. 7.6.4]. In complex arithmetic, the backward substitution phase computes a scaled eigenvector $\gamma^{-1}y = [\gamma^{-1}y_1, 1, 0]^T$ that satisfies

$$\begin{bmatrix} T_{11} & t_{12} & t_{13} \\ 0 & \lambda & t_{23} \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} \gamma^{-1}y_1 \\ 1 \\ 0 \end{bmatrix} = \lambda \begin{bmatrix} \gamma^{-1}y_1 \\ 1 \\ 0 \end{bmatrix}.$$

The purpose of the scalar $\gamma \in (0, 1]$ is the avoidance of overflow. In complex arithmetic, T_{11} is upper triangular and a solution to $(T_{11} - \lambda I)y_1 = -\gamma t_{12}$ can be obtained by robust standard backward substitution. Under the assumption that the backtransform computes $x = Q(\gamma^{-1}y)$ without overflow, the obtained vector x is an eigenvector of A.

In real arithmetic, T_{11} is upper quasi-triangular. The backward substitution requires a variation of backward substitution that can handle 2-by-2 blocks on the diagonal of T_{11}. In case of a complex conjugate pair of eigenvalues given as the 2-by-2 block $\begin{bmatrix} a & b \\ c & a \end{bmatrix}$ on the diagonal of T, the system

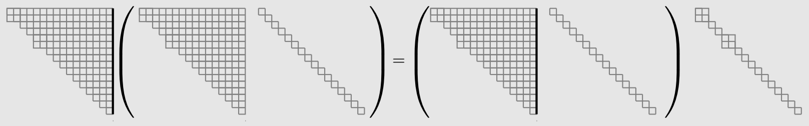
$$\begin{bmatrix} T_{11} & t_{12} & t_{13} & T_{14} \\ & a & b & t_{24}^T \\ & c & a & t_{34}^T \\ & & & T_{44} \end{bmatrix} \left(\gamma^{-1} \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \\ 0 & 0 \end{bmatrix} \right) = \left(\gamma^{-1} \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} a & \sqrt{|bc|} \\ -\sqrt{|bc|} & a \end{bmatrix}$$

is solved for a scaled complex eigenvector $\gamma^{-1}x = \gamma^{-1}(u + iv)$. The scalars u_2, v_2, u_3 and v_3 are set mutually dependent on each other and the system given by the first block row is solved. The storage of the real and the imaginary parts of a complex eigenvector as adjacent columns allows the linear updates and the backtransform to be executed entirely in real arithmetic.

When several eigenvectors, real or complex, are computed simultaneously, data can be reused, which introduces level-3 BLAS potential. The backward substitution then solves $T(X\Gamma^{-1}) = (X\Gamma^{-1})\Lambda$. Here Λ is block diagonal with 1-by-1 or 2-by-2 blocks and represents the eigenvalues. The eigenvectors X, one per eigenvalue block, are stored in interleaved storage. The diagonal matrix of scaling factors Γ avoids overflow such that the real and the imaginary part of complex eigenvectors are scaled alike.

Example: $T(X\Gamma^{-1}) = (X\Gamma^{-1})\Lambda$ in real arithmetic

The illustration shows the structure of the matrices when computing one eigenvector per block of a matrix T that has two 2-by-2 blocks and otherwise 1-by-1 blocks on the diagonal.



Eigenvectors from the Hessenberg matrix. Eigenvectors can be computed from the Hessenberg matrix by inverse iteration, see [20, Sec. 7.6.1] or [29, Sec. 2.2]. Assuming that good approximations $\hat{\lambda}$ to the true eigenvalues are available, inverse iteration approximates an eigenvector y by solving

$$(H - \hat{\lambda}I)y^{(k)} = s^{(k)}y^{(k-1)}, \quad k \geq 1. \tag{3.7}$$

The scalar $s^{(k)}$ normalizes the iterate $y^{(k)}$ and $y^{(0)}$ is a given starting vector. Inverse iteration commonly computes only a single iteration of (3.7) because of two reasons. First, the starting vector can be chosen such that a single iteration most frequently leads to convergence [45, 40]. Second, the residual can grow by doing additional iterations [29, Sec. 4.1]. Hence, if $y^{(1)}$ meets the convergence criterion, it is accepted as an eigenvector. Otherwise, a new starting vector orthogonal to previous choices is tried. This strategy is, for example, realized in the LAPACK 3.9.0 inverse iteration routine `DLAEIN`.

Standard approaches to the solution of $(H - \hat{\lambda} I)y^{(1)} = s^{(1)}y^{(0)}$ are an LU or an RQ decomposition. The LU decomposition employed by `DLAEIN` factors $H - \hat{\lambda} I$ into a lower unit triangular matrix L and an upper triangular matrix U with partial pivoting through a permutation matrix P and computes

$$P(H - \hat{\lambda} I) = LU, \quad Lz = Py^{(0)}, \quad Uy^{(1)} = \gamma z, \quad y \leftarrow s^{(1)}y^{(1)}.$$

The scalar $\gamma \in (0, 1]$ serves the avoidance of overflow. `DLAEIN` chooses the starting vector implicitly as $y^{(0)} = P^{-1}Le$, where e is the vector with all ones. Thereby the computation reduces to the solution of $Uy^{(1)} = \gamma z$ through backward substitution. The RQ approach introduced by Henry [26, 25] factors $H - \hat{\lambda} I$ into an upper triangular R and an orthogonal Q and computes

$$H - \hat{\lambda} I = RQ, \quad Rz = \gamma y^{(0)}, \quad y^{(1)} \leftarrow Q^T z, \quad y \leftarrow s^{(1)}y^{(1)}.$$

As before, the scalar $\gamma \in (0, 1]$ serves the avoidance of overflow. Henry transforms $H - \lambda I$ into a triangular R by eliminating the subdiagonal entries of $H - \lambda I$ from right to left through a series of Givens rotations. Each Givens rotation annihilates a subdiagonal entry of $H - \lambda I$. The accumulation of the Givens rotations yields Q. As soon as a column of R is available, it is immediately used in a column-oriented backward substitution and then discarded. If H is not overwritten with R, a single column suffices as workspace. Furthermore, due to the intertwined execution of the transformation to triangular shape and the backward substitution, the matrix H is traversed only once.

3.2 Robust Solution of the Sylvester Equation

A linear matrix equation occurring in, for example, control theory, image processing or model reduction is the general Sylvester equation

$$AX - XB = C. \tag{3.8}$$

Here, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{m \times m}$, $C \in \mathbb{R}^{n \times m}$ are given and $X \in \mathbb{R}^{n \times m}$ is the solution that is sought.

The Sylvester equation and the eigenvalue problem are closely related. A unique solution to (3.8) exists if and only if A and B do not have any eigenvalues in common. Moreover, the eigenvalue equation $AX - X\Lambda = 0$ is a special case of the Sylvester equation. Small Sylvester equations are solved in eigenvalue problems. This includes, for example, eigenvalue reordering in the real Schur form [35] or the computation of eigenvectors from the Schur form. The latter is addressed in Paper II and III. Furthermore, the sensitivity of the solution X in (3.8) is related to the separation of A and B [46],

$$\operatorname{sep}(A,B) = \min_{X\neq 0} \frac{\|AX - XB\|_F}{\|X\|_F}.$$

The separation of two matrices is relevant to the measurement of invariant subspace sensitivity [20, Sec. 7.2.4], [27, p. 313], of which the eigenvector sensitivity is a special case [20, Sec. 7.2.5].

Two standard approaches to solving (3.8) are the Hessenberg-Schur method [19] and the Bartels–Stewart method [4]. Both methods are similar in the sense that they approach the solution by transforming (3.8) into a form that is easier to solve.

Bartels–Stewart method. The Bartels–Stewart method [4] reduces both coefficient matrices to real Schur form, $\tilde{A} = U^T AU$ and $\tilde{B} = V^T BV$. Here, the matrices $\tilde{A}$ and $\tilde{B}$ are quasi-triangular and U and V are orthogonal. The application of the computed orthogonal transformations transforms the general Sylvester equation (3.8) into a triangular Sylvester equation

$$\underbrace{U^T AU}_{\tilde{A}}\underbrace{U^T XV}_{Y} - \underbrace{U^T XV}_{Y}\underbrace{V^T BV}_{\tilde{B}} = \underbrace{U^T CV}_{\tilde{C}}. \tag{3.9}$$

The triangular Sylvester equation can be solved by a variation of backward substitution that can handle four cases arising from 1-by-1 or 2-by-2 blocks on the diagonal of $\tilde{A}$ and $\tilde{B}$. Then the solution to (3.8) is given by $X = UYV^T$.

Hessenberg–Schur method. The Hessenberg–Schur method [19] is structurally similar to the Bartels–Stewart method, but reduces only one coefficient matrix to real Schur form. The Hessenberg–Schur method therefore solves (3.8) by reducing A only to Hessenberg form, $H = U^T AU$, and B to real Schur form, $S = V^T BV$. The matrices U and V are again orthogonal. When the orthogonal transformations are applied to (3.8), the system is transformed into

$$\underbrace{U^T AU}_{H}\underbrace{U^T XV}_{Y} - \underbrace{U^T XV}_{Y}\underbrace{V^T BV}_{S} = \underbrace{U^T CV}_{\tilde{C}},$$

which can be solved with a column-oriented variant of backward substitution. The solution to (3.8) can be recovered by $X = UYV^T$.

Robust Solution of the Triangular Sylvester Equation. The triangular Sylvester equation (3.9) occurs as a computational step in the Bartels–Stewart method. It is a well-studied problem and many library solvers exist. These include the LAPACK solver `DTRSYL`, the Fortran library `RECSY` [21, 30, 31, 32], which approaches the solution with recursion, and its distributed memory extension `scasy` [21, 22, 23, 24], and the tiled task-parallel solver `FLASH_Sylv` [41] part of `libflame`.

The triangular Sylvester equation (3.9) can be solved with a variation of backward substitution. Since backward substitution is prone to overflow, robust solvers for the triangular Sylvester equation introduce a scaling factor $\gamma \in (0, 1]$ and solve the scaled triangular Sylvester equation $\tilde{A}Y - Y\tilde{B} = \gamma\tilde{C}$. Since the scaling factor is associated with the entire solution matrix, frequent scaling events can incur significant overhead if the solution matrix is kept consistently scaled throughout the computation.

Chapter 4 summarizes the contributions of the papers included in this thesis to each topic discussed in this chapter. The papers are grouped by topic and concern robustness (Paper I), the eigenvector computation (Paper II – IV) and the robust solution of the triangular Sylvester equation (Paper V).

Chapter 4

Summary of Contributions

4.1 Paper I

This summary is a condensed version of the summary found in the author's licentiate thesis [42, Sec. 2.2].

Paper I concerns the task-parallel robust solution of triangular systems with many right-hand sides $TX = B\Gamma$. By virtue of the diagonal matrix of scaling factors Γ, the scaled triangular system can be solved for $X\Gamma^{-1}$ such that the floating-point overflow threshold is not exceeded. The robust tiled backward substitution algorithm follows a standard pattern of tiled backward substitution: When a tile of the solution has been solved through a small backward substitution, this solution is used in a linear tile update. Paper I utilizes overflow protection logic that leads to a robust small backward substitution and a robust linear tile update. The robust linear tile update is designed such that it can benefit from an efficient implementation of `DGEMM`. The resulting task-parallel tiled algorithm attains a reasonable fraction of the peak performance in all numerical experiments.

The author's contributions: Collective experiment design, conducting the experiments and plotting the results, describing the test environment, reading and commenting on the manuscript.

4.2 Paper II

This summary is a condensed version of the summary found in the author's licentiate thesis [42, Sec. 2.1].

Paper II concerns the simultaneous computation of several eigenvectors from the real Schur form $T = Q^T AQ$ of a non-symmetric matrix A. The eigenvectors can be computed through backward substitution on T, followed by a backtransform with Q. The LAPACK 3.9.0 routine `DTREVC3` [17, 16] uses level-2 BLAS in the backward substitution and level-3 BLAS in the backtransform if all eigenvectors are sought. Paper II introduces level-3 BLAS to the backward substitution by expressing the computation as tiled algorithms. Thereby most of the computation corresponds to matrix–matrix multiplications. Paper II evaluates the performance and parallel scalability attained by the new set of algorithms. Since the presented algorithms are not robust, the analysis can be viewed as a scalability study under ideal conditions.

The author's contributions: Joint development and improvement of the presented algorithms, programming, joint analysis of bottlenecks, writing of the first draft, revising the draft jointly, designing and conducting the experiments, joint analysis and presentation of the numerical results.

4.3 Paper III

Paper III renders the eigenvector computation from the real Schur form robust. For this purpose, the overflow protection scheme from Paper I is applied to the algorithms of Paper II. The resulting algorithms can solve the same class of problems as the LAPACK 3.9.0 routine `DTREVC3`. In contrast to Elemental's solver `TriangEig` [37], also quasi-triangular matrices are supported. Paper III investigates the performance and parallel scalability of the robust algorithms. In the numerical experiments, the robust eigenvector routines are 15-20% slower than their non-robust counterparts, but have a similar parallel scalability.

The author's contributions: Improving the algorithms, idea presented in Sec. 5.3, programming, authoring the first draft, conducting and analyzing the experiments, joint revision.

4.4 Paper IV

Paper IV revises the RQ approach for computing eigenvectors from the Hessenberg matrix by inverse iteration. The original RQ approach [26, 25] factors $(H - \lambda_\ell I) = R_\ell Q_\ell$, where H is upper Hessenberg, R_ℓ triangular and Q_ℓ orthogonal. Since distinct shifts λ_ℓ have distinct RQ decompositions, the approach of computing the RQ decomposition and then (possibly simultaneously) solving the triangular system on R_ℓ through backward substitution has limited level-3 BLAS potential. Paper IV rearranges the computation so that in spite of distinct shifts the backward substitution phase mostly comprises matrix–matrix multiplications. The speedup is demonstrated in experiments.

4.5 Paper V

This summary is a condensed version of the summary found in the author's licentiate thesis [42, Sec. 2.3].

Paper V concerns the solution of the scaled triangular Sylvester equation $AX - XB = \gamma C$. Here A and B are upper quasi-triangular, C is a general matrix and the scalar $\gamma \in (0, 1]$ serves the avoidance of overflow. To the best of the authors' knowledge, the algorithm devised in Paper V is the first robust, tiled, level-3 BLAS based, task-parallel solver for the triangular Sylvester equation. By adding robustness, the computational gap between existing non-robust tiled task-parallel solvers such as `Flash_Sylv` [41] part of libflame 5.1.0-58 [44] and the class of problems solvable by, for example, LAPACK's `DTRSYL` is closed. The new algorithm attains a similar performance as `Flash_Sylv` if overflow protection is not necessary to solve the problem. If overflow protection is necessary and numerical scaling is triggered, the overhead is roughly a constant fraction of the performance and does not impede the parallel scalability.

The author's contributions: Derivation and programming of the algorithm, writing the draft, joint design of the experiments, conducting the experiments, interpretation and presentation of the numerical results, joint revision.

CHAPTER 5

Bibliography

[1] E. Anderson. Robust Triangular Solves for Use in Condition Estimation. LAPACK Working Note 36, Cray Research Inc., August 1991.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and A. McKenney. *LAPACK Users' Guide.* SIAM, 3rd edition, 1999.

[3] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A View of the Parallel Computing Landscape. *Communications of the ACM*, 52(10):56–67, Oct. 2009.

[4] R. H. Bartels and G. W. Stewart. Solution of the Matrix Equation $AX + XB = C$. *Communications of the ACM*, 15(9):820–826, 1972.

[5] P. Benner, V. Mehrmann, V. Sima, S. Van Huffel, and A. Varga. SLICOT – a subroutine library in systems and control theory. In *Applied and computational control, signals, and circuits*, pages 499–539. Springer, 1999.

[6] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[7] K. Braman, R. Byers, and R. Mathias. The Multishift QR Algorithm. Part I: Maintaining Well-Focused Shifts and Level 3 Performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–047, 2002. Revised edition of the original article from 1991.

[8] K. Braman, R. Byers, and R. Mathias. The Multishift QR Algorithm. Part II: Aggressive Early Deflation. *SIAM Journal on Matrix Analysis and Applications*, 23(4):948–973, 2002.

[9] A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov. The Impact of Multicore on Math Software. In *International Workshop on Applied Parallel Computing*, pages 1–10. Springer, 2006.

[10] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35(1):38–53, 2009.

[11] J. Demmel. Open Problems in Numerical Linear Algebra. LAPACK Working Note 47, April 1992.

[12] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki. Accelerating Numerical Dense Linear Algebra Calculations with GPUs. *Numerical Computations with GPUs*, pages 1–26, 2014.

[13] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[14] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[15] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software. *SIAM Review*, 46(1):3–45, 2004.

[16] M. R. Fahey. New Complex Parallel Eigenvalue and Eigenvector Routines. LAPACK Working Note 153, Computational Migration Group, Computer Science Corporation, August 2001.

[17] M. Gates, A. Haidar, and J. Dongarra. Accelerating computation of eigenvectors in the dense nonsymmetric eigenvalue problem. In *International Conference on High Performance Computing for Computational Science*, pages 182–191. Springer, 2014.

[18] P. L. Gatti. *Applied Structural and Mechanical Vibrations: Theory and Methods*. CRC Press, 2014.

[19] G. Golub, S. Nash, and C. Van Loan. A Hessenberg-Schur method for the problem $AX + XB = C$. *IEEE Transactions on Automatic Control*, 24(6):909–913, 1979.

[20] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 4rd edition, 2013.

[21] R. Granat, I. Jonsson, and B. Kågström. RECSY and SCASY library software: Recursive blocked and parallel algorithms for Sylvester-type matrix equations with some applications. In *Parallel Scientific Computing and Optimization*, pages 3–24. Springer, 2009.

[22] R. Granat and B. Kågström. Parallel Solvers for Sylvester-Type Matrix Equations with Applications in Condition Estimation, Part I: Theory and Algorithms. *ACM Transactions on Mathematical Software*, 37(3), 2010. Article No. 32.

[23] R. Granat and B. Kågström. Parallel Solvers for Sylvester-type Matrix Equations with Applications in Condition Estimation, Part II: The SCASY Software Library. *ACM Transactions on Mathematical Software*, 37(3), 2010. Article No. 33.

[24] R. Granat, B. Kågström, and P. Poromaa. Parallel ScaLAPACK-style algorithms for solving continuous-time Sylvester matrix equations. In *European Conference on Parallel Processing*, pages 800–809. Springer, 2003.

[25] G. Henry. The Shifted Hessenberg System Solve Computation, 1994. Cornell University, NY, USA.

[26] G. Henry. A Parallel Unsymmetric Inverse Iteration Solver. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 546–551, 1995.

[27] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[28] Intel oneAPI Math Kernel Library. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html [March 2021].

[29] I. C. Ipsen. Computing an Eigenvector with Inverse Iteration. *SIAM Review*, 39(2):254–291, 1997.

[30] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems – Part I: One-sided and coupled Sylvester-type matrix equations. *ACM Transactions on Mathematical Software*, 28(4):392–415, 2002.

[31] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems – Part II: Two-sided and Generalized Sylvester and Lyapunov Matrix Equations. *ACM Transactions on Mathematical Software*, 28(4):416–435, 2002.

[32] I. Jonsson and B. Kågström. RECSY – A High Performance Library for Sylvester-Type Matrix Equations. In *European Conference on Parallel Processing*. Springer, 2003.

[33] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software*, 24(3):268–302, 1998.

[34] C. C. Kjelgaard Mikkelsen and L. Karlsson. Blocked Algorithms for Robust Solution of Triangular Linear Systems. In *International Conference on Parallel Processing and Applied Mathematics*, pages 68–78. Springer, 2017.

[35] D. Kressner. Block algorithms for reordering standard and generalized Schur forms. *ACM Transactions on Mathematical Software*, 32(4):521–532, 2006.

[36] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[37] T. Moon and J. Poulson. Accelerating eigenvector and pseudospectra computation using blocked multi-shift triangular solves, July 2016. arXiv:1607.01477.

[38] M. Myllykoski and C. C. Kjelgaard Mikkelsen. Task-based, GPU-accelerated and robust library for solving dense nonsymmetric eigenvalue problems. *Concurrency and Computation: Practice and Experience*, page e5915, 2020.

[39] OpenBLAS. http://www.openblas.net/ [March, 2021].

[40] G. Peters and J. H. Wilkinson. The Calculation of Specified Eigenvectors by Inverse Iteration. In *Handbook for Automatic Computation*, pages 418–439. Springer, 1971.

[41] E. S. Quintana-Ortí and R. A. van de Geijn. Formal Derivation of Algorithms: The Triangular Sylvester Equation. *ACM Transactions on Mathematical Software*, 29(2):218–243, 2003.

[42] A. B. Schwarz. Towards Efficient Overflow-free Solvers for Systems of Triangular Type. Licentiate thesis, 2019.

[43] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra. Dense Linear Algebra Solvers for Multicore with GPU Accelerators. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1–8, Atlanta, GA, April 19-23 2010. IEEE Computer Society.

[44] F. Van Zee, E. Chan, R. A. van de Geijn, E. S. Quintana-Ortí, and G. Quintana-Ortí. Introducing: The Libflame Library for Dense Matrix Computations. *Computing in Science & Engineering*, 2009.

[45] J. Varah. The Calculation of the Eigenvectors of a General Complex Matrix by Inverse Iteration. *Mathematics of Computation*, 22(104):785–791, 1968.

[46] J. Varah. On the Separation of Two Matrices. *SIAM Journal on Numerical Analysis*, 16(2):216–222, 1979.

[47] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1-2):3–35, 2001.