Dense and Structured Matrix Computations—the Parallel QR Algorithm and Matrix Exponentials

THÈSE N^{O} 6067 (2014)

PRÉSENTÉE LE 17 JANVIER 2014

À LA FACULTÉ DES SCIENCES DE BASE

ALGORITHMES NUMÉRIQUES ET CALCUL HAUTE PERFORMANCE - CHAIRE CADMOS

PROGRAMME DOCTORAL EN MATHÉMATIQUES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Meiyue SHAO

acceptée sur proposition du jury:

Prof. J. Krieger, président du jury

Prof. D. Kressner, Prof. B. Kågström, directeurs de thèse

Prof. Z. Bai, rapporteur

Prof. F. M. Dopico, rapporteur

Prof. J. Maddocks, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse

2014

Acknowledgements

My deepest appreciation goes to my advisors Prof. Daniel Kressner and Prof. Bo Kågström, who have provided me all kinds of supports during my doctoral study at Umeå and Lausanne. It has been really great to work with them. They both spent a lot of time discussing my research work with me, providing many constructive comments and suggestions, patiently answer all questions raised by me, and kindly encouraging me for any progress I had made. I have greatly benefited from both of them through scientific discussions and daily communicaiton.

I am indebted to Prof. Lars Karlsson, for all kinds of assistance, both scientifically and non-scientifically. I am also grateful to all other members in the Numerical Linear Algebra and Parallel High Performance Computing Groups at Umeå University, and in MATHICSE ANCHP at EPFL, especially to Björn Adlerborn, Cedric Effenberger, Carl Christian Kjelgaard Mikkelsen, and Christine Tobler, for a lot of helpful discussions.

Many thanks to High Performance Computing Center North (HPC2N) at Umeå and HPC-DIT at EPFL for providing computational resources and technical supports.

I would like to express my gratitude to all professors in the Computational Mathematics Group at Fudan University, and to Dr. Xiaoye Li at Lawrence Berkeley National Laboratory. They introduced me to the area of numerical linear algebra and high performance computing, and enlightened me the first glance of research.

I also would like to thank all friends (among whom I mention several ones : Le Chen, Kai Du, Wubin Li, Da Wang, Lei Xu, Wenqi You, etc.) in Sweden and Switzerland, for making my life more enjoyable.

Last but not the least, a special thank goes to my parents and my girlfriend Liming Zhang. Without their unconditional support, this work would not have been accomplished.

Lausanne, December 2013 Meiyue Shao

Abstract

This thesis is concerned with two classical topics in matrix computations : The QR algorithm for solving nonsymmetric eigenvalue problems and the computation of matrix exponentials for two types of structured matrices. We focus on the performance in the former topic and on accuracy in the latter one.

For computing all eigenvalues of a non-Hermitian matrix, the QR algorithm which iteratively computes a Schur decomposition of the matrix is the method of choice. We present a new parallel implementation of the multishift QR algorithm targeting distributed memory architectures. Starting from recent developments of the parallel multishift QR algorithm, we propose a number of algorithmic and implementation improvements. Guidelines concerning several important tunable algorithmic parameters are also provided. Numerous computational experiments confirm that our new implementation significantly outperforms previous parallel implementations of the QR algorithm.

The computation of the exponential of a square matrix is also an important task in matrix computations. For a general dense matrix, the scaling and squaring method coupled with Padé approximation is the most popular approach. However, for an essentially nonnegative matrix (a real square matrix with nonnegative off-diagonal entries), truncated Taylor series rather than Padé approximation is preferred to achieve componentwise accuracy in the matrix exponential. We propose a method which efficiently computes all entries of the exponential of an essentially nonnegative matrix to high relative accuracy. Truncation and rounding error bounds, as well as numerical experiments demonstrate the efficiency and accuracy of our method.

When the matrix is banded, the entries of its matrix exponential decay exponentially away from the main diagonal. We analyze the decay property for the exponentials of several classes of doubly-infinite skew-Hermitian matrices. Then finite section methods based on the decay property are established. We also propose a repeated doubling strategy which works well even when a priori error estimates are pessimistic or not easy to compute. Finally, numerical experiments are presented to illustrate the effectiveness of the finite section method.

Keywords : Nonsymmetric eigenvalue problem, matrix exponential, multishift QR algorithm, aggressive early deflation, parallel algorithm, distributed memory architecture, aggressively truncated Taylor series method, finite section method, exponential decay

Zusammenfassung

Das Thema dieser Doktorarbeit umfasst zwei klassische Gebiete der numerischen linearen Algebra: Der parallele unsymmetrische QR-Algorithmus zur Eigenwertbestimmung sowie die Berechnung von Matrixexponentialen für zwei Typen von strukturierten Matrizen. Während im ersten Thema die effiziente, hoch-performante Berechnung im Vordergrund steht, liegt der Schwerpunkt des zweiten Themas auf der Genauigkeit der erhaltenen Lösungen.

Die Berechnung der Eigenwerte einer nicht-hermiteschen Matrix ist ein klassisches Thema der Matrizenrechnung. Falls alle Eigenwerte benötigt werden, ist der QR-Algorithmus—der iterativ eine Schur-Zerlegung der Matrix bestimmt—das Mittel der Wahl. In der vorliegenden Doktorarbeit stellen wir eine neue, parallele Implementierung des Multishift-QR-Algorithmus für verteilte Speicherarchitekturen vor. Ausgehend von aktuellen Entwicklungen auf dem Gebiet des parallelen Multishift-QR-Algorithmus werden dabei verschiedene algorithmische und implementationstechnische Verbesserungen entwickelt. Hierbei erläutern wir die dabei auftretenden konfigurierbaren Parameter und geben Hilfestellungen für ihre problemspezifische Wahl. Eine Vielzahl an numerischen Experimenten bestätigt dabei die signifikante Verbesserung gegenüber bereits exisitierenden parallelen Implementationen des QR-Algorithmus.

Die Berechnung des Matrixexponentials einer quadratischen Matrix ist eine weitere wichtige Problemstellung im Rahmen der Matrizenrechnung. Für allgemeine, dichtbesetzte Matrizen ist dabei die sogenannte *scaling-and-squaring*-Methode in Verbindung mit Padé-Approximation das meistgenutzte Verfahren. Bei Metzler-Matrizen, d.h. reellen Matrizen mit nicht-negativen Nebendiagonaleinträgen, ist jedoch die abgeschnittene Taylorapproximation zu bevorzugen. Die Verwendung der Taylorreihe anstatt der Padé-Approximation erlaubt hierbei eine Verbesserung der elementweisen Genauigkeit des Matrixexponentials. Das in dieser Doktorarbeit vorgestellte Verfahren berechnet alle Einträge des Exponentials einer Metzler-Matrix mit hoher relativer Genauigkeit. Die Schranken für Abschneide- sowie Rundungsfehler zeigen zusammen mit numerischen Experimenten die Effizienz und Genauigkeit unseres Ansatzes.

Im Fall von Bandmatrizen nehmen die Einträge des Matrixexponentials ausserhalb der Hauptdiagonalen exponentiell schnell ab. Wir analysieren die Gesetzmäßigkeiten dieser Eigenschaft für Exponentiale verschiedener Klassen zweifach-unendlicher, schiefhermitescher Matrizen und stellen finite-section-Methoden vor, die dieses Abklingverhalten ausnutzen. Weiterhin betrachten wir eine neue Strategie basierend auf wiederholter Verdopplung, die auch für pessimistische oder schwierig zu berechnende a-priori Fehlerschätzer gut funk-

tioniert. Zu guter Letzt illustrieren wir die Effektivität der *Finite-section*-Methode anhand numerischer Beispiele.

Stichwörter: Unsymmetrische Eigenwertprobleme Matrixexponential, Multishift-QR-Algorithmus, aggressive early deflation, Parallele Algorithmen, verteilte Speicherarchitektur, Aggressively-truncated-Taylor-series-Methode, Finite-section-Methode, Exponentieller Abfall

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Notation

$\emptyset$	empty set
$\mathbb{N}$	set of natural numbers, i.e., $\{0,1,2,\ldots\}$
$\mathbb{Z}$	set of integers, i.e., $\{0,\pm1,\pm2,\ldots\}$
$\mathbb{R}$	set of real numbers
$\mathbb{R}_+$	set of nonnegative real numbers, i.e., $\{x \in \mathbb{R}\colon\ x \geq 0\}$
$\mathbb{C}$	set of complex numbers
e	Napier's constant, i.e., $\mathrm{e} = \lim_{n\to\infty}(1+1/n)^n$
i	imaginary unit
$\lfloor x \rfloor$	floor function, $\lfloor x \rfloor = \max\{n \in \mathbb{Z}\colon\ n \leq x\}$
$\lceil x \rceil$	ceiling function, $\lceil x \rceil = \min\{n \in \mathbb{Z}\colon\ n \geq x\}$
$i:j$	$\{i, i+1, \ldots, j\}$
$\binom{n}{m}$	binomial coefficients, $\binom{n}{m} = n!/[m!(n-m)!]$
z^*	complex conjugate of z
$\Re(z)$	real part of z
$\Im(z)$	imaginary part of z
$\lvert z \rvert$	modulus of z
$A = [a_{ij}]$	matrix with entries a_{ij}
a_{ij}	(i,j)-th entry of the matrix A
A_{ij}	(i,j)-th entry (or (i,j)-th block, depending on the context) of the matrix A
$A_{(i,:)}$	ith column of A
$A_{(:,j)}$	jth column of A
$A_{(i_1:i_2,j_1:j_2)}$	submatrix of A with row indices $i_1:i_2$ and column indices $j_1:j_2$
struct(A)	nonzero pattern of a matrix A, i.e., $\{(i,j)\colon\ a_{ij} \neq 0\}$
I	identity matrix/operator
e_j	jth column of the identity matrix
$l^2(\mathbb{Z})$	space of doubly-square-summable sequences, i.e., $\{x = [x_i]\colon\ x_i \in \mathbb{C}, \sum_{i\in\mathbb{Z}} \lvert x_i \rvert^2 < +\infty\}$
$\overline{\phi}$	closure of a linear operator ϕ
$\lVert\cdot\rVert_p$	L^p-norm (Minkowski norm)
$\lVert\cdot\rVert_F$	Frobenius norm
A^T	transpose of A

A^*	conjugate transpose of A
$A \otimes B$	Kronecker product of A and B
$A \geq B$, $B \leq A$	$a_{ij} \geq b_{ij}$ for all i, j
$\lvert A \rvert$	matrix of absolute values of A
$\det(A)$	determinant of A
$\Lambda(A)$	spectral of A, i.e., $\{\lambda \in \mathbb{C}\colon\ \lambda I - A$ is not invertible$\}$
$\rho(A)$	spectral radius of A, i.e., $\sup\{\lvert\lambda\rvert : \lambda \in \Lambda(A)\}$
$\mathbb{K}_m[x]$	set of polynomials in x up to degree m over $\mathbb{K}$
$\deg p(x)$	degree of the polynomial $p(x)$
$\partial\Omega$	boundary of the domain Ω
$g(n) = \mathcal{O}\big(f(n)\big)$	$\exists C > 0$ such that $\big\lvert g(n)\big\rvert \leq C\big\lvert f(n)\big\rvert$
$g(n) = \Theta\big(f(n)\big)$	$g(n) = \mathcal{O}\big(f(n)\big)$ and $f(n) = \mathcal{O}\big(g(n)\big)$
$g(n) = \omega\big(f(n)\big)$	$\liminf_{n\to\infty} \big\lvert g(n)/f(n)\big\rvert = +\infty$
$\mathrm{fl}(x \circ y)$	floating-point operation
$\mathrm{fl}(A)$	computational result of A in floating-point arithmetic (computed by a certain algorithm)
$\underline{\mathrm{fl}}(A)$	computational result of A satisfying $\underline{\mathrm{fl}}(A) \leq A$
$\overline{\mathrm{fl}}(A)$	computational result of A satisfying $\overline{\mathrm{fl}}(A) \geq A$
$\mathbf{u}$	unit roundoff
$R_{\max}$	largest finite floating-point number
$R_{\min}$	smallest positive normalized floating-point number
$\mathrm{Diag}\{A_1, A_2, \ldots, A_n\}$	block diagonal matrix with A_i's as its diagonal blocks
$\mathrm{Tridiag}\left\{\begin{matrix} & c_1 & \cdots & c_{n-1} & \\ b_1 & \cdots & \cdots & & b_n \\ & a_2 & \cdots & a_n & \end{matrix}\right\}$	tridiagonal matrix whose diagonal, subdiagonal, and superdiagonal entries are $\{b_1, \ldots, b_n\}$, $\{a_2, \ldots, a_n\}$, and $\{c_1, \ldots, c_{n-1}\}$, respectively

1 Introduction

This thesis contains contributions in two important areas in numerical linear algebra—nonsymmetric eigenvalue problems and matrix exponentials. In the first part of the thesis (Chapters 2 and 3), we present a parallel implementation of the QR algorithm which solves dense nonsymmetric eigenvalue problems; in the second part (Chapters 4–6), we discuss the computation of matrix exponentials for two types of structured matrices. This chapter provides a short introduction to each problem, while detailed discussions will be given in the subsequent chapters.

The QR algorithm. Computing the eigenvalues of a given matrix is one of the most important tasks in matrix computations. The most general form of this problem is to compute all eigenvalues of $A \in \mathbb{C}^{N\times N}$ where no special structure of A (e.g., symmetry or sparsity) is exploited. Such a task arises in various numerical algorithms. For example, the Bartels-Stewart algorithm [18] and the Schur-Parlett algorithm [116] both require a preprocessing stage in which all eigenvalues of a matrix need to be computed. Even if only a subset of eigenvalues are of interest, many projection-based algorithms end up with solving a (smaller) dense eigenvalue problem.

The most popular approach for solving the dense nonsymmetric eigenvalue problem is the QR algorithm, which was proposed independently by Francis [49, 50] and Kublanovskaya [99], based on Rutishauser's LR algorithm [126, 127]. In the past 50 years the QR algorithm has been the method of choice because of its robustness and effectiveness. Concerning the implementation of the QR algorithm, the ALGOL procedures `QR` by Ruhe [125] and `hqr` by Martin, Petersen, and Wilkinson [105] were among the first publicly available computer implementations of the QR algorithm. A Fortran translation of `hqr` was included in EISPACK [137] as routine `HQR`. The initial version of the LAPACK [6] routine `DHSEQR` was based on work by Bai and Demmel [10]; the most notable difference to `HQR` was the use of multishift techniques to improve data locality. This routine had only seen a few minor modifications [3] until LAPACK version 3.1, when it was replaced by an implementation incorporating pipelined bulges and aggressive early deflation techniques from the works by Braman, Byers, and Mathias [30, 31]. This implementation is

described in more detail in [32].

There has been a lot of early work on parallelizing the QR algorithm, for example in [69, 129, 147, 148, 149, 156]. Based on some theoretical analyses on the convergence [148, 156] and scalability [69], the first publicly available parallel implementation was developed and released 1997 in ScaLAPACK [26] version 1.5 as routine PDLAHQR, see [70] for details. A complex version PZLAHQR of this routine was included later [47]. However, it might be interesting to note that all recently released high-performance linear algebra packages, such as MAGMA and PLASMA [1], ELPA [8], FLAME [24] lack adapted implementations of the QR algorithm or other nonsymmetric eigenvalue solvers. Recently a novel parallel QR algorithm incorporating several modern techniques has been developed by Granat, Kågström, and Kressner [61]. In the first part of this thesis, we describe a new parallel implementation of the QR algorithm that aims to replace ScaLAPACK's PDLAHQR. This implementation is largely based on the early work in [61]. We propose a number of algorithmic and implementation improvements including multilevel aggressive early deflation, data redistribution technique, as well as strategies of choosing parameters. With the help of these improvements, our new implememtation of the parallel QR algorithm outperforms the previous version in [61] and ScaLAPACK's PDLAHQR.

Matrix exponentials. The computation of matrix functions is another fundamental topic in matrix computations. The most important and well-studied transcendental matrix function is the matrix exponential since it naturally appears in the solution of linear dynamical systems and has wide applications in physics, biology, finance, and engineering. We refer to [73, Chapter 2] and references therein for applications of the matrix exponential.

The matrix exponential can be computed in many ways, based on various properties of the exponential function. The classical paper [108] published in 1978 and reprinted with updates [109] in 2003 summarizes and analyzes many methods for computing the matrix exponential. Another survey on the theory and computation of the matrix exponential is provided in the monograph [73] (see, especially, Chapter 10 in [73]). Among many potential candidates, the scaling and squaring method coupled with Padé approximation is the most popular approach for a dense nonsymmetric matrix. The function `expm` in MATLAB is an excellent general purpose solver which implements such a method incorporating several advance techniques [72, 74].

Unfortunately, none of the existing methods is completely satisfactory for computing the matrix exponential [109]. Even the excellent solver `expm` makes no exception. Therefore, when a certain structure of the matrix can be exploited, it is desirable to design a specialized method tailored to this structure so that higher performance or accuracy is achieved compared to a general purpose solver. In the second part of this thesis, we will study the computation of matrix exponentials for two types of structured matrices—essentially nonnegative matrices and doubly-infinite skew-Hermitian matrices. Accuracy is the main consideration while efficiency is also taken into account.

The exponential of an essentially nonnegative matrix often arise in Markov chains and requires accurate computation for each small entry even if the solution is badly-scaled. Since MATLAB's `expm` ensures no more than normwise backward stability, an alternative approach is needed to obtain componentwise accuracy. In Chapter 5, we will develop $\mathcal{O}(N^3 \log N)$ algorithms which efficiently compute the exponential of an essentially nonnegative matrix to high componentwise relative accuracy.

For doubly-infinite skew-Hermitian matrices, certainly no solver designed for finite matrices can be directly applied. In some applications only a finite diagonal block of the solution is required. This request is accomplished by the so called *finite section method* which only involves computations on finite matrices. Detail analyses for the finite section method will be presented in Chapter 6.

Organization of the thesis. This thesis is largely based on the papers [62, 88, 133, 134]. It is organized as follows. Chapter 2 briefly recalls the sequential QR algorithm. Then we discuss the parallel multishift QR algorithm on distributed memory architectures as well as its implementation in Chapter 3. The user's guide of our library software is provided in Appendix C. Chapter 4 contains a brief review on the theory and computation of matrix exponentials. Then in Chapter 5, we present efficient algorithms which compute exponentials of essentially nonnegative matrices to high componentwise relative accuracy. In Chapter 6, we discuss finite section methods for exponentials of doubly-infinite skew-Hermitian matrices.

Parts of this thesis are based on material discussed in

> M. Shao. *Parallel variants and library software for the QR algorithm and the computation of the matrix exponential of essentially nonnegative matrices*. Licentiate Thesis, Department of Computing Science, Umeå University, Sweden. April 2012.

Apart from a brief introduction, the Licentiate thesis consists of a conference paper and two technical reports:

Paper I. B. Kågström, D. Kressner, and M. Shao. On aggressive early deflation in parallel variants of the QR algorithm. *Applied Parallel and Scientific Computing (PARA 2010)*, Lecture Notes in Computer Science, Springer, LNCS 7133, pages 1–10, 2012.

Paper II. R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. Technical Report, UMINF-12.06, April 2012.

Paper III. M. Shao, W. Gao, and J. Xue. Componentwise high relative accuracy algorithms for the exponential of an essentially nonnegative matrix. Technical Report, UMINF-12.04, March 2012.

In the following, we explain the relations between Papers I–III and the results presented in this thesis in more detail.

Chapter 3 represents a significant extension of preliminary results on the parallel QR algorithm presented in Papers I and II. While Papers I and II focus on software aspects, Chapter 3 presents a detailed derivation of the involved parallel bulge-chasing algorithm. Moreover, the performance model from Paper II has been adjusted and is supported by experimental data verifying the model. As documented in Section 3.7 and Appendix C, several additional efforts have gone into the public release of the software. This includes the tuning of parameters (Section C.4) and adjusting the software to novel architectures (*Abisko* and *Bellatrix*). As part of this adjustment, new algorithmic developments to avoid redundant computations (Section 3.3.4) have been made.

Chapter 5 and Paper III are both concerned with computing exponentials of essentially nonnegative matrices to high relative accuracy. Paper III represents a very preliminary version of the results obtained in Chapter 5. In fact, the analysis and all algorithms in Chapter 5 have seen major new developments and the derivations of the results have been redeveloped from scratch.

All the other chapters (Chapters 1, 2, 4, 6) contain new material and have no relation to the Licentiate thesis.

Part I—The QR Algorithm

2 Nonsymmetric Eigenvalue Problems and the QR Algorithm

The solution of matrix eigenvalue problems is a fundamental topic in numerical linear algebra, with applications in various areas of science and engineering. Numerous methods have been proposed for solving matrix eigenvalue problems, based on different properties of the matrix and different demands on the knowledge of the spectrum, see, e.g., [143]. We are interested in a general case—computing all eigenvalues of a square matrix. In this chapter we briefly recall the sequential QR algorithm. Detailed discussions of this topic can be found in, e.g., [95].

2.1 Nonsymmetric eigenvalue problems

For $A \in \mathbb{C}^{N\times N}$, the standard eigenvalue problem is to solve a nonlinear equation of the form

$$Ax = \lambda x, \qquad (x \neq 0), \tag{2.1.1}$$

where $\lambda \in \mathbb{C}$ and $x \in \mathbb{C}^{N\times 1}$. The scalar λ is called an *eigenvalue* of A, and the vector x is the corresponding *eigenvector*. The set of all eigenvalues of A, denoted by $\Lambda(A)$, is called the *spectrum* of A. Evidently, $\lambda \in \Lambda(A)$ if and only if

$$\det(\lambda I - A) = 0.$$

The polynomial $p(t) = \det(tI - A)$ is called the *characteristic polynomial* of A. An important property of the spectrum is that it is preserved under *similarity transformations*, i.e., for any nonsingular matrix P, we have $\Lambda(P^{-1}AP) = \Lambda(A)$. Theorem 2.1.1 summarizes several important matrix decompositions related to $\Lambda(A)$ involving similarity transformations.

Theorem 2.1.1. *Let A be an $N \times N$ complex matrix.*
*(a) (*Jordan decomposition [83]*) There exists a nonsingular matrix P such that*

$$J = P^{-1}AP = \mathrm{Diag}\left\{J_{k_1}(\lambda_1), J_{k_2}(\lambda_2), \ldots, J_{k_s}(\lambda_s)\right\},$$

where $k_1 + k_2 + \cdots + k_s = N$ *and each* $J_{k_j}(\lambda_j)$ *is of the form*

$$J_{k_j}(\lambda_j) = \begin{bmatrix} \lambda_j & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_j \end{bmatrix}_{k_j \times k_j}.$$

The block diagonal matrix J *is called the* Jordan canonical form *of* A. *(The eigenvalues* λ_1, λ_2, $\ldots$, λ_s *are not necessarily distinct.)*

*(b) (*Schur decomposition [130]*) There exists a unitary matrix* $Q \in \mathbb{C}^{N\times N}$ *such that* $T = Q^* AQ$ *is upper triangular. The upper triangular matrix* T *is called a* Schur form *of* A.

*(c) (*Real Schur decomposition [113]*) If all entries of* A *are real, then there exists a real orthogonal matrix* $Q \in \mathbb{R}^{N\times N}$ *such that* $T = Q^T AQ$ *is* quasi-upper triangular.[1] *The quasi-upper triangular matrix* T *is called a* real Schur form *of* A.

Proof. See, e.g., [52, 77]. □

The choice of method for computing the eigenvalues largely depends on the properties of A. For example, when A is Hermitian, the Jordan canonical form (as well as the Schur form) of A is a real diagonal matrix. This property leads to various approaches to symmetric eigenvalue problems, see, e.g., [117]. When A is sparse but only a few eigenvalues of A are of interest, Krylov subspace methods are preferred. We refer to [128] for discussions in this direction. In this thesis, we only study dense nonsymmetric eigenvalue problems. By dense and nonsymmetric we mean that no potential structure such as symmetry or sparsity in A is exploited.

Theoretically $\Lambda(A)$ can be read off from the diagonal entries of the Jordan canonical form of A. However, the Jordan canonical form can be extremely sensitive to small perturbations [163] and hence is not easy to compute in practice [86, 89, 90]. The Schur form plays a key role in the computation of $\Lambda(A)$ since only unitary similarities are involved. It can be computed in a backward stable manner by the QR algorithm, which will be presented in the next subsection. Once a Schur form of A has been calculated, then $\Lambda(A)$ is read off from its diagonal and the eigenvectors of A can also be computed easily, see [58, 143]. When the matrix A is real, $\Lambda(A)$ is symmetric with respect to the real axis. Thus it is desirable to avoid complex arithmetic during the computation so that the symmetry in $\Lambda(A)$ is preserved. The real Schur form offers such a possibility. Since the (complex) Schur form is conceptionally simpler than the real Schur form, we restrict our discussion to the real Schur form which automatically covers the simpler case.

1. A quasi-upper triangular matrix is a block upper triangular whose diagonal blocks are of order one or two, where any irreducible 2×2 diagonal block contains a conjugate pair of complex eigenvalues.

A notable remark is that the order of eigenvalues in the real Schur form can be arbitrarily chosen, as long as conjugate pairs of complex eigenvalues do not split. It can be easily verified that two consecutive real eigenvalues can be swapped by

$$G(\theta)^T \begin{bmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{bmatrix} G(\theta) = \begin{bmatrix} t_{22} & t_{12} \\ 0 & t_{11} \end{bmatrix},$$

where

$$G(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

is a *Givens rotation* and $\cot\theta = t_{12}/(t_{11} - t_{22})$, provided that $t_{11} \neq t_{22}$, see Appendix A for details. Swapping diagonal blocks involving complex eigenvalues is a bit more complicated. Let

$$T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix},$$

where T_{11} and T_{22} are real square matrices of order one or two and $\Lambda(T_{11}) \cap \Lambda(T_{22}) = \emptyset$. We seek for an orthogonal matrix U such that

$$U^T T U = T = \begin{bmatrix} \tilde{T}_{11} & \tilde{T}_{12} \\ & \tilde{T}_{22} \end{bmatrix}, \qquad \Lambda(\tilde{T}_{11}) = \Lambda(T_{22}), \qquad \Lambda(\tilde{T}_{22}) = \Lambda(T_{11}).$$

By solving the Sylvester equation $T_{11}X - XT_{22} = T_{12}$ and computing the QR decomposition

$$\begin{bmatrix} -X \\ I \end{bmatrix} = U \begin{bmatrix} R \\ 0 \end{bmatrix},$$

a desired orthogonal matrix U is found,[2] see [12] for details. Based on this swapping algorithm, any order of eigenvalues can be obtained by swapping consecutive diagonal blocks. For an advanced reordering strategy, we refer to [96].

Another remark is that a real Schur form can always be converted into a *standardized* form in the sense that any of its 2×2 diagonal block can be put into either the form

$$\begin{bmatrix} \alpha & \delta \\ 0 & \beta \end{bmatrix}$$

or

$$\begin{bmatrix} \alpha & \beta \\ \delta & \alpha \end{bmatrix}, \qquad (\beta\delta < 0).$$

2. In practice, we solve $T_{11}X - XT_{22} = \xi T_{12}$, where $\xi \in [0,1]$ is chosen to avoid overflow, and then compute the QR decomposition of $\begin{bmatrix} -X \\ \xi I \end{bmatrix}$.

Let $T \in \mathbb{R}^{N\times N}$ be a block upper triangular matrix with 1×1 and 2×2 diagonal blocks. For a 2×2 diagonal block of T which contains two real eigenvalues, a Schur decomposition of this block splits it into two 1×1 diagonal blocks; for a 2×2 diagonal block

$$T_k = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}$$

containing a conjugate pair of complex eigenvalues, it can be easily verified that a Givens rotation $G(\theta)$ with $\tan 2\theta = (t_{11}-t_{22})/(t_{12}+t_{21})$ transforms T_k to $G(\theta)^T T_k G(\theta)$ which is in the standardized form. Therefore, by standardizing each diagonal block, T is converted to a standardized form by orthogonal similarity. An advantage of the standardized Schur form is that all 2×2 irreducible diagonal blocks correspond to conjugate pairs of complex eigenvalues. We will see in Section 2.2.3 that this property is helpful when checking convergence of the eigenvalues.

2.2 The QR Algorithm

The QR algorithm has been extensively studied in the past decades, see, e.g., [57, 160] and the references therein. By now it is the de facto standard for solving dense nonsymmetric eigenvalue problems. The software libraries LAPACK [6] and ScaLAPACK [26] both provide implementations of modern variants of the QR algorithm as standard dense eigensolvers. We remark that there exist other approaches such as Jacobi-like algorithms [140] and divide-and-conquer algorithms [11, 15] for solving (2.1.1) when A is dense and non-Hermitian. The discussion of these methods is beyond the scope of this thesis.

In the following we briefly recall some basics about the QR algorithm, as well as a modern variant proposed by Braman et al. [30, 31] on which our parallel QR algorithm in Chapter 3 has been built.

2.2.1 Francis double-shift QR algorithm

Basic QR iteration. Setting $A^{(0)} = A$, the basic QR iteration reads

$$A^{(k)} = Q^{(k)}R^{(k)}, \qquad A^{(k+1)} = R^{(k)}Q^{(k)}, \tag{2.2.1}$$

where $Q^{(k)}$ is orthogonal and $R^{(k)}$ is upper triangular. Notice that $A^{(k+1)} = \left(Q^{(k)}\right)^T A^{(k)} Q^{(k)}$, all matrices in the sequence $\left\{A^{(k)}\right\}$ are orthogonally similar to each other. Since (2.2.1) can be interpreted as an orthogonal iteration, under mild assumption of A, $\left\{A^{(k)}\right\}$ converges to a block upper triangular form, and each diagonal block corresponds to a set of eigenvalues with the same magnitude [42]. Moreover, it can be shown [42, 58] that

$$\left\| \left(A^{(k)}\right)_{(p+1:N,1:p)} \right\| = \mathcal{O}\left(\left| \frac{\lambda_{p+1}}{\lambda_p} \right|^k \right), \qquad (p = 1, 2, \ldots, N-1)$$

where λ_j is the jth largest eigenvalue in magnitude ($j = 1, 2, \ldots, N$). To accelerate the convergence, the shifted QR iteration

$$A^{(k)} - \mu^{(k)} I = Q^{(k)} R^{(k)}, \qquad A^{(k+1)} = \left(Q^{(k)}\right)^T A^{(k)} Q^{(k)}, \tag{2.2.2}$$

can be applied, where $\mu^{(k)}$ is a *shift* which is usually chosen to approximate an eigenvalue of A. For example, if we choose a stationary shift $\mu^{(k)} = \mu$ which is close to a certain eigenvalue λ_j, then

$$\left(A^{(k)}\right)_{(N,N)} \to \lambda_j \quad \text{and} \quad \left\| \left(A^{(k)}\right)_{(N,1:N-1)} \right\| = \mathcal{O}(\rho^k),$$

provided that

$$\rho = \max_{i \neq j} \left| \frac{\lambda_j - \mu}{\lambda_i - \mu} \right| < 1.$$

Hence, the convergence of λ_j is accelerated by choosing a shift which is sufficiently close to λ_j.

Hessenberg reduction. A drawback of (2.2.1) and (2.2.2) is that in general each iterate requires $\Theta(N^3)$ arithmetic operations for the QR decomposition. To reduce the cost, a preprocessing step is to reduce A to an *upper Hessenberg matrix*[3] $H = Q_0^T A Q_0$. This preprocessing step can be accomplished by using $N-2$ *Householder reflections* (see Appendix A). Figure 2.1 illustrates the procedure of reducing a full matrix to an upper Hessenberg matrix. We remark that Q_0 is of the form $Q_0 = \mathrm{Diag}\{1, V_0\}$ and hence $(Q_0)_{(:,1)} = Q_0 e_1 = e_1$. In LAPACK, the routine `DGEHRD` implements a blocked version of the Hessenberg reduction process, see [45, 121] for details.

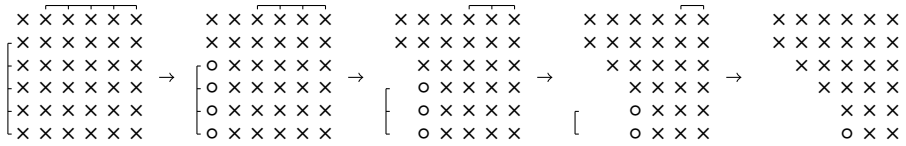


Figure 2.1 – Reducing a full matrix to an upper Hessenberg matrix ($N = 6$).

Once the matrix is reduced to an upper Hessenberg matrix, the nonzero pattern is then preserved in (2.2.2); the computational cost also becomes much cheaper compared to that for a full matrix. As all nonzero entries in H below the diagonal are on the subdiagonal, the QR decomposition of H can be obtained by applying $N-1$ Givens rotations. Consequently, (2.2.1) and (2.2.2) can both be performed using $\Theta(N^2)$ arithmetic operations, see Figure 2.2.

Another advantage of Hessenberg matrices is that the checking of convergence also be-

3. A square matrix H is called upper Hessenberg if $h_{ij} = 0$ when $i > j + 1$.

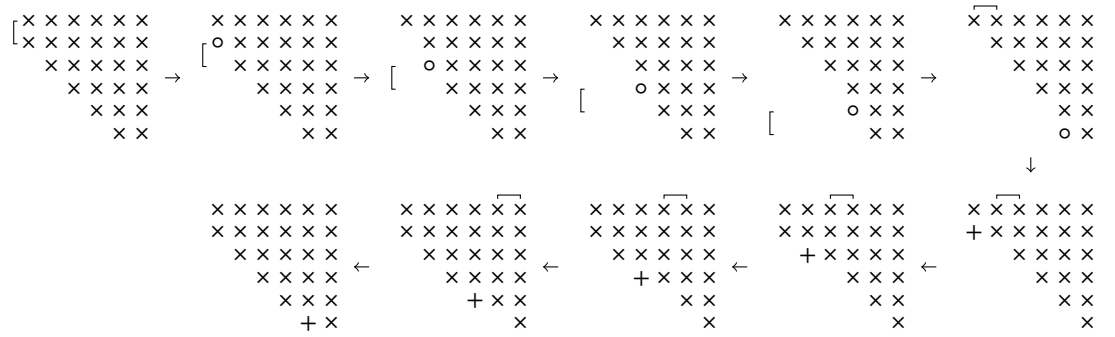


Figure 2.2 – One step QR iteration applied to an $N \times N$ upper Hessenberg matrix requires $2(N-1)$ Givens rotations ($N = 6$).

comes cheap. A practical criterion for setting a subdiagonal entry $h_{i+1,i}^{(k)}$ to zero is that both

$$\begin{cases} \left|h_{i+1,i}^{(k)}\right| \le \mathbf{u}\left(\left|h_{i,i}^{(k)}\right| + \left|h_{i+1,i+1}^{(k)}\right|\right), & \text{if } \left|h_{i,i}^{(k)}\right| + \left|h_{i+1,i+1}^{(k)}\right| > 0, \\ \left|h_{i+1,i}^{(k)}\right| \le \mathbf{u}\|A\|_2, & \text{if } \left|h_{i,i}^{(k)}\right| = \left|h_{i+1,i+1}^{(k)}\right| = 0, \end{cases}$$

and

$$\left|h_{i+1,i}^{(k)}\right|\left|h_{i,i+1}^{(k)}\right| \le \mathbf{u}\left|h_{i+1,i+1}^{(k)}\right|\left|h_{i+1,i+1}^{(k)} - h_{i,i}^{(k)}\right|$$

are satisfied (see [3, 163]). Once a subdiagonal entry is set to zero, the upper Hessenberg matrix becomes

$$\begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix},$$

which is a block upper triangular matrix. Then the problem is reduced to computing Schur forms of two smaller matrices H_{11} and H_{22}.

Francis double-shift QR algorithm. Another important technique proposed by Francis [50] is the so-called *implicit QR iteration*, which avoids explicit formulation of the QR decomposition in the QR iteration. It is based on the following theorem.

Theorem 2.2.1 (Implicit Q Theorem [50])**.** *Let Q be an $N \times N$ orthogonal matrix with $H = Q^T AQ$ upper Hessenberg. Then $Q_{(:,2)}, \ldots, Q_{(:,s)}$ and $h_{2,1}, \ldots, h_{s+1,s}$ (we formally define $h_{N+1,N} = 0$) are uniquely (up to signs) determined by $Q_{(:,1)}$, where*

$$s = \min\left\{i : h_{i+1,i} = 0\right\}.$$

Proof. See, e.g., [58, Theorem 7.4.2]. □

The implicit Q theorem provides a lot of freedom in how (2.2.2) is performed. More im-

portantly, it allows us to merge several steps of (2.2.2) into one iterate in a cheap manner. Consider m steps of the QR iteration on an unreduced [4] upper Hessenberg matrix H:

$$\begin{aligned}
&H^{(k)} - \mu^{(k)} I = Q^{(k)} R^{(k)},\\
&H^{(k+1)} = \left(Q^{(k)}\right)^T H^{(k)} Q^{(k)},\\
&H^{(k+1)} - \mu^{(k+1)} I = Q^{(k+1)} R^{(k+1)},\\
&H^{(k+2)} = \left(Q^{(k+1)}\right)^T H^{(k+1)} Q^{(k+1)},\\
&\cdots\\
&H^{(k+m-1)} - \mu^{(k+m-1)} I = Q^{(k+m-1)} R^{(k+m-1)},\\
&H^{(k+m)} = \left(Q^{(k+m-1)}\right)^T H^{(k+m-1)} Q^{(k+m-1)}.
\end{aligned}$$

We define

$$p^{(k)}(t) = \left(t - \mu^{(k)}\right) \cdots \left(t - \mu^{(k+m-1)}\right)$$

as the *shift polynomial*. It can then be shown [58, Section 7.5] that

$$p^{(k)}\left(H^{(k)}\right) = \left(Q^{(k)} \cdots Q^{(k+m-1)}\right)\left(R^{(k+m-1)} \cdots R^{(k)}\right) =: \tilde{Q}^{(k)} \tilde{R}^{(k)}.$$

Therefore, it is equivalent to evaluate

$$p^{(k)}\left(H^{(k)}\right) = \tilde{Q}^{(k)} \tilde{R}^{(k)}, \qquad H^{(k+m)} = \left(\tilde{Q}^{(k)}\right)^T H^{(k)} \tilde{Q}^{(k)}. \tag{2.2.3}$$

Notice that when $H^{(k)}$ is upper Hessenberg, only the first $m+1$ entries of $\tilde{Q}^{(k)} e_1 = p^{(k)}(H^{(k)}) e_1 / \tilde{r}_{11}$ can be nonzero. Then (2.2.3) is performed in the following way:

(1) Construct a Householder reflection $U^{(k,0)}$ with $U^{(k,0)} e_1$ being parallel to $p^{(k)}(H^{(k)}) e_1$.

(2) Reduce $\left(U^{(k,0)}\right)^T H^{(k)} U^{(k,0)}$ to an upper Hessenberg matrix using $N-2$ Householder reflections of the form $U^{(k,j)} = \mathrm{Diag}\left\{1, V^{(k,j)}\right\}$.

This procedure is called a *bulge chasing* process or a *QR sweep*. Figure 2.3 provides a pictorial illustration of a QR sweep with $m = 2$. Let

$$U^{(k)} = U^{(k,0)} U^{(k,1)} \cdots U^{(k,N-2)}.$$

As $U^{(k)} e_1 = U^{(k,0)} e_1$, by the implicit Q theorem, $U^{(k)}$ and $Q^{(k)}$ are essentially the same in the sense that $U^{(k)}\left(Q^{(k)}\right)^T$ is a diagonal orthogonal matrix (i.e., $U^{(k)} = Q^{(k)} \mathrm{Diag}\{\pm 1, \pm 1, \ldots, \pm 1\}$). Therefore, a QR sweep indeed accomplishes (2.2.3). In practice $p^{(k)}(t)$ is often chosen as the characteristic polynomial of the $N_{\text{shift}} \times N_{\text{shift}}$ trailing principal submatrix of $A^{(k)}$, where $N_{\text{shift}} = m$ is a positive integer. The strategy with $m = 2$ (i.e., double-shift) was first proposed by Francis [50] to avoid complex arithmetic when applying a conjugate pair of complex shifts to a real matrix. Algorithm 2.1 summarizes the procedure of Francis double-shift QR algorithm.

4. An $N \times N$ upper Hessenberg matrix H is called unreduced if $h_{i+1,i} \neq 0$ for $i = 1, 2, \ldots, N-1$.

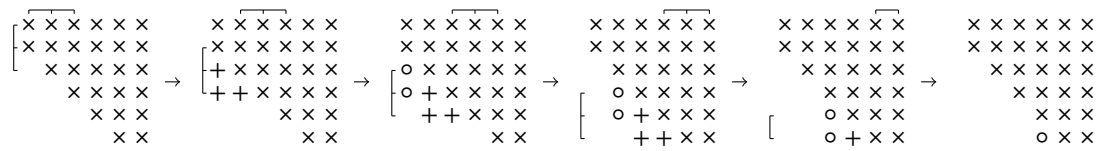


Figure 2.3 – A double-shift QR sweep ($N = 6$).

Algorithm 2.1 Francis double-shift QR algorithm

Input: $A \in \mathbb{R}^{N\times N}$.
Output: A real Schur form of A.
1: Reduce A to an upper Hessenberg matrix H.
2: **while** not converged **do**
3: Find the bottommost unreduced diagonal block $H_{(p:q,p:q)}$.
4: Use $\det\big(tI - H_{(q-1:q,q-1:q)}\big)$ as the shift polynomial and perform a double-shift QR sweep on $H_{(p:q,p:q)}$.
5: Set negligible subdiagonal entries to zeros.
6: Standardize deflated 2×2 diagonal blocks.
7: **end while**

2.2.2 Multishift QR sweeps

Concerning the memory hierarchy on modern computer architectures, Francis double-shift QR algorithm has an obvious drawback—most operations are performed with level 1 *computational intensity*[5] and have relatively low performance. To overcome this drawback, an early attempt is to introduce multiple shifts in the QR algorithm rather than double shifts, i.e., chase a larger bulge [10]. Unfortunately, the quality of transmitted shifts becomes poor due to roundoff when chasing a large bulge [157, 158]. This effect degrades the convergence of the QR algorithm. Therefore, the size of the bulge has to be small enough to avoid numerical instability in the bulge chasing process. Prior to LAPACK version 3.1, the routine `DHSEQR` implements such a multishift QR algorithm with up to six simultaneous shifts. The computational intensity is improved a bit, but is still between level 1 and level 2.

To further increase the computational intensity, Braman et al. [30] and Lang [100] proposed the following strategy which introduces more shifts and achieves level 3 performance. It consists of three stages.

Bulge introduction. The first stage is to introduce a *tightly-coupled* chain of $N_{\text{shift}}/2$ bulges—each bulge contains a pair of shifts, see Figure 2.4(a). These bulges are introduced and chased to appropriate positions one by one so that no bulge is chased across another one. The validness of such a strategy is ensured by the implicit Q theorem. The Householder reflections are only applied within the top-left diagonal block. Then the orthogonal matrix, which accu-

5. The computational intensity is defined as the ratio between the number of arithmetic operations and the number of memory access operations.

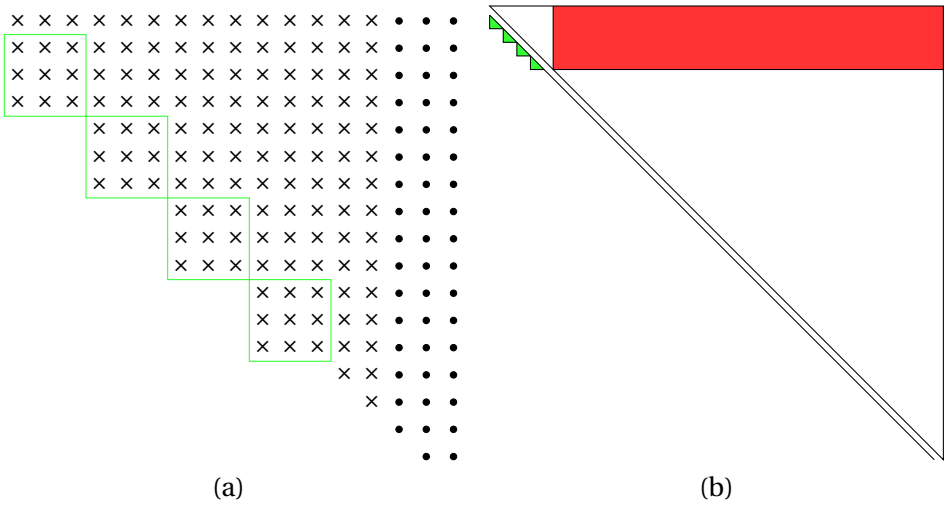


Figure 2.4 – Introducing a tightly-coupled chain of bulges ($N_{\text{shift}} = 8$).

mulates all involved Householder reflections, is explicitly multiplied with the corresponding off-diagonal block, see Figure 2.4(b). Therefore, most arithmetic operations in this stage are performed using level 3 basic linear algebra subprograms (BLAS) [101].

Bulge chasing. In the second stage, the chain of bulges are chased from the top-left corner to the bottom-right corner in many rounds. In each round, the chain of bulges are chased M steps down towards the bottom-right corner. Again, Householder reflections are only applied within the diagonal block where the chain of bulges are involved; the corresponding off-diagonal blocks are updated by explicit multiplication with the orthogonal matrix accumulated in the diagonal chasing step, see Figure 2.5. In practice $M = 3N_{\text{shift}}/2$ is adopted so that the number of arithmetic operations is nearly minimized [30]. Such a *delay-and-accumulate* technique yields level 3 performance when updating the off-diagonal blocks. We remark that the accumulated orthogonal matrix in each round has a special banded structure. Taking advantage of this structure sometimes reduces the cost of matrix multiplications, see [30].

Bulge annihilating. The final stage is to annihilate the bulges one by one at the bottom-right corner of the matrix. Similar to the other two stages, the update of the off-diagonal block is delayed until the accumulated orthogonal matrix is explicitly formed.

The three-stage procedure discussed above is called a *small-bulge multishift* QR sweep. A tightly-coupled chain of small bulges retains the numerical stability in the bulge-chasing process. Although the number of arithmetic operations is roughly doubled [30] compared to $N_{\text{shift}}/2$ rounds of Francis double-shift QR sweeps, the performance is significantly improved since this multishift QR algorithm makes intensive use of level 3 BLAS operations. We make two further remarks on the tightness of the bulge chain. First, loosely-coupled chains of bulges can also be useful in practice, especially when parallelizing the QR algorithm. Detailed discussion

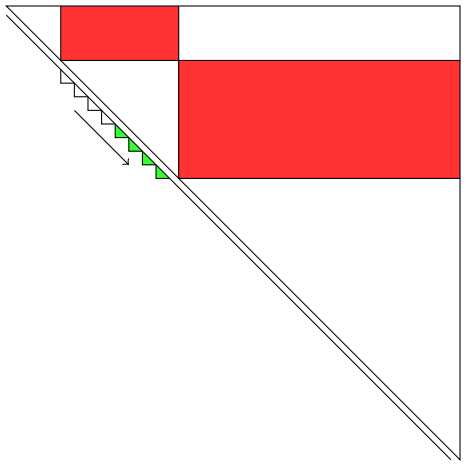


Figure 2.5 – The delay-and-accumulate technique in the bulge chasing stage ($N_{\text{shift}} = 8$).

will be provided in Section 3.2. Second, the tightness of the bulge chain presented here is still not optimal. For advanced developments in this direction, we refer to [92].

2.2.3 Aggressive early deflation

Aggressive early deflation (AED) has been proposed by Braman et al. [31] aiming at accelerating the convergence of the QR algorithm. It proceeds by partitioning the current upper Hessenberg matrix $H \in \mathbb{R}^{N\times N}$ as

$$H = \begin{array}{c} \\ N-N_{\text{AED}}-1 \\ 1 \\ N_{\text{AED}} \end{array} \begin{array}{c} \begin{array}{ccc} N-N_{\text{AED}}-1 & 1 & N_{\text{AED}} \end{array} \\ \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ 0 & H_{32} & H_{33} \end{bmatrix} \end{array},$$

where $H_{33} \in \mathbb{R}^{N_{\text{AED}}\times N_{\text{AED}}}$ is the so called *AED window*. By computing the (real) Schur decomposition $H_{33} = VTV^T$, and applying the corresponding similarity transformation to H, we obtain

$$U^T HU = \begin{bmatrix} H_{11} & H_{12} & H_{13}V \\ H_{21} & H_{22} & H_{23}V \\ 0 & s & T \end{bmatrix},$$

where

$$U = \begin{array}{c} \\ N-N_{\text{AED}}-1 \\ 1 \\ N_{\text{AED}} \end{array} \begin{array}{c} \begin{array}{ccc} N-N_{\text{AED}}-1 & 1 & N_{\text{AED}} \end{array} \\ \begin{bmatrix} I & & \\ & 1 & \\ & & V \end{bmatrix} \end{array}.$$

The vector $s \in \mathbb{R}^{N_{\text{AED}}}$ is the so called *spike*, created from the first entry of the vector H_{32}. The last diagonal entry (or 2×2 diagonal block) of T can be deflated if the magnitude of the last component (or the last two components) of the spike is negligible. Undeflatable eigenvalues are moved to the top left corner of T by a swapping algorithm, see Section 2.1. The orthogonal transformations for reordering eigenvalues in the Schur form of the AED window are accumulated in an $N_{\text{AED}} \times N_{\text{AED}}$ orthogonal matrix. By repeating the same procedure to all diagonal entries (or 2×2 blocks) of T, the eigenvalues of T are checked subsequently and possibly deflated. Then the entire matrix is reduced back to upper Hessenberg and the off-diagonal blocks H_{13} and H_{23} are multiplied by $\tilde{V}$, the product of all involved orthogonal transformations. Figure 2.6 illustrates the whole procedure of AED.

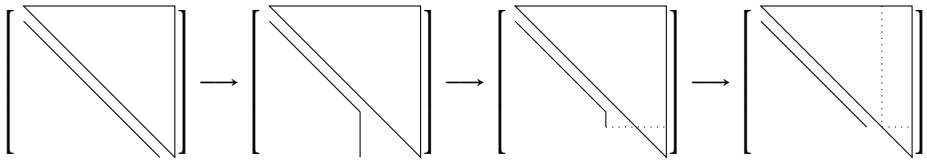


Figure 2.6 – A pictorial illustration of aggressive early deflation.

In practice, an AED step is performed before a multishift QR sweep. Typically, the size of the AED window (N_{AED}) is recommended to be somewhat larger, e.g., by 50%, than the number of shifts (N_{shift}) in the multishift QR sweeps [30, 32]. Then undeflatable eigenvalues can be used as shifts in the subsequent QR sweep. We remark that in case an AED step is efficient enough in the sense that a reasonably large fraction (e.g., 15%) of eigenvalues has been deflated, it is advisable to skip the QR sweep and perform another AED. The multishift QR algorithm with aggressive early deflation discussed in this subsection is summarized as Algorithm 2.2. An implementation of Algorithm 2.2 is available in the routine `DHSEQR` since LAPACK version 3.1.

Algorithm 2.2 Multishift QR algorithm with aggressive early deflation

Input: $H \in \mathbb{R}^{N \times N}$, H is upper Hessenberg.
Output: A real Schur form of H.

1: **while** not converged **do**
2: Perform AED on the $N_{\text{AED}} \times N_{\text{AED}}$ trailing principle submatrix.
3: Apply the accumulated orthogonal transformation to the corresponding off-diagonal blocks.
4: **if** a large fraction of eigenvalues has been deflated in Step 2 **then**
5: **goto** Step 2.
6: **end if**
7: Perform a small-bulge multishift QR sweep with N_{shift} undeflatable eigenvalues obtained from Step 2 as shifts.
8: Set negligible subdiagonal entries to zeros.
9: Standardize deflated 2×2 diagonal blocks.
10: **end while**

There is a close relationship between AED and the Krylov-Schur method [142], see [95,

97]. Such a relationship leads to an explanation of the effectiveness of AED—even if the subdiagonal entry $|h_{N-N_{\mathrm{AED}}+1,N-N_{\mathrm{AED}}}|$ is not small, indicating that the last N_{AED} columns in the current orthogonal transformation matrix do not span a left invariant subspace of A, it is still possible to exploit potentially converged Ritz vectors from this subspace. We refer to [97] for detailed discussions. A simple application of AED for Hermitian matrices will be presented in Chapter 6.

In principle, aggressive early deflation can be incorporated into any variant of the QR algorithm. It has been observed [31, 95] that AED dramatically accelerates the convergence of both Francis double-shift QR algorithm and the multishift QR algorithm. We will see another variant of the QR algorithm equipped with AED in Section 3.3.2.

3 The Parallel QR Algorithm with Aggressive Early Deflation

When solving large-scale dense nonsymmetric eigenvalue problems, it is naturally desirable to parallelize the QR algorithm to handle large-scale problems. In the parallel setting, the QR algorithm for dense nonsymmetric eigenvalue problems also consists of two stages:

(1) Reduce a full matrix A to an upper Hessenberg matrix H, i.e., $H = Q_0^T A Q_0$.

(2) Iteratively reduce H to a (real) Schur form, i.e., $T = Z^T H Z$.

For the first stage, we refer to [33] for the parallel Hessenberg reduction algorithm which is implemented in the ScaLAPACK routine PDGEHRD, and to [87, 91, 146] for recent developments in this direction. In this thesis we only focus on the second stage, i.e., how to further reduce an upper Hessenberg matrix to a real Schur form on *distributed memory architectures*. Figure 3.1 shows the software hierarchy of our implementation of the parallel multishift QR algorithm, which is built on Algorithm 2.2 presented in Chapter 2. Details regarding the parallel algorithm and some implementation issues are discussed in this chapter. To avoid ambiguity, the terminology "QR algorithm" only refers to the second stage throughout this chapter.

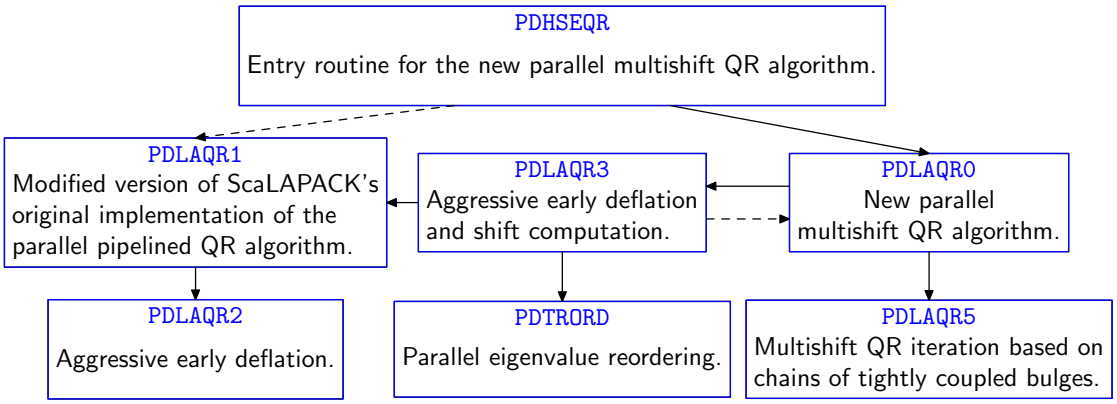


Figure 3.1 – Software hierarchy of the parallel multishift QR algorithm with AED.

The rest of this chapter is largely based on the manuscript [62] submitted for publication in *ACM Trans. Math. Software*. It is organized as follows. In Section 3.1, we first briefly recall ScaLAPACK's data layout convention which is adopted in our library software. Then

in Sections 3.2–3.4, we present the parallel multishift QR algorithm with AED. More detailed, Section 3.2 and 3.3 discuss parallel implementations of QR sweeps and AED, respectively. The criterion of switching between QR sweeps and AED is provided in Section 3.4. In Section 3.5, we establish a performance model which provides insights into the cost of computations and communications. Then suggestions on the choice of the parameters are provided in Section 3.6. Finally, we evaluate the performance of our parallel multishift QR algorithm by a large set of numerical experiments. The user's guide of our parallel library software is provided in Appendix C.

3.1 Data layout convention in ScaLAPACK

In ScaLAPACK, the $p = p_r \cdot p_c$ processors are usually arranged into a $p_r \times p_c$ grid. Matrices are distributed over the rectangular processor grid in a *2D block-cyclic layout* with block size $M_b \times N_b$ (see an example in Figure 3.2). The information regarding the data layout is stored in an *array descriptor* so that the mapping between entries of the global matrix and their corresponding locations in the memory hierarchy can be established. We adopt ScaLAPACK's data layout convention and require that the $N \times N$ input matrices H and Z have identical data layout with square data blocks (i.e., $M_b = N_b$). However, the processor grid need not to be square unless explicitly specified.

(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)	(1,0)	(1,1)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)	(1,0)	(1,1)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)	(1,0)	(1,1)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)	(0,0)	(0,1)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)	(1,0)	(1,1)

Figure 3.2 – The 2D block-cyclic data layout across a 2×3 processor grid. For example, processor $(0,0)$ owns all highlighted blocks.

3.2 Parallel QR sweeps

Since the concept of aggressive early deflation is relatively recent compared to earlier developments of the parallel QR algorithm, a lot of efforts on the parallelization of the QR algorithm in 1980s–1990s focused on how to improve the performance in the bulge chasing stage. In the following we first briefly recall the pipelined QR algorithm implemented in the

ScaLAPACK routine PDLAHQR, and then discuss our new parallel approach to the multishift QR sweeps.

3.2.1 The pipelined QR algorithm

In [69], it has been shown that a scalable parallel variant of the QR algorithm must use at least $\Theta(\sqrt{p})$ shifts simultaneously. The pipelined QR algorithm is a scalable approach which realizes this level of concurrency. The basic idea of the pipelined QR algorithm is to introduce several bulges of shifts—each contains two shifts, and chase them in parallel. These bulges are *loosely-coupled* so that they can be chased simultaneously using different processors, see Figure 3.3. Such an idea has been studied by many authors, e.g., Heller and Ipsen [68], Stewart [141], van de Geijn [147, 148], and Watkins [156]. In practice, $N_{\text{shift}} = \Theta(\sqrt{p})$ shifts are chosen as the eigenvalues of the $N_{\text{shift}} \times N_{\text{shift}}$ trailing principal submatrix. Local quadratic convergence of this shifting strategy has been confirmed in [148, 156, 161]. A parallel implementation of the pipelined QR algorithm is available in ScaLAPACK since version 1.5 as routine PDLAHQR. For detailed discussion about the pipelined QR algorithm and its implementation, we refer to [70].

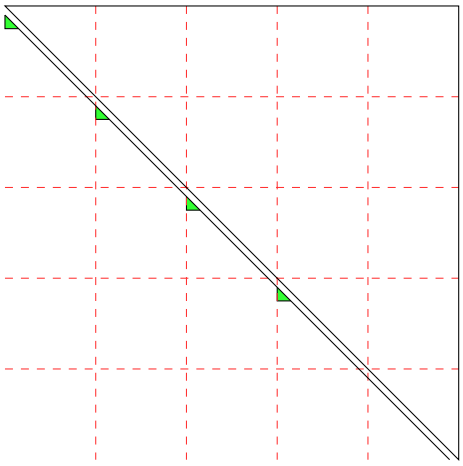


Figure 3.3 – The pipelined QR algorithm chases several loosely-coupled bulges in parallel. The dashed lines represent borders of the processor grid. Only parts of the matrix are displayed.

Although the pipelined QR algorithm clearly gives potential for parallelism, it comes with the disadvantage that its computational intensity is concentrated at level 1 and level 2 BLAS. In addition, frequent communication between processors is required, which causes the actual performance of the pipelined QR algorithm to not be very satisfactory. These shortcomings limit the practical application of the pipelined QR algorithm to only small- to medium-size matrices. In the next subsection, we discuss how to avoid these shortcomings and present an approach which is suitable for large-scale problems.

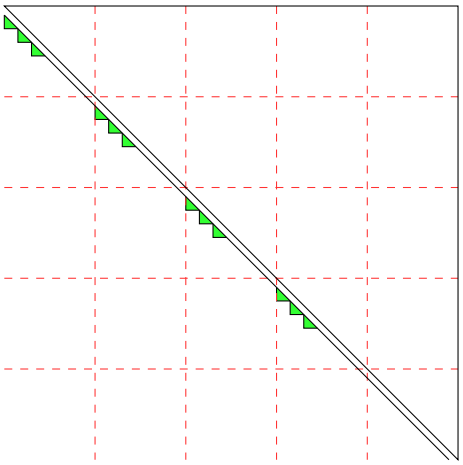


Figure 3.4 – Chains of tightly-coupled bulges are chased in the new parallel multishift QR algorithm.

3.2.2 New parallel multishift QR sweeps

As we have seen in the previous subsection, the main drawback of the pipelined QR algorithm is that it requires too frequent communication, not only between processors, but also within the local memory hierarchy. To overcome this drawback, we combine the idea of the pipelined QR algorithm with the small-bulge multishift QR algorithm. In our new parallel multishift QR algorithm implemented in the routine `PDLAQR5`, the remedy is to use several chains of *tightly-coupled* bulges, see Figure 3.4, so that local performance is improved and the communication is reduced [61].

Local chasing. The total number of shifts (N_{shift}) used in a single QR sweep is shown in Table 3.1, and is usually much larger compared to the pipelined approach. These shifts are divided into several chains of tightly-coupled bulges with up to $\lfloor N_b/3 \rfloor$ shifts per chain so that the length of each chain does not exceed $N_b/2$. The chains are placed on different diagonal blocks, such that $\Theta(\sqrt{p})$ chains can be chased locally and simultaneously. Once the bulge chasing within the diagonal blocks are performed, the accumulated orthogonal matrices are broadcasted to the corresponding rows/columns of processors. The off-diagonal blocks are then updated by explicit multiplication with these orthogonal matrices. The degree of concurrency of off-diagonal updating is $\Theta(p)$, just like that in the pipelined QR algorithm, while the computational intensity is improved to level 3 by the delay-and-accumulate technique.

Crossborder chasing. Once a local chasing step is performed, the chains of bulges reach the bottom-right corners of the diagonal blocks. We then need to perform a *crossborder* chasing so that each chain is moved to the next diagonal block. Compared to the pipelined QR algorithm where only six rows/columns are involved for each bulge in the crossborder bulge chasing stage, the situation in the new parallel multishift QR algorithm is considerably more complicated and thus requires extra care.

Table 3.1 – Recommended values for N_{shift} and N_{AED}.

Matrix size (N)	N_{shift}	N_{AED}
<6K	see [32]	
6K–12K	256	384
12K–24K	512	768
24K–48K	1024	1536
48K–96K	2048	3072
96K–192K	4096	6144
192K–384K	8192	12288
384K–768K	16384	24576
768K–1000K	32768	49152
> 1M	$\lceil N/25 \rceil$	$3N_{\text{shift}}/2$

We first discuss the crossborder chasing of a single chain. Let $2m$ be the number of shifts of the bulge chain. Consider the following $(6m+2) \times (6m+2)$ diagonal block

$$D = \begin{array}{c} \\ 3m+1 \\ 3m+1 \end{array} \begin{array}{c} \begin{array}{cc} 3m+1 & 3m+1 \end{array} \\ \begin{bmatrix} H_{kk} & H_{k,k+1} \\ H_{k+1,k} & H_{k+1,k+1} \end{bmatrix} \end{array},$$

where H_{ij} is owned by the processor P_{ij} (for $i, j \in \{k, k+1\}$). Suppose we are about to chase the chain of bulges from H_{kk} to $H_{k+1,k+1}$. First, processor P_{kk} collects $H_{k,k+1}$, $H_{k+1,k}$, and $H_{k+1,k+1}$ from other processors. Then processor P_{kk}, which now owns a copy of D, performs the desired chasing locally. Finally, each block of the new $(6m+2) \times (6m+2)$ matrix is sent back to its corresponding owner. The diagonal chasing is then finished. We remark that when the chain reaches the bottom-right corner of the whole unreduced upper Hessenberg matrix, the size of $H_{k+1,k+1}$ might be smaller than $(3m+1) \times (3m+1)$ so that there is not enough room within $H_{k+1,k+1}$ to receive the chain from H_{kk}. In this case the chain is chased off directly when performing crossborder diagonal chasing.

To update the corresponding off-diagonal blocks, the orthogonal matrix accumulated in the diagonal chasing stage is broadcasted to the corresponding rows/columns of processors which are involved in off-diagonal updating. Then each involved processor exchanges data blocks with its neighbor, see Figure 3.5. Finally the off-diagonal blocks are updated by explicit multiplication with the accumulated orthogonal matrix.

An important difference compared to local chasing is that in the crossborder chasing stage we do *not* chase all bulge chains simultaneously. To avoid conflicts between different tightly-coupled chains, the chains are chased in an odd-even manner when passing through the processor border, see Figure 3.6. Then no processor needs to send and receive two different chains at the same time. The degree of parallelism still remains $\Theta(p)$.

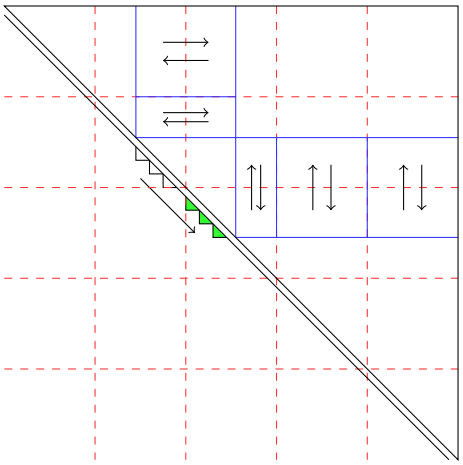


Figure 3.5 – Exchange data blocks with neighbors when updating off-diagonal blocks in the crossborder bulge chasing stage.

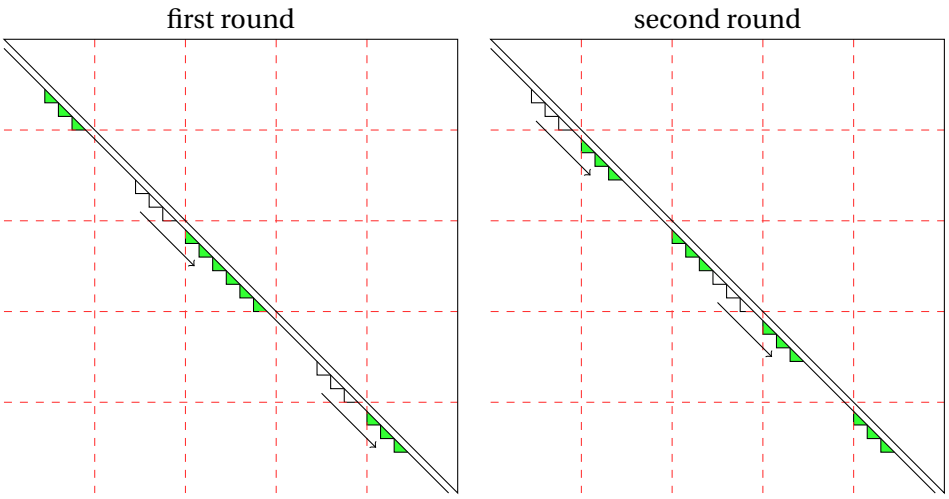


Figure 3.6 – Crossborder bulge chasing. Odd-numbered chains (left) and even-numbered chains (right) are chased separately in two rounds.

Parallel eigenvalue reordering. Finally, we remark that reordering eigenvalues in a Schur form can be implemented in a similar manner. The high level structure of the parallel algorithm stays the same—distribute several chains of eigenvalues (which are about to be reordered) on different diagonal blocks and reorder them in parallel. The only difference is that within each diagonal block an eigenvalue swapping algorithm (see Section 2.1) instead of a bulge chasing algorithm is applied. We refer to [60] for detailed descriptions. This parallel eigenvalue reordering algorithm, implemented in the routine `PDTRORD`, will be used in the parallel AED stage in our new parallel multishift QR algorithm.

3.3 Aggressive early deflation

As pointed out in Section 2.2.3, aggressive early deflation can be incorporated into any variant of the QR algorithm. Therefore both the new parallel multishift QR algorithm and the pipelined QR algorithm benefit from performing AED. As the efficient implementation of AED requires some care, we will now discuss these two settings in more detail.

3.3.1 AED within the new multishift QR algorithm

As this setting will be used for targeting large-scale problems, the AED window can be expected to become quite large. It is therefore not reasonable to expect that executing AED locally and sequentially yields good performance. Hence, the corresponding routine `PDLAQR3` for performing AED requires a parallel approach.

The first and most costly step of AED is to calculate the Schur decomposition of the AED window, i.e., the $N_{\text{AED}} \times N_{\text{AED}}$ trailing principal submatrix of H. This eigenvalue problem can be solved by either recursively using the new multishift QR algorithm (`PDLAQR0`) or using the pipelined QR algorithm (`PDLAQR1`). The choice of the solver is determined by the size of the AED window as well as the number of processors used. Since N_{AED} is relatively small compared to N, the number of available processors may be too large to facilitate all of them without causing significant communication overhead. In this case, we only use a subset of the processors to reduce the overhead and approximately minimize the execution time. See Section 3.3.3 for a more detailed discussion.

In the deflation checking phase, the reordering algorithm is arranged in a blocked manner, to reduce memory transfers and communication. Unlike the procedure described in [30], undeflatable eigenvalues are not moved immediately and individually towards the top left corner of the AED window. Instead, they are first reordered within an $N_b \times N_b$ computational window. Only after all eigenvalues in this $N_b \times N_b$ window are checked, the group of undeflatable eigenvalues is moved simultaneously to the top left corner of the AED window. This blocked approach increases the computational intensity and avoids the frequent communication needed when reordering each eigenvalue individually. The procedure is repeated until all eigenvalues in the AED window are checked.

The last step is to eliminate the spike and reduce the AED window back to the upper Hessenberg form. This task is performed with the ScaLAPACK routine `PDGEHRD`. The corresponding off-diagonal blocks are updated by explicitly multiplying the accumulated orthogonal matrix using the PBLAS routine `PDGEMM`. The whole parallel AED algorithm is summarized in Algorithm 3.1. The routine `PDLAQR3` in our software implements this algorithm.

Algorithm 3.1 Parallel aggressive early deflation

Input: $H \in \mathbb{R}^{N\times N}$ is upper Hessenberg.

1: (optional) Redistribute the AED window to a subset of processors.
2: Compute the Schur decomposition of the AED window (using a subset of processors).
3: (optional) Redistribute Schur decomposition of the AED window back to the orginal process grid.
4: **repeat**
5: Check deflation for the last N_b eigenvalues; undeflatable eigenvalues are moved to the top-left corner of this $N_b \times N_b$ block.
6: Move undeflatable eigenvalues in this group of N_b eigenvalues to the top-left corner of the AED window.
7: **until** all eigenvalues within the AED window are tested
8: Eliminate the spike and reduce the matrix back to an upper Hessenberg matrix.

3.3.2 AED within the pipelined QR algorithm

The original ScaLAPACK (version 1.8.0) implementation `PDLAHQR` of the pipelined QR algorithm is not equipped with the AED strategy. In our software, we provide a new routine `PDLAQR1`, which is modified from `PDLAHQR` and implements Algorithm 3.2. When adding AED, we have taken into account that we will only use this routine for small- to medium-size (sub)matrices. In particular, we can expect the AED window to be sufficiently small such that AED can be performed on one processor efficiently, by using the LAPACK implementation of AED.

Algorithm 3.2 Parallel pipelined QR algorithm with AED

Input: $A \in \mathbb{R}^{N\times N}$.
Output: A real Schur form of A.

1: Reduce A to an upper Hessenberg matrix H.
2: **while** not converged **do**
3: Copy the $(N_{\text{AED}}+1)\times(N_{\text{AED}}+1)$ trailing submatrix to one processor and perform sequential AED on the $N_{\text{AED}} \times N_{\text{AED}}$ window.
4: Broadcast the output of AED to all involved processors and update the corresponding off-diagonal blocks in parallel.
5: **if** a large fraction of eigenvalues has been deflated in Step 3 **then**
6: **goto** Step 3.
7: **end if**
8: Compute the eigenvalues of the $(N_{\text{AED}}+1)\times(N_{\text{AED}}+1)$ trailing submatrix sequentially.
9: Perform a pipelined QR sweep using N_{shift} shifts computed in Step 8.
10: Set negligible subdiagonal entries to zeros.
11: Standardize deflated 2×2 diagonal blocks.
12: **end while**

Apart from AED, our new routine `PDLAQR1` incorporates further modifications to `PDLAHQR`, making it both faster and more robust. In the following we summarize the most important

aspects; additional details can be found in [88, 132].

Aggressive early deflation. AED is implemented in an auxiliary routine PDLAQR2 which copies the trailing $(N_{\text{AED}}+1) \times (N_{\text{AED}}+1)$ submatrix to local memory and calls the LAPACK routine DLAQR3 to solve the problem sequentially. To determine the algorithmic parameters of AED, we use the settings of the LAPACK installation determined by ILAENV. However, a notable difference is that we do *not* use the undeflatable eigenvalues from the AED step as shifts in the subsequent pipelined QR sweep. Instead, we recompute the eigenvalues of the trailing $(N_{\text{AED}}+1) \times (N_{\text{AED}}+1)$ submatrix and (randomly) take N_{shift} of them as shifts. We have observed that this shifting strategy improves the quality of the shifts and accelerates the convergence of the pipelined QR algorithm. The computation of shifts are also performed locally and sequentially.

Conventional deflation. In PDLAHQR, pipelined QR sweeps are performed until the very end, that is, the remaining diagonal blocks are all of size 1×1 or 2×2. In PDLAQR1, we use a different strategy: Once the active block is sufficiently small (say, not larger than 385×385), we copy this block to local memory and call the LAPACK routines DLAHQR/DLAQR4 to solve the problem sequentially. This strategy significantly reduces communication overhead in the latter stages and is implemented in an auxiliary routine PDLAQR4.

Avoidance of anomalies. The original ScaLAPACK routine PDLAHQR suffered from two anomalies, which have been removed. First, the routine sometimes returned 2×2 diagonal blocks containing real eigenvalues, which is not in accordance with the specification of the interface. In PDLAQR1, each 2×2 diagonal block contains a conjugate pair of complex eigenvalues. This change is also helpful when checking deflation in the AED procedure. The second issue is concerned with a "deflation" strategy already proposed by Francis [50], which allows to introduce bulges below the top left corner of the active submatrix if there are two consecutive small subdiagonal entries. However, this turns out to be difficult to implement in a safe manner in pipelined or multishift QR sweeps, see [132] for detailed explanations. When using the ScaLAPACK routine PDLACONSB which implements this strategy, we have observed large relative residuals of norm up to 10^{-5}, indicating numerical instabilities. As the performance improvements gained from this strategy are usually negligible, we have decided to remove it. In return, the numerical stability is improved.

3.3.3 Avoiding communication via data redistribution

As observed in [88], the parallel QR algorithm is not efficient at solving relatively small eigenvalue problem on many processors, due to excessive communication, to the point that the execution time actually increases when increasing the number of processors. Such a situation is regularly encountered when calculating the Schur decomposition of the AED

window. Therefore, the computation of the Schur decomposition of the AED window often became a bottleneck in the previous implementation of parallel multishift QR algorithm [61].

Recent work on communication avoiding algorithms [16, 15, 76, 81] usually focuses on the design of algorithms that can attain the theoretical lower bounds of the communication cost. A basic assumption in these theoretical analyses is that the data are nearly evenly distributed over the processors. Here we propose an alternative approach, which does not rely on this assumption and is especially useful for operations involving smaller submatrices.

We first consider a simple and extreme case. Suppose there is one processor which has a large amount of local memory and very high clock speed. Then by gathering all data to this processor, the problem can be solved without further communication. Once the computation is completed, the data are scattered to their original owners. The total amount of communication does not exceed the cost of scattering and gathering regardless of the complexity of computational work. Although this simple idea does not work for large problems that cannot be stored on a single processor, it is still useful for smaller problems. For example, the AED process in the pipelined QR algorithm is implemented in such a manner since we know in advance that the AED window is always sufficiently small, such that the associated Schur decomposition can be efficiently solved sequentially. By introducing the overhead of data redistribution, the total amount of communication as well as the execution time are reduced.

For larger problems, it is not feasible to solve them sequentially via data redistribution. Specifically, the AED window within the new parallel multishift QR algorithm usually becomes quite large, although much smaller compared to the whole matrix. In this case, we choose a subset of processors instead of a single processor to perform AED. The data redistribution is performed using the routine `PDGEMR2D` in ScaLAPACK; its overhead has been observed to be negligible relative to the AED process as a whole.

The tunable parameter $p_{\min}$ = `PILAENVX(ISPEC=23)` determines our heuristic strategy for choosing the number of processors for the redistribution. If $\min(p_r, p_c) > p_{\min} + 1$, we redistribute the AED window to a $p_{\min} \times p_{\min}$ processor grid and perform the calculations on this subset of processors. The same strategy is also applied if we need to compute shifts after an AED step (see Section 3.4). The default value for this parameter is $p_{\min} = \lceil N_{\text{AED}}/(N_b\lceil 384/N_b\rceil)\rceil$, implying that each processor needs to own at least 384 columns of the AED window. The constant 384 has been obtained via extensive numerical experiments on one of our target architectures. It certainly needs adjustment for optimal performance on other architectures.

3.3.4 Task duplication—efficient but hazardous

Task duplication is a common technique in parallel computing to reduce communication and potentially improve performance, see, e.g., [59]. This technique has already been employed in previous implementations of the parallel QR algorithm, such as `PDLAHQR` and software developed in [61, 88]. However, a crucial assumption is made when applying this technique:

All involved processors need to produce identical outputs for identical tasks. Due to the effect of rounding error, this assumption is not always satisfied, especially on heterogeneous architectures.

This lack of numerical reproducibility is potentially harmful to the robustness of the parallel QR algorithm. As discussed in Section 3.3.2, all computations within the AED window are performed sequentially for the pipelined QR algorithm. In [88], these sequential parts are duplicated on all involved processors. The parallel update of the off-diagonal blocks can then be performed with the local copy of the orthogonal transformation matrix resulting from AED, without any extra communication. However, even the slightest change in finite precision arithmetic may lead to very different outputs produced by AED since QR sweeps are not forward stable [118, 157]. In particular, the ordering of the eigenvalues in the Schur decomposition computed within AED is very sensitive to such changes. In turn, the off-diagonal blocks are updated using completely different local copies of the orthogonal transformation matrices, leading to meaningless results. We have observed similar problems in crossborder bulge chasing and eigenvalue reordering. To avoid this, we use explicit communication rather than task duplication in the new implementation. For a moderate number of processors (e.g., $p \leq 100$), the change in performance is negligible; while for a large number of processors, the performance can drop. For example, for computing the Schur decomposition of a $100,000 \times 100,000$ matrix using 40×40 processors, up to 25% performance drop has been observed by replacing task duplication with explicit communication.

3.4 Switching between QR sweeps and AED

As we have pointed out in Section 2.2.3, there are rules for balancing the cost between multishift QR sweeps and AED in the QR algorithm (e.g., Step 4 in Algorithm 2.2 and Step 5 in Algorithm 3.2). The precise meaning of these rules is characterized by a threshold called `NIBBLE`. Let N_{undflt} denote the number of undeflatable shifts in an AED step. The multishift QR sweep is skipped if

$$\frac{N_{\text{AED}} - N_{\text{undflt}}}{N_{\text{AED}}} \geq \frac{\texttt{NIBBLE}}{100}.$$

Since AED behaves differently for different matrices, this strategy automatically adjusts the choice between AED and QR sweeps based on the properties of the matrix.

The default value of `NIBBLE` in the sequential LAPACK implementation is 14, which provides a good balance between multishift QR sweeps and AED. The same default value is used in our modified version of the pipelined QR algorithm with AED. However, in the new parallel multishift QR algorithm, the parallel AED process becomes substantially more expensive than the sequential AED process due to communication. As explained above, the AED process only involves a smaller trailing submatrix, leading to decreased parallel efficiency. To account for this, `NIBBLE` should be set larger to avoid performing AED too frequently. A good choice of this

threshold depends both on the size of the matrix H and the number of processors involved. We use the model NIBBLE $= a \cdot N^b p^c$ for this purpose, where a, b, and c are machine-dependent constants. An appropriate choice of these constants can be gained from repeated runs of the program with different thresholds. It turns out that the right choice of NIBBLE becomes rather sensitive when communication is slow. In our numerical experiments, the default values on our computing architectures are chosen as $(a, b, c) = (335, -0.44, 0.5)$.

A complication arises when using NIBBLE > 33. Such a choice may lead to situations where the number of undeflatable eigenvalues is less than the desired number of shifts, that is $N_{\text{undflt}} < N_{\text{shift}}$, due to the fact that $N_{\text{AED}} = 3N_{\text{shift}}/2$. The solution in the software is that as long as $N_{\text{undflt}} \geq N_{\text{shift}}/2$, we only use these N_{undflt} undeflatable eigenvalues as shifts in the subsequent QR sweep. However, the condition $N_{\text{undflt}} \geq N_{\text{shift}}/2$ may also fail when NIBBLE > 66. In this case we calculate the eigenvalues of the $N_{\text{shift}} \times N_{\text{shift}}$ trailing principal submatrix of H and use them as shifts. The calculation can be performed by either PDLAQR0 or PDLAQR1, just like computing the Schur decomposition in the AED step.

3.5 Performance model

In this section, we analyze the cost of computation and communication of the new parallel multishift QR algorithm for reducing a Hessenberg matrix to Schur form. For simplicity, we consider a square processor grid, that is, $p_r = p_c = \sqrt{p}$. In addition, we assume that each processor contains reasonably many data blocks of the matrices, i.e., $\sqrt{p}\, N_b \ll N$, so that the work load is balanced. The parallel execution time consists of two main components:

$$T_p = T_a + T_c,$$

where T_a and T_c are the times for arithmetic operations and communication, respectively. The possibility of overlapping communication with computations is not taken into account. By neglecting the communication between memory and cache lines inside one core, the serial runtime can be approximated by

$$T_a = \frac{\#(\text{flops})}{f(p)}\gamma,$$

where γ is the average time for performing one floating-point operation and $f(p)$ is the degree of concurrency. For the communication between two processors, we define α and β as the start-up time (or communication latency) and the time for transferring one word without latency (or reciprocal of bandwidth), respectively. The time for a single point-to-point communication is modeled as $\alpha + L\beta$ where L is the message size in words. A one-to-all broadcast or an all-to-one reduction within a scope of p processors is assumed to take $\Theta(\log p)$ steps.

Let k_{AED} and k_{QR} denote the number of AED steps and QR sweeps, respectively, performed by the new parallel multishift QR algorithm. We have $k_{\text{QR}} \leq k_{\text{AED}}$, since some QR sweeps

are skipped when the percentage of deflated eigenvalues in the AED step is larger than the threshold (`NIBBLE`). When the number of undeflatable eigenvalues from AED is not sufficient for performing the next QR sweep, we need to calculate shifts from the trailing submatrix. The number of extra calls to the parallel Schur decomposition solver in this case is denoted by k_{shift}, which of course satisfies $k_{\text{shift}} \le k_{\text{QR}}$. Given the constants k_{AED}, k_{QR}, and k_{shift}, the execution time of the parallel QR algorithm is modeled as the sum of the corresponding phases:

$$T(N,p) = k_{\text{AED}}\,T_{\text{AED}}(N,N_{\text{AED}},p) + k_{\text{QR}}\,T_{\text{QR}}(N,N_{\text{shift}},p) + k_{\text{shift}}\,T_{\text{shift}}(N,N_{\text{shift}},p),$$

where T_{AED}, T_{QR}, and T_{shift} are the runtimes for performing each phase once. For simplicity, it is assumed that N_{AED} and N_{shift} remain constant throughout the entire QR algorithm, and all QR sweeps act on the entire matrix. We further assume that AED is always performed on a $\sqrt{p_{\text{AED}}} \times \sqrt{p_{\text{AED}}}$ processor grid, so that the property $\sqrt{p_{\text{AED}}}\,N_b \ll N_{\text{AED}}$ is also valid inside the AED window. The same assumption is made for the shift calculation phase. A typical relationship among these parameters is

$$N_{\text{shift}} \approx \frac{2}{3}N_{\text{AED}} \approx \frac{1}{C_1}N \quad \text{and} \quad \frac{N_{\text{AED}}}{\sqrt{p_{\text{AED}}}} \approx \frac{N_{\text{shift}}}{\sqrt{p_{\text{shift}}}} \ge C_2,$$

where C_1 and C_2 are constants (e.g., $C_1 = 24$, $C_2 = 384$). In practice k_{AED}, k_{QR}, k_{shift} can vary a lot for different matrices. We assume $k_{\text{AED}} = \Theta(N/N_{\text{AED}}) = \Theta(C_1)$, which appears to be reasonable.

In the following we present performance models based on the assumptions above. Tiny terms, especially lower order terms with reasonably sized constants, are omitted.

3.5.1 Estimating T_{QR}

The QR sweep is relatively simple because the computation and communication cost is well-determined by N, N_{shift}, and p. Usually there are up to $\sqrt{p}$ simultaneous computational windows, one at each diagonal processor in the grid, with at most $N_b/3$ shifts in each window. If $N_{\text{shift}} > \sqrt{p}\,N_b/3$, these shifts are chased in several rounds. So we use a rough approximation $N^{\star}_{\text{shift}} = \sqrt{p}\,N_b/3$ to represent the total amount of shifts which can be chased simultaneously in the QR sweep. Based on the assumption $\sqrt{p}\,N_b \ll N$, the overhead for the start-up and ending phases of the bulge chasing are not important. Therefore the cost of one QR sweep is roughly

$$T_{\text{QR}}(N,N_{\text{shift}},p) = \frac{N_{\text{shift}}N}{N^{\star}_{\text{shift}}N_b}(T_{\text{local}} + T_{\text{cross}}),$$

where T_{local} and T_{cross} represent the runtime for local and crossborder bulge chasing, respectively. Both parts require chasing the chain of bulges with $N_b/2$ steps inside the computational window, as well as updating the corresponding off-diagonal blocks. Hence the runtime for arithmetic operations is $\left(4N_b^3 + 4NN_b^2/\sqrt{p}\right)\gamma$, half of which is for accumulating the orthogonal matrix Q. The only communication cost in the local chasing phase is broadcasting the accu-

mulated orthogonal matrix rowwise and columnwise in the processor grid, which requires $\log_2 p\,(\alpha + N_b^2\beta)$ runtime. Therefore,

$$T_{\text{local}} = \left(4N_b^3 + \frac{4NN_b^2}{\sqrt{p}}\right)\gamma + \log_2 p\,(\alpha + N_b^2\beta) \approx \frac{4NN_b^2}{\sqrt{p}}\gamma + \log_2 p\,(\alpha + N_b^2\beta).$$

One round crossborder chasing requires at least the same amount of communication as in one local chasing step, with some extra cost for explicitly forming the $N_b \times N_b$ computational window and exchanging data with processor neighbors for updating the off-diagonal blocks. Notice that usually there are two rounds for a crossborder chasing step, therefore we have

$$T_{\text{cross}} = 2\left[T_{\text{local}} + 3\left(\alpha + \frac{N_b^2}{4}\beta\right) + 3\left(\alpha + \frac{NN_b}{2\sqrt{p}}\beta\right)\right],$$

and then

$$\begin{aligned} T_{\text{QR}}(N, N_{\text{shift}}, p) &\approx \frac{12N^2 N_{\text{shift}} N_b}{\sqrt{p}\,N_{\text{shift}}^\star}\gamma + \frac{3NN_{\text{shift}}}{N_{\text{shift}}^\star N_b}(\log_2 p + 4)\alpha + \frac{3N^2 N_{\text{shift}}}{\sqrt{p}\,N_{\text{shift}}^\star}\beta \\ &= \frac{36N^2 N_{\text{shift}}}{p}\gamma + \frac{9NN_{\text{shift}}}{\sqrt{p}\,N_b^2}(\log_2 p + 4)\alpha + \frac{9N^2 N_{\text{shift}}}{pN_b}\beta. \end{aligned}$$

From this model, we can see that the cost for updating the off-diagonal blocks dominates in both the computation and communication parts, under the assumption that $\sqrt{p}\,N_b \ll N$ (or equivalently $N_{\text{shift}}^\star \ll N$). As a byproduct, the performance model of a plain multishift QR algorithm without AED can also be obtained. By assuming the convergence rate as $\Theta(1)$ shifts per eigenvalue, i.e., $k_{\text{QR}} = \Theta(N/N_{\text{shift}})$, and neglecting the cost for generating shifts, the total execution time of a plain multishift QR algorithm is

$$T(N,p) = \Theta\left(\frac{N^3}{p}\right)\gamma + \Theta\left(\frac{N^2 \log p}{\sqrt{p}\,N_b^2}\right)\alpha + \Theta\left(\frac{N^3}{pN_b}\right)\beta.$$

Fixing the memory load per processor (i.e., $N/\sqrt{p}$ = constant) yields

$$T(N,p) = \Theta(N)\,\gamma + \Theta(N\log N)\,\alpha + \Theta(N)\,\beta.$$

3.5.2 Estimating T_{AED} and T_{shift}

The execution time for one step AED is modeled as

$$\begin{aligned} T_{\text{AED}}(N, N_{\text{AED}}, p) = {} & T_{\text{redist}}(N_{\text{AED}}, p, p_{\text{AED}}) + T_{\text{Schur}}(N_{\text{AED}}, p_{\text{AED}}) \\ & + T_{\text{reorder}}(N_{\text{AED}}, p) + T_{\text{Hess}}(N_{\text{AED}}, p) + T_{\text{update}}(N, N_{\text{AED}}, p), \end{aligned}$$

where the terms in the right-hand-side represent the runtime for data redistribution, Schur decomposition of the AED window, deflation checking and reordering of eigenvalues, Hes-

senberg reduction, and updating the off-diagonal blocks corresponding to the AED window, respectively. We estimate these terms one by one using the hierarchical approach described in [37].

Estimating T_{redist}. The general purpose data redistribution routine `PDGEMR2D` in ScaLAPACK uses the algorithm described in [120]. Since the scheduling part is tiny compared to the communication part, the complexity of data redistribution is provided [120] as

$$T_{\text{redist}}(N_{\text{AED}}, p, p_{\text{AED}}) = \Theta(p)\,\alpha + \Theta\left(\frac{N_{\text{AED}}^2}{\sqrt{p\,p_{\text{AED}}}}\right)\beta.$$

Estimating T_{Schur}. The complexity of the Schur decomposition performed by `PDLAQR1` largely depends on the properties of the matrix, since AED affects the convergence rate significantly. To obtain an estimate of the complexity, we assume that AED roughly reduces the number of pipelined QR sweeps by half. According to the experimental results presented in [88], this assumption usually provides a reasonable upper bound of the runtime, although it can be overestimated. Using the model in [70], we obtain an approximate execution time

$$T_{\text{Schur}}(N, p) = \frac{20N^3}{p}\gamma + \frac{3N^2}{\sqrt{p}\,N_b}(\log_2 p + 2)\alpha + \left(\frac{3N^2\log_2 p}{\sqrt{p}} + \frac{8N^3}{pN_b}\right)\beta. \tag{3.5.1}$$

If the orthogonal matrix Q is not accumulated in the calculation, the arithmetic operations are roughly halved, i.e.,

$$\tilde{T}_{\text{Schur}}(N, p) = \frac{10N^3}{p}\gamma + \frac{3N^2}{\sqrt{p}\,N_b}(\log_2 p + 2)\alpha + \left(\frac{3N^2\log_2 p}{\sqrt{p}} + \frac{8N^3}{pN_b}\right)\beta.$$

The model provided in [26] is similar, but with slightly different coefficients.

Estimating T_{reorder}. Obviously, the cost for eigenvalue reordering depends on the deflation ratio. However, we can evaluate an upper bound for the cost—all eigenvalues are involved in the reordering. Then the performance model is almost the same as that of QR sweeps, since updating the off-diagonal blocks is the dominant operation. Notice that each eigenvalue needs to move $N_{\text{AED}}/2$ steps in average, so the overall cost for eigenvalue reordering inside the AED window is bounded by

$$T_{\text{reorder}}(N_{\text{AED}}, p) \approx \frac{4N_{\text{AED}}^2 N_b}{\sqrt{p}}\gamma + \frac{2N_{\text{AED}}}{N_b}(\log_2 p + 3)\alpha + \frac{3N_{\text{AED}}^2}{2\sqrt{p}}\beta.$$

As a different feature compared to QR sweeps or the performance model in [60] for parallel eigenvalue reordering, the degree of concurrency here is $\Theta(\sqrt{p})$ instead of $\Theta(p)$. The reason

for such a difference is that there is only one chain of up to N_b eigenvalues to be reordered and hence at most two diagonal blocks are involved for the reordering phase inside the AED window.

Estimating T_{Hess}. The Hessenberg reduction routine `PDGEHRD` uses the parallel algorithm described in [33]. Almost all computations and communication are performed on matrix-vector and matrix-matrix multiplications. Therefore we need to model these PBLAS operations first. The level 2 operations GEMV and GER require

$$T_{\text{GEMV}}(M,N,p) \approx T_{\text{GER}}(M,N,p) \approx \frac{2MN}{p}\gamma + \log_2 p\left(\alpha + \frac{M+N}{2\sqrt{p}}\beta\right),$$

where $M \times N$ is the size of the matrix. This model directly carries over to multiplying two $M \times K$ and $K \times N$ matrices as long as $\min\{M,N,K\} \le N_b$ since it is merely a "fat" level 2 operation. In the Hessenberg reduction algorithm, all level 3 operations are "fat" level 2 operations, so the cost for one GEMM operation can be modeled as

$$T_{\text{GEMM}}(M,N,N_b,p) \approx T_{\text{GEMM}}(M,N_b,N,p) \approx \frac{2MNN_b}{p}\gamma + \log_2 p\left(\alpha + \frac{(M+N)N_b}{2\sqrt{p}}\beta\right). \quad (3.5.2)$$

Using these simple models of the PBLAS operations, we are now ready to establish a model for T_{Hess}. The level 2 part consists roughly of N matrix-vector multiplications of dimension $N \times (N-j)$ (for $j = 1, 2, \ldots, N$). Therefore the cost is

$$T_{\text{level2}} = \sum_{j=1}^{N}\left[\frac{2N(N-j)}{p}\gamma + \log_2 p\left(\alpha + \frac{2N-j}{2\sqrt{p}}\right)\right] \approx \frac{N^3}{p}\gamma + \log_2 p\left(N\alpha + \frac{3N^2}{4\sqrt{p}}\beta\right).$$

The level 3 part contains roughly N/N_b iterations with one `PDGEMM` and one `PDLARFB` per iteration. Within the jth iteration ($j = 1, 2, \ldots, N/N_b$), `PDGEMM` involves matrices of dimension $N \times N_b$ and $N_b \times (N - jN_b - N_b)$; `PDLARFB` mainly performs two parallel GEMM operations, with $\{N_b \times (N-jN_b), (N-jN_b) \times (N-jN_b)\}$ and $\{(N-jN_b) \times N_b, N_b \times (N-jN_b)\}$ matrices involved. Another sequential TRMM operation in `PDLARFB` is ignored since it only contributes lower order terms in both arithmetic and communication costs. So the cost for level 3 part is

$$\begin{aligned} T_{\text{level3}} &= \sum_{j=1}^{N/N_b}\left[\frac{2jN_b + 6(N-jN_b)}{p}N_b(N-jN_b)\gamma + \log_2 p\left(3\alpha + \frac{6N-5jN_b}{2\sqrt{p}}\beta\right)\right] \\ &\approx \frac{7N^3}{3p}\gamma + \frac{3N\log_2 p}{N_b}\alpha + \frac{7N^2\log_2 p}{4\sqrt{p}}\beta, \end{aligned}$$

and hence the execution time for Hessenberg reduction (without explicitly forming the orthogonal matrix) is

$$\tilde{T}_{\text{Hess}}(N,p) = T_{\text{level2}} + T_{\text{level3}} \approx \frac{10N^3}{3p}\gamma + N\log_2 p\,\alpha + \frac{5N^2\log_2 p}{2\sqrt{p}}\beta. \quad (3.5.3)$$

Even if the proportion of level 3 operations is improved to 80% as suggested in [121] (this is not implemented in the current PDGEHRD yet), the estimate in (3.5.3) would not change too much since the number of messages in the level 2 part is not reduced.

Since the Householder reflections are stored in a compact form in the lower triangular part of the upper Hessenberg matrix, formulating the orthogonal matrix after Hessenberg reduction is another necessary step. This step is done by the ScaLAPACK routine PDORMHR, which is mainly a series of calls to PDLARFB. Similar to the discussion above, we obtain

$$T_{\text{ORMHR}} \approx \frac{2N^3}{p}\gamma + \frac{3N\log_2 p}{N_b}\alpha + \frac{7N^2}{4\sqrt{p}}\beta.$$

Therefore, the total runtime for the Hessenberg reduction process including formulating the orthogonal matrix is

$$T_{\text{Hess}}(N,p) = \tilde{T}_{\text{Hess}} + T_{\text{ORMHR}} \approx \frac{16N^3}{3p}\gamma + N\log_2 p\,\alpha + \frac{17N^2\log_2 p}{4\sqrt{p}}\beta. \tag{3.5.4}$$

Estimating T_{update}. The cost for updating the off-diagonal blocks with respect to the AED window is simple to analyze since it merely contains three GEMM operations. Since these GEMM operations are not "fat" level 2 operations, we need to use a model different from (3.5.2). According to [150], the execution time for a GEMM operation on a $\sqrt{p}\times\sqrt{p}$ processor grid with $M\times K$ and $K\times N$ matrices involved is

$$T_{\text{GEMM}}(M,N,K,p) \approx \frac{2MNK}{p}\gamma + \left(\frac{K}{N_b} + 2\sqrt{p}\right)\left(2\alpha + \frac{(M+N)N_b}{\sqrt{p}}\beta\right)$$

if $\min\{M,N,K\} = K \gg N_b$. Then we conclude that

$$T_{\text{update}}(N,N_{\text{AED}},p) \approx \frac{2NN_{\text{AED}}^2}{p}\gamma + \frac{N_{\text{AED}}}{N_b}\left(6\alpha + \frac{2NN_b}{\sqrt{p}}\beta\right).$$

Now we are ready to estimate the overall runtime T_{AED} by substituting N with N_{AED} in (3.5.1) and (3.5.4). We can see that T_{redist} is always negligible compared to other components. Reordering contributes with only marginal communication costs also. By merging all these estimates together, we eventually obtain

$$\begin{aligned} T_{\text{AED}}(N,N_{\text{AED}},p) &\approx T_{\text{Schur}}(N_{\text{AED}},p_{\text{AED}}) + T_{\text{reorder}}(N_{\text{AED}},p) + T_{\text{Hess}}(N_{\text{AED}},p) + T_{\text{update}}(N,N_{\text{AED}},p) \\ &\approx \left(\frac{20N_{\text{AED}}}{p_{\text{AED}}} + \frac{4\sqrt{p}\,N_b + 16N_{\text{AED}} + 2N}{p}\right)N_{\text{AED}}^2\gamma \\ &\quad + \frac{N_{\text{AED}}^2}{N_b}\left(\frac{3(\log_2 p_{\text{AED}} + 2)}{\sqrt{p_{\text{AED}}}} + \frac{N_b\log_2 p}{N_{\text{AED}}}\right)\alpha \end{aligned}$$

$$
\begin{aligned}
&+ \frac{N_{\text{AED}}^2}{N_b}\left(\frac{3N_b\log_2 p_{\text{AED}}}{\sqrt{p_{\text{AED}}}} + \frac{8N_{\text{AED}}}{p_{\text{AED}}} + \frac{3N_b}{2\sqrt{p}} + \frac{17N_b\log_2 p}{4\sqrt{p}} + \frac{2NN_b}{N_{\text{AED}}\sqrt{p}}\right)\beta \\
\approx\ & \left[\frac{30C_2^2N}{C_1} + \frac{9N^2N_b}{C_1^2\sqrt{p}} + \frac{9(C_1+6)N^3}{2C_1^3p}\right]\gamma \\
&+ \left(\frac{9C_2N}{C_1N_b}\log_2\frac{3N}{2C_1C_2} + \frac{3N}{2C_1}\log_2 p\right)\alpha \\
&+ \left[\frac{9C_2N}{C_1}\log_2\frac{3N}{2C_1C_2} + \frac{12C_2^2N}{C_1N_b} + \frac{3N^2\left(18+C_1+51\log_2 p\right)}{16C_1^2\sqrt{p}}\right]\beta.
\end{aligned}
$$

When N is extremely large (i.e., C_1, C_2 and N_b are all tiny enough compared to N) and $N/\sqrt{p}$ = constant, we have

$$
\begin{aligned}
T_{\text{AED}} &= \Theta\left(N + \frac{N^2}{\sqrt{p}} + \frac{N^3}{p}\right)\gamma + \Theta\left(N\log N + N\log p\right)\alpha + \Theta\left(N\log N + \frac{N^2}{\sqrt{p}}\log p\right)\beta \\
&= \Theta(N)\,\gamma + \Theta(N\log N)\,\alpha + \Theta(N\log N)\,\beta.
\end{aligned}
$$

Asymptotically AED only has slightly larger message sizes by a $\Theta(\log N)$ factor compared to QR sweeps and is hence not much more expensive. However, in practice we still need to handle AED very carefully since large leading factors in lower order terms have significant impact on the performance when the matrix is not large enough. Similar to the analysis for T_{AED}, the cost for computing shifts can be estimated by

$$
\begin{aligned}
T_{\text{shift}}(N, N_{\text{shift}}, p) &\approx \tilde{T}_{\text{Schur}}(N_{\text{shift}}, p_{\text{shift}}) \\
&\approx \frac{10N_{\text{shift}}^3}{p_{\text{shift}}}\gamma + \frac{3N_{\text{shift}}^2}{\sqrt{p_{\text{shift}}}N_b}(\log_2 p_{\text{shift}} + 2)\,\alpha \\
&\quad + \left(\frac{3N_{\text{shift}}^2\log_2 p_{\text{shift}}}{\sqrt{p_{\text{shift}}}} + \frac{8N_{\text{shift}}^3}{p_{\text{shift}}N_b}\right)\beta \\
&\approx \frac{10C_2^2N}{C_1}\gamma + \frac{6C_2N}{C_1N_b}\log_2\frac{N}{C_1C_2}\,\alpha + \left(\frac{6C_2N}{C_1}\log_2\frac{N}{C_1C_2} + \frac{8C_2^2N}{C_1N_b}\right)\beta.
\end{aligned}
$$

Asymptotically T_{shift} is not so important in the scalability analysis since it can never be larger than T_{AED}.

3.5.3 An overall model

Let us first make a comparison between the pipelined QR algorithm using loosely-coupled shifts and the new parallel multishift QR algorithm using tightly-coupled shifts. Asymptotically, the execution time of the pipelined QR algorithm (3.5.1) is

$$
T_{\text{Schur}}(N,p) = \Theta\left(\frac{N^3}{p}\right)\gamma + \Theta\left(\frac{N^2\log p}{\sqrt{p}\,N_b}\right)\alpha + \Theta\left(\frac{N^3}{pN_b}\right)\beta, \tag{3.5.5}
$$

provided that the average number of shifts required for deflating each eigenvalue is $\Theta(1)$. Under the same assumption, we have shown that the execution time of the new parallel multishift QR algorithm *without* AED is

$$T(N,p) = \Theta\left(\frac{N^3}{p}\right)\gamma + \Theta\left(\frac{N^2\log p}{\sqrt{p}\,N_b^2}\right)\alpha + \Theta\left(\frac{N^3}{pN_b}\right)\beta. \tag{3.5.6}$$

Both solvers have an ideal degree of concurrency. However, tightly-coupled shifts are superior to loosely-coupled shifts, because they require less frequent communication. Compared to (3.5.5), the number of messages in (3.5.6) is reduced by a factor of $\Theta(N_b)$; in return the average message length increases correspondingly. Another important observation is that the parameter γ in (3.5.5) is much larger than that in (3.5.6), because these algorithms have different computational intensity. This already explains why the pipelined QR algorithm is usually much slower than the new parallel multishift QR algorithm for larger matrices, even when neglecting the effects of AED.

Taking AED into account makes the model significantly more complicated. To be able to provide some intuition, we assign concrete values to most parameters. For example, let us set $C_1 = 24$, $C_2 = 384$, and assume $k_{\text{AED}} = 2k_{\text{QR}} = 16k_{\text{shift}} = 64$. Then

$$\begin{aligned}
T_{\text{QR}}(N,N_{\text{shift}},p) &\approx \frac{3N^3}{2p}\gamma + \frac{3N^2}{8\sqrt{p}\,N_b^2}(\log_2 p + 4)\alpha + \frac{3N^3}{8pN_b}\beta,\\
T_{\text{AED}}(N,N_{\text{AED}},p) &\approx \left(184320N + \frac{N^2N_b}{64\sqrt{p}} + \frac{5N^3}{256p}\right)\gamma\\
&\quad + \left[\frac{144N}{N_b}(\log_2 N - 14) + \frac{N\log_2 p}{8}\right]\alpha\\
&\quad + \left[144N\left(\log_2 N - 14 + \frac{512}{N_b}\right) + \frac{\left(51\log_2 p + 42\right)N^2}{3072\sqrt{p}}\right]\beta,\\
T_{\text{shift}}(N,N_{\text{shift}},p) &\approx 61440N\gamma + \frac{96N}{N_b}(\log_2 N - 13)\alpha + 96N\left(\log_2 N - 13 + \frac{512}{N_b}\right)\beta.
\end{aligned} \tag{3.5.7}$$

This yields the following overall estimate for the new parallel QR algorithm with AED:

$$T(N,p) \approx \left(\frac{48N^3}{p} + 1.2\times 10^7 N\right)\gamma + \frac{12N^2\log_2 p}{\sqrt{p}\,N_b^2}\alpha + \left(\frac{12N^3}{pN_b} + \frac{17N^2\log_2 p}{16\sqrt{p}}\right)\beta, \tag{3.5.8}$$

$$= \Theta\left(\frac{N^3}{p}\right)\gamma + \Theta\left(\frac{N^2\log p}{\sqrt{p}\,N_b^2}\right)\alpha + \Theta\left(\frac{N^3}{pN_b}\right)\beta, \tag{3.5.9}$$

where most small-order terms are neglected. It turns out that both QR sweeps and AED have significant serial runtime when N is not very large. However, QR sweeps usually dominate the communication cost. As a consequence, the models (3.5.5), (3.5.6), and (3.5.9) nearly have the same asymptotic behavior. AED is asymptotically not more expensive compared to QR sweeps, and hence it does not represent a computational bottleneck for larger matrices. Combined with the convergence acceleration often observed when using AED (and not fully attributed in

the model above), this contributes to the superior performance of the new parallel multishift QR algorithm.

3.6 Software and implementation issues

3.6.1 Calling sequence

The calling sequence of the newly developed routine PDHSEQR is nearly identical with the LAPACK routine DHSEQR, see Figure 3.7. Apart from the need of a descriptor for each globally distributed matrix and the leading dimension for each local matrix, the only difference is that PDHSEQR requires an extra integer workspace. The calling sequence of the ScaLAPACK routine PDLAHQR is also similar, hopefully allowing to easily switch from PDLAHQR and DHSEQR in existing software making use of ScaLAPACK. In practice, it is advisable to call PDHSEQR twice—one call for the workspace query (by setting LWORK = −1) and another call for actually doing the computation. This follows the convention of many LAPACK/ScaLAPACK routines that make use of workspace. We refer to Appendix C for details regarding the calling sequence.

```
      SUBROUTINE PDHSEQR( JOB, COMPZ, N, ILO, IHI, H, DESCH, WR, WI, Z,
     $                    DESCZ, WORK, LWORK, IWORK, LIWORK, INFO )
*
*     .. Scalar Arguments ..
      INTEGER            IHI, ILO, INFO, LWORK, LIWORK, N
      CHARACTER          COMPZ, JOB
*     ..
*     .. Array Arguments ..
      INTEGER            DESCH( * ) , DESCZ( * ), IWORK( * )
      DOUBLE PRECISION   H( * ), WI( N ), WORK( * ), WR( N ), Z( * )
```

```
      SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
     $                   LDZ, WORK, LWORK, INFO )
```

```
      SUBROUTINE PDLAHQR( WANTT, WANTZ, N, ILO, IHI, A, DESCA, WR, WI,
     $                    ILOZ, IHIZ, Z, DESCZ, WORK, LWORK, IWORK,
     $                    ILWORK, INFO )
```

Figure 3.7 – Calling sequences of the newly developed routine PDHSEQR, the corresponding LAPACK routine DHSEQR, and the ScaLAPACK routine PDLAHQR.

3.6.2 Tuning parameters

In the new software for the parallel multishift QR algorithm, tunable parameters are defined in the routine PILAENVX. They are available via the function call

```
PILAENVX(ICTXT, ISPEC, ...)
```

Table 3.2 – List of tunable parameters.

ISPEC	Name	Description	Recommended value
12	$N_{\min}$	Crossover point between PDLAQR0 and PDLAQR1	$220\min(p_r, p_c)$
13	N_{AED}	Size of the AED window	See Table 3.1
14	NIBBLE	Threshold for skipping a multishift QR sweep	See Section 3.4
15	N_{shift}	Number of simultaneous shifts	See Table 3.1
16	KACC22	Specification of how to update off-diagonal blocks in the multishift QR sweep	Use GEMM/TRMM
17	NUMWIN	Maximum number of concurrent computational windows (for both QR sweep and eigenvalue reordering)	$\min(p_r, p_c, \lceil N/N_b \rceil)$
18	WINEIG	Number of eigenvalues in each window (for eigenvalue reordering)	$\min(N_b/2, 40)$
19	WINSIZE	Computational window size (for both bulge-chasing and eigenvalue reordering)	$\min(N_b, 80)$
20	MMULT	Minimal percentage of flops for performing GEMM instead of pipelined Householder reflections when updating the off-diagonal blocks in the eigenvalue reordering routine	50
21	NCB	Width of block column slabs for rowwise update of Householder reflections in factorized form	$\min(N_b, 32)$
22	WNEICR	Maximum number of eigenvalues to move over a block border in the eigenvalue reordering routine	Identical to WINEIG
23	$p_{\min}$	Size of processor grid involving AED	See Section 3.3.3

with $12 \leq$ ISPEC ≤ 23. A complete list of these parameters is provided in Table 3.2. Some of them require fine tuning to attain nearly optimal performance across different architectures.

Although a reasonable choice of N_b, the data layout block size, is important, we have observed the performance to be not overly sensitive to this choice. On the one hand, N_b should be large enough so that the local computations can achieve level 3 performance. On the other hand, it is advisable to avoid N_b being too large. A large value of N_b harms load balance and increases the overhead in the start-up and ending stages of the bulge chasing process, especially when computing the Schur decomposition of the AED window. In our performance model, we always assume $N_b \ll N/\sqrt{p}$ to avoid such kind of overhead. For many architectures, $N_b \in [32, 128]$ will offer a good choice.

We expect that most of the recommended values in Table 3.2 yield reasonable performance on existing architectures. However, the parameters $N_{\min}$, $p_{\min}$, and NIBBLE require some extra care, as the performance of AED crucially relies on them. The values of these parameters need to be determined by performing a bunch of test runs. To determine $N_{\min}$ and $p_{\min}$,

it is advisable to use the typical sizes of the AED windows (see Table 3.1) and run tests on different processor grids. Then the optimal values for both $N_{\min}$ and $p_{\min}$ can be chosen via examining the number of columns of H owned by each processor. `NIBBLE` should be tuned lastly, once all other parameters are fixed. Tuning `NIBBLE` is time-consuming but highly recommended, especially on older architectures with relatively slow communication. As discussed in Section 3.4, $\texttt{NIBBLE} = a \cdot N^b p^c$ is a reasonably good model that takes into account both N and p. It is not unlikely that this model may need to be adjusted for very large-scale computations.

3.7 Computational experiments

We have performed a large set of computational experiments on *Akka*[1] and *Abisko*[2] hosted by the High Performance Computing Center North (HPC2N), and on *Bellatrix*[3] hosted by École Polytechnique Fédérale de Lausanne. In this section, we present a subset from these computational experiments to confirm and demonstrate the improvements we have made in the parallel QR algorithm. The computational environments are summarized in Table 3.3.

Table 3.3 – Computational environments.

Akka	64-bit Intel Xeon (Harpertown) Linux cluster 672 dual socket nodes with L5420 quad-core 2.5GHz processors and 16GB RAM per node Cisco Infiniband and Gigabit Ethernet, 10Gbps bandwidth PathScale compiler version 4.0.13 OpenMPI 1.4.4, LAPACK 3.4.0, GotoBLAS2 1.13
Abisko	64-bit AMD Opteron (Interlagos) Linux cluster 322 nodes with four Opteron 6238 12-core 2.6GHz processors and 128GB RAM per node Mellanox 4X QSFP Infiniband connectivity, 40Gbps bandwidth PathScale compiler version 4.0.13 OpenMPI 1.6.4, LAPACK 3.4.0, OpenBLAS 0.1 alpha 2.4
Bellatrix	64-bit Intel Xeon (Sandy Bridge) Linux cluster 424 dual socket nodes with E5-2660 octa-core 2.2GHz processors and 32GB RAM per node Qlogic Infiniband QDR 2:1 connectivity, 40Gbps bandwidth Intel compiler version 13.0.1 Intel MPI version 4.1.0, Intel Math Kernel Library version 11.0

We compare the following implementations:

S-v180	Pipelined QR algorithm in ScaLAPACK version 1.8.0.
SISC	Previous implementation of parallel multishift QR algorithm with AED, as described in [61].
NEW	New implementation of parallel multishift QR algorithm with AED, as described in this chapter.

1. http://www.hpc2n.umu.se/resources/akka/
2. http://www.hpc2n.umu.se/resources/abisko/
3. http://hpc.epfl.ch/clusters/bellatrix/

Table 3.4 – Execution time (in seconds) on *Akka* for `fullrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	834	178	115	10730	939	628						
2×2	317	87	56	2780	533	292						
4×4	136	50	35	764	205	170	6671	1220	710			
6×6	112	50	43	576	142	116	3508	754	446	∞	3163	2200
8×8	100	45	37	464	127	104	2536	506	339	∞	2979	1470
10×10	97	50	36	417	159	119	2142	457	320	∞	2401	1321

Table 3.5 – Execution time (in seconds) on *Akka* for `hessrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	685	317	14	6981	2050	78						
2×2	322	200	8	2464	1904	27						
4×4	163	112	36	1066	679	71	8653	2439	65			
6×6	137	84	31	768	412	113	4475	1254	71	∞	373	252
8×8	121	68	25	634	321	107	3613	719	71	∞	919	228
10×10	131	83	23	559	313	111	3549	667	76	∞	943	267

The implementation NEW improves upon the implementation SISC in terms of robustness and performance, in particular because of the proposed modifications to the AED step.

The data layout block size $N_b = 50$ is used for all experiments. No multithreaded features (such as OpenMP or threaded BLAS) are used. Therefore the number of processors ($p = p_r \cdot p_c$) means the number of cores involved in the computation.

3.7.1 Random matrices

First, we consider two types of random matrices—`fullrand` and `hessrand` [61].

Matrices of the type `fullrand` are dense square matrices with all entries randomly generated from a uniform distribution in $[0,1]$. We call the ScaLAPACK routine `PDGEHRD` to reduce them to upper Hessenberg form before applying the QR algorithm. Only the time for the QR algorithm (i.e., reducing the upper Hessenberg matrix to Schur form) is measured. These matrices usually have well-conditioned eigenvalues and exhibit "regular" convergence behavior.

Matrices of the type `hessrand` are upper Hessenberg matrices whose nonzero entries are randomly generated from a uniform distribution in $[0,1]$. The eigenvalues of these matrices are extremely ill-conditioned for larger N, affecting the convergence behavior of the QR sweeps [61]. On the other hand, AED often deflates a high fraction of eigenvalues in the AED window for such matrices. These properties sometimes cause erratic convergence rates.

Tables 3.4 and 3.5 show the parallel execution times of the three solvers on *Akka*. Both the real Schur form T and the orthogonal transformation matrix Z are calculated. We limit the total execution time (including the Hessenberg reduction) by 10 hours for each individual

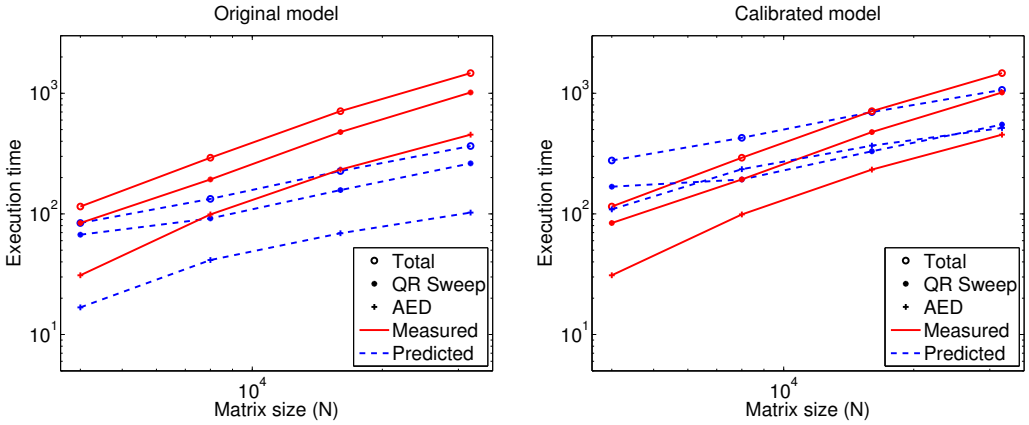


Figure 3.8 – Comparison between the measured execution times and the predicted times using (3.5.7) (`fullrand`, $N/\sqrt{p} = 4000$). The original model (left) uses theoretical values of (α, β, γ) according to the hardware information, while the calibrated one (right) adjusts γ according to different computational intensities (level 1, 2, and 3).

problem, to avoid excessive use of the computational resources. An entry ∞ corresponds to an execution time larger than 10 hours. These tables reveal that our new version of PDHSEQR (i.e., NEW) always improves the performance compared to the SISC version. On average, the improvement is 31% for matrices of type `fullrand` and 14 times for matrices of type `hessrand`. Not surprisingly, the improvements compared to PDLAHQR in ScaLAPACK version 1.8.0 are even more significant.

The convergence rates for `fullrand` are sufficiently regular, so that we can analyze the scalability of the parallel multishift QR algorithm. If we fix the memory load per core to $N/\sqrt{p} = 4000$, the execution times in Table 3.4 satisfy

$$2T(N,p) \leq T(2N,4p) < 4T(N,p),$$

indicating that the parallel multishift QR algorithm scales reasonably well but not perfectly. To verify the performance models we have derived in Section 3.5, we use (3.5.7) together with the measured values of k_{AED}, k_{QR}, and k_{shift} to predict the execution time. Since $k_{\text{shift}} = 0$ is observed for all these examples, there are only two components in the total execution time (i.e., $T = k_{\text{QR}} T_{\text{QR}} + k_{\text{AED}} T_{\text{AED}}$). Figure 3.8(a) illustrates that the predicted execution times underestimate the measured ones (especially, for AED), mainly due to too optimistic choices of the parameters (α, β, γ). If we assume that the program executes at 40% and 7.5% of the peak core performance for level 3 and level 1–2 BLAS operations, respectively, the calibrated model fits the actual execution time quite well for large matrices (see Figure 3.8(b)). Since the model(s) are asymptotic, the results are very satisfactory.

For `hessrand`, it is observed that most eigenvalues are deflated with very few (or even no) QR sweeps. Considering that the main difference of PDHSEQR between versions NEW and

SISC is in the AED process, it is not surprising to see the great convergence acceleration for `hessrand`, where AED dominates the calculation. In Table 3.5, sometimes the execution time for the new parallel multishift QR algorithm does not change too much when increasing the number of processors. This is mainly because the calculation of the Schur decomposition of the AED window, which is the most expensive part of the algorithm, is performed by a constant number of processors ($p_{\min} \cdot p_{\min} \leq p$) after data redistribution.

In Tables 3.6–3.9 we list the execution times received from *Abisko* and *Bellatrix*. The observations are similar to those obtained from *Akka*. Therefore, in the rest of this section we only present experiments on *Akka* for economical consideration.

Table 3.6 – Execution time (in seconds) on *Abisko* for `fullrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	764	139	97	7373	694	471						
2×2	302	77	49	2479	417	240						
4×4	91	40	31	781	156	132	5507	1040	548			
6×6	76	36	28	541	101	91	2799	591	374	∞	2641	1706
8×8	52	34	29	276	88	98	1881	383	294	∞	2506	1245
10×10	52	30	18	234	99	92	1455	317	257	∞	1909	1118

Table 3.7 – Execution time (in seconds) on *Abisko* for `hessrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	611	307	12	5021	2064	63						
2×2	302	202	7	1966	1458	29						
4×4	110	77	18	881	516	48	6671	1603	53			
6×6	96	61	21	578	339	70	4006	1034	52	∞	231	176
8×8	84	53	18	423	249	98	2822	605	53	∞	737	166
10×10	73	58	17	360	214	79	2456	553	56	∞	670	166

Table 3.8 – Execution time (in seconds) on *Bellatrix* for `fullrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	637	73	50	5377	441	252						
2×2	192	24	21	1594	137	91						
4×4	68	16	13	498	79	63	4505	552	271			
6×6	47	12	11	294	44	39	1886	247	165	∞	1267	901
8×8	36	16	12	204	42	37	1347	184	129	∞	1362	714
10×10	37	14	9	181	39	40	961	140	110	∞	726	525

3.7.2 $100,000 \times 100,000$ matrices

The modifications proposed and presented result into dramatic improvements for settings with very large matrices and many processors. To demonstrate this, we present the obtained execution times for $100,000 \times 100,000$ matrices in Table 3.10. Although the QR algorithm does not scale as well as Hessenberg reduction when fixing the problem size and increasing

Table 3.9 – Execution time (in seconds) on *Bellatrix* for `hessrand` matrices.

$p =$	$N = 4000$			$N = 8000$			$N = 16000$			$N = 32000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	510	174	6	4353	1102	26						
2×2	205	66	4	1590	530	11						
4×4	87	42	7	657	259	19	5416	808	22			
6×6	57	23	9	428	134	37	3054	380	22	∞	102	77
8×8	54	37	9	340	125	32	2227	365	22	∞	342	72
10×10	46	22	8	280	92	27	1846	214	24	∞	214	115

the number of processors, the execution times of these two reduction steps are still on the same order of magnitude. With the help of the dynamic `NIBBLE` strategy, the fraction of the execution time spent on AED for `fullrand` matrices is under control. In contrast to the earlier implementation (the SISC version), AED is not a bottleneck of the whole QR algorithm now. As reported in [61], it took 7 hours for the SISC version of `PDHSEQR` to solve the $100,000 \times 100,000$ `fullrand` problem with 32×32 processors; 80% execution time of the QR algorithm was spent on AED. Our new version of `PDHSEQR` is able to solve the same problem in roughly 1.85 hours, which is about four times faster. The time fraction spent on AED is reduced to 39%.

Table 3.10 – Execution time (in seconds) on *Akka* of the new parallel multishift QR algorithm (NEW) for $100,000 \times 100,000$ matrices.

	$p = 16 \times 16$		$p = 24 \times 24$		$p = 32 \times 32$		$p = 40 \times 40$	
	`fullrand`	`hessrand`	`fullrand`	`hessrand`	`fullrand`	`hessrand`	`fullrand`	`hessrand`
Balancing	876	–	881	–	886	–	912	–
Hess. reduction	10084	–	6441	–	3868	–	2751	–
QR algorithm	13797	922	8055	1268	6646	5799	8631	1091
k_{AED}	35	19	31	19	27	18	23	18
k_{QR}	5	0	6	0	13	0	12	0
#(shifts)/N	0.20	0	0.23	0	0.35	0	0.49	0
AED% in the QR alg.	48%	100%	43%	100%	39%	100%	54%	100%

3.7.3 Benchmark examples

Besides random matrices, we also report performance results for some commonly used benchmark matrices. For comparison, we have tested the same matrices as in [61], see Table 3.11. The execution times for the three solvers are listed in Tables 3.12–3.19. The conclusions are similar to those we have made for random matrices: The earlier version of `PDHSEQR` outperforms the ScaLAPACK 1.8.0 routine `PDLAHQR` by a large extent; the new `PDHSEQR` is usually even faster, especially for BBMSN and GRCAR.

In [61], it was observed that the accuracy for AF23560 is not fully satisfactory; the relative residuals $R_r = \|Q^T AQ - T\|_F / \|A\|_F$ were large for both `PDLAHQR` and `PDHSEQR`. It turns out that these large residuals are caused by an anomaly in `PDLAHQR`, which has been fixed by avoiding the use of `PDLACONSB`, see Section 3.3.2. As a result, the new `PDHSEQR` always produce $R_r \in \left[10^{-15}, 10^{-13}\right]$ for all test matrices.

Table 3.11 – Benchmark matrices.

ID	Name	Dimension (N)	Type/Structure
1	BBMSN [31]	N	$S_N = \begin{bmatrix} N & N-1 & N-2 & \cdots & 2 & 1 \\ 10^{-3} & 1 & 0 & & 0 & 0 \\ & 10^{-3} & 2 & & 0 & 0 \\ & & 10^{-3} & & 0 & 0 \\ & & & \ddots & N-2 & 0 \\ & & & & 10^{-3} & N-1 \end{bmatrix}$
2	AF23560 [9]	23560	Computational fluid dynamics
3	CRY10000 [9]	10000	Material science
4	OLM5000 [9]	5000	Computational fluid dynamics
5	DW8192 [9]	8192	Electrical engineering
6	MATRAN [9]	N	Sparse random matrix
7	MATPDE [9]	N	Partial differential equations
8	GRCAR [9]	N	$G_N = \begin{bmatrix} 1 & 1 & 1 & 1 & & & & \\ -1 & 1 & 1 & 1 & 1 & & & \\ & -1 & 1 & 1 & 1 & 1 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -1 & 1 & 1 & 1 & 1 \\ & & & & -1 & 1 & 1 & 1 \\ & & & & & -1 & 1 & 1 \\ & & & & & & -1 & 1 \end{bmatrix}$

Table 3.12 – Execution time (in seconds) for BBMSN.

N	$p = p_r \times p_c$	S-v180	SISC	NEW
5000	1×1	523	6	3
10000	2×2	1401	30	9
15000	3×3	1489	62	13

Table 3.13 – Execution time (in seconds) for AF23560.

$p = p_r \times p_c$	S-v180	SISC	NEW
4×4	15486	2651	1375
6×6	9088	1279	826
8×8	6808	793	563
10×10	5694	662	475
12×12	5422	578	404

3.8 Summary

In this chapter we have presented a new parallel implementation of the multishift QR algorithm with aggressive early deflation. The new routine PDHSEQR combines a number of techniques to improve serial performance and reduce communication. Our computational experiments provide compelling evidence that PDHSEQR significantly outperforms not only the

Table 3.14 – Execution time (in seconds) for CRY10000.

$p = p_r \times p_c$	S-v180	SISC	NEW
1×1	6394	1331	1251
2×2	2123	580	495
4×4	979	236	209
6×6	731	161	132
8×8	545	128	95
10×10	496	144	90

Table 3.15 – Execution time (in seconds) for OLM5000.

$p = p_r \times p_c$	S-v180	SISC	NEW
1×1	426	206	189
2×2	167	98	108
4×4	76	54	53
6×6	58	52	42
8×8	46	49	29
10×10	48	51	47

Table 3.16 – Execution time (in seconds) for DW8192.

$p = p_r \times p_c$	S-v180	SISC	NEW
1×1	10307	1329	1297
2×2	1187	572	524
4×4	635	225	236
6×6	357	152	138
8×8	302	129	104
10×10	275	121	93

Table 3.17 – Execution time (in seconds) for MATRAN.

$p =$	$N = 5000$			$N = 10000$			$N = 15000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	1617	332	218	∞	1756	1137			
2×2	579	152	122	4495	931	471			
4×4	247	74	60	1555	321	268	5122	937	575
6×6	178	64	57	1035	207	170	3046	535	382
8×8	147	58	50	746	153	157	2166	390	315
10×10	149	59	43	615	169	140	1669	362	264

original ScaLAPACK routine `PDLAHQR` but also an earlier version of `PDHSEQR` presented in [61]. In particular, our new implementation removes a bottleneck in the aggressive early deflation strategy by reducing communication and tuning algorithmic parameters. As a result, our new version is both faster and more robust. An intermediate version of the software presented in this chapter is available in ScaLAPACK version 2.0.

Table 3.18 – Execution time (in seconds) for MATPDE.

$p =$	$N = 10000$			$N = 14400$			$N = 19600$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	12429	2600	1699						
2×2	3531	1081	966						
4×4	1565	415	361	4446	1207	1027	9915	2844	2406
6×6	1118	256	225	3069	654	573	6130	1426	1284
8×8	871	189	156	2259	449	384	4615	912	802
10×10	789	189	137	1955	431	313	4046	743	628
12×12	719	194	126	1736	367	260	3483	648	504

Table 3.19 – Execution time (in seconds) for GRCAR.

$p =$	$N = 6000$			$N = 12000$			$N = 18000$		
$p_r \times p_c$	S-v180	SISC	NEW	S-v180	SISC	NEW	S-v180	SISC	NEW
1×1	2738	1340	69						
2×2	850	645	40	8199	2734	132			
4×4	363	258	226	2499	1336	114	8171	4037	182
6×6	244	190	173	1471	849	112	4385	2172	187
8×8	217	150	145	1107	515	142	3342	1345	175
10×10	207	161	126	923	538	338	2675	1104	276

We believe to have come to a point, where it will be difficult to attain further dramatic performance improvements for parallel nonsymmetric eigensolvers on distributed memory architectures, without leaving the classical framework of QR algorithms. Considering the fact that the execution times spent on Hessenberg reduction and on QR iterations are now nearly on the same level, any further improvement of the iterative part will only have a limited impact on the total execution time. The situation is quite different when shared memory many-core processors with accelerators, such as GPUs, are considered. Although efficient implementations of the Hessenberg reduction on such architectures have recently been proposed [87, 91, 146], the iterative part remains to be done. Another future challenge is to combine the message passing paradigm used in this new implementation of the multishift QR algorithm and dynamic and static scheduling on many-core nodes using multithreading.

Part II—Matrix Exponentials

4 The Matrix Exponential

From this chapter on, we discuss another topic in matrix computations—computation of the matrix exponential. The matrix exponential is one of the most important matrix functions. It naturally arises in many applications, especially those related to the solution of dynamical systems. Many methods for computing the matrix exponential have been proposed, see, e.g., [109]. In this chapter, we briefly recall some properties of the matrix exponential and several popular algorithms. This material provides the foundations for our developments in the subsequent two chapters. We refer to [73, 109] for more detailed discussions regarding the matrix exponential.

4.1 Properties of the matrix exponential

In the following, we provide the definition and several basic properties of the matrix exponential. Since the exponential is an analytic function, we can make use of the following definition of matrix functions.

Definition 4.1.1. *Let A be a complex square matrix. Let $\Omega \subset \mathbb{C}$ be a domain which contains $\Lambda(A)$ and has piecewise smooth boundary. Then for a function $F\colon \Omega \to \mathbb{C}$ being analytic, $F(A)$ is defined as*

$$F(A) = \frac{1}{2\pi \mathrm{i}} \oint_{\partial\Omega} F(z)(zI - A)^{-1}\,\mathrm{d}z. \tag{4.1.1}$$

By the Cauchy integral theorem [2], it can be verified that $F(A)$ is independent of the concrete choice of Ω as long as $\Lambda(A) \subset \Omega$ and hence is well-defined by A and F. It is also straightforward to show by definition the two basic properties

$$AF(A) = F(A)A \tag{4.1.2}$$

and

$$F(P^{-1}AP) = P^{-1}F(A)P, \qquad (\det(P) \neq 0). \tag{4.1.3}$$

According to (4.1.1), the matrix exponential is defined as

$$\exp(A) = \frac{1}{2\pi \mathrm{i}} \oint_{\partial\Omega} \exp(z)(zI - A)^{-1}\,\mathrm{d}z. \tag{4.1.4}$$

The following theorem summarizes several properties of the matrix exponential. These properties are extensively used in this thesis.

Theorem 4.1.2. *For $A \in \mathbb{C}^{N\times N}$, the matrix exponential defined by* (4.1.4) *satisfies the following properties.*

(a) $\exp(nA) = [\exp(A)]^n$ *for* $n \in \mathbb{Z}$.

(b) $\exp(A) = \exp(\alpha)\exp(A - \alpha I)$ *for* $\alpha \in \mathbb{C}$.

(c) $\exp(A)$ *has the series expansion*

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \tag{4.1.5}$$

(d) The unique solution of the initial value problem

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = AX(t), \qquad X(0) = I, \tag{4.1.6}$$

is given by $X(t) = \exp(tA)$.

(e) $\exp(A)$ *is the limit*

$$\exp(A) = \lim_{n\to\infty} \left(I + \frac{A}{n}\right)^n. \tag{4.1.7}$$

(f) For any $\Delta A \in \mathbb{C}^{N\times N}$, we have

$$\exp[t(A+\Delta A)] - \exp(tA) = \int_0^t \exp[(t-s)A]\Delta A \exp[s(A+\Delta A)]\,\mathrm{d}s. \tag{4.1.8}$$

Proof. See, e.g., [73, Chapter 1], [78, Chapter 6]. □

It is worth pointing out that a condition number of $\exp(A)$ can be derived from (4.1.8). Since

$$\begin{aligned}
&\left\|\exp(A+\Delta A) - \exp(A)\right\| \\
&= \left\| \int_0^1 \exp[(1-s)A]\Delta A \exp[s(A+\Delta A)]\,\mathrm{d}s \right\| \\
&= \left\| \int_0^1 \exp[(1-s)A]\Delta A \left[\exp(sA) + \int_0^1 \exp[(1-t)A]\Delta A \exp[t(A+\Delta A)]\,\mathrm{d}t \right] \mathrm{d}s \right\| \\
&= \left\| \int_0^1 \exp[(1-s)A]\Delta A \exp(sA)\,\mathrm{d}s \right\| + \mathcal{O}\left(\|\Delta A\|^2\right)
\end{aligned}$$

as $\|\Delta A\| \to 0$, the condition number of $\exp(A)$ with respect to the norm $\|\cdot\|$ is thus defined as

$$\kappa_{\exp}(A) = \max_{X \neq 0} \frac{\|A\|}{\|X\| \, \|\exp(A)\|} \left\| \int_0^1 \exp[(1-s)A] X \exp(sA) \, \mathrm{d}s \right\|. \tag{4.1.9}$$

For detailed perturbation analyses, we refer to [73, 84, 151].

4.2 Algorithms for the matrix exponential

The properties listed in Theorem 4.1.2 provide many candidates for computing the matrix exponential. Especially, properties (c), (d), (e) in Theorem 4.1.2 can be considered as alternative definitions to $\exp(A)$. Because of these different but equivalent representations of $\exp(A)$, many algorithms have been proposed in the past decades, see [109] for an excellent survey. Unfortunately, none of these existing methods can always satisfactorily compute $\exp(A)$. Hence special care needs to be taken when solving a particular problem. In the following we summarize several algorithms which are closely related to the next two chapters.

Eigenvalue-based methods. The property (4.1.3) leads to a class of "direct" methods for computing matrix functions. The simplest case is when A is Hermitian. In this case $F(A) = QF(\Lambda)Q^*$ can immediately be obtained once the spectral decomposition $A = Q\Lambda Q^*$ is computed, where Q is unitary and Λ is diagonal. For non-Hermitian matrices, we demonstrate a simplified version the Schur-Parlett method as an example. The first step is to compute the Schur decomposition $A = QTQ^*$, where Q is unitary and T is upper triangular. We assume that all eigenvalues of A are distinct for the sake of simplicity. Then by (4.1.2), $X = F(T)$ is a solution of the matrix equation $TX - XT = 0$. Since $F(T)$ is also upper triangular with diagonal entries $x_{ii} = F(t_{ii})$, the off-diagonal entries of $F(T)$ are computed via the *Parlett recurrence* [116]

$$x_{ij} = t_{ij} \frac{x_{ii} - x_{jj}}{t_{ii} - t_{jj}} + \sum_{k=i+1}^{j-1} \frac{x_{ik} t_{kj} - t_{ik} x_{kj}}{t_{ii} - t_{jj}}$$

by diagonals—from the first super-diagonal towards the top-right corner. Finally, the solution is given by $F(A) = QF(T)Q^*$.

For a matrix A with a certain special structure, computing the Schur form of A may destroy the structure and provoke unnecessary errors. There is a more serious drawback of this method in general—close eigenvalues of A often cause numerical instability in the Parlett recurrence. In practice, the diagonal of T is reordered into groups of clustered eigenvalues, and the method is usually applied in a blocked manner to gain both stability and performance [73, 85, 116].

The Schur-Parlett method does not rely too much on the properties of $F(z)$. On the one hand, this makes it suitable for computing general analytic matrix functions. With the help of the new parallel implementation of the multishift QR algorithm presented in Chapter 3, it is possible to apply this method to large-scale matrices also. Studies in this direction are planned

as our future work. On the other hand, this generality of the Schur-Parlett method may render it less competitive compared to some other methods stated below, which make use of special properties of $\exp(z)$. Therefore, the Schur-Parlett method is usually not the method of choice for computing the matrix exponential in practice.

Scaling and squaring method. The scaling and squaring method uses the property $\exp(A) = \exp\left(A/2^k\right)^{2^k}$ to evaluate $\exp(A)$, while $\exp\left(A/2^k\right)$ is approximated by a rational function of A. Compared to $\exp(A)$, $\exp\left(A/2^k\right)$ is generally much easier to approximate using polynomials or rational functions since the spectrum of $A/2^k$ is closer to the origin. Hence the scaling and squaring method is typically applied as a preprocessing step in other methods and often significantly reduces the cost of approximating the matrix exponential. As an extreme case, a first order approximation $I + A/2^k$ to $\exp\left(A/2^k\right)$ is already sufficient when k is large enough, according to (4.1.7).

The biggest drawback of the scaling and squaring method is that the rounding error (see Appendix B) can sometimes grow quickly in the squaring phase [73, 109]:

$$\left\| \mathrm{fl}(B^{2^k}) - B^{2^k} \right\| \leq (2^k - 1)N\mathbf{u} \cdot \|B\| \prod_{j=0}^{k-1} \left\| B^{2^j} \right\| + \mathscr{O}(\mathbf{u}^2). \tag{4.2.1}$$

Therefore, in general the scale factor 2^k cannot be chosen too large to avoid numerical instability. We will discuss in Chapter 5 a special case for which the rounding error bound is less pessimistic and hence a moderately large scale factor is admissible.

Truncated Taylor series method. Consider the mth order approximation

$$T_m(x) = \sum_{k=0}^{m} \frac{x^k}{k!}$$

in the Maclaurin series expansion of $\exp(x)$. Based on property (c) in Theorem 4.1.2, it is natural to expect that $T_m(A)$ approximates $\exp(A)$ well for a sufficiently large truncation order m. In practice, shifting and scaling on A are also applied before truncating the Taylor series, i.e.,

$$\exp(A) \approx \exp(\alpha) T_m\left(\frac{A - \alpha I}{2^k}\right)^{2^k}.$$

The shift α should approximately minimize $\rho(A - \alpha I)$ and the scale factor is often chosen such that $\|B\| < 1$ (or $\rho(B) < 1$), where $B = (A - \alpha I)/2^k$. According to the error estimate in [103], the truncation order is chosen so that

$$\frac{\|B\|^{m+1}}{(m+1)!}\left(1 - \frac{\|B\|}{m+2}\right)^{-1} \leq \tau, \tag{4.2.2}$$

where τ is the desired (absolute) accuracy prescribed by the user. This method has the advantage that only matrix multiplications are involved and it is therefore easy to implement. It is also worth pointing out that for a nonnegative matrix B, the evaluation of $T_m(B)$ can be performed without cancellation and hence achieves high componentwise accuracy, see Chapter 5 for details.

Padé approximation method. Similar to the truncated Taylor series method, it is also natural to use the Padé approximant

$$R_{pq}(A) = N_{pq}(A)D_{pq}(A)^{-1}$$

to approximate $\exp(A)$, where

$$N_{pq}(x) = \sum_{j=0}^{p} \frac{(p+q-j)!\,p!}{(p+q)!\,j!\,(p-j)!}x^j, \qquad D_{pq}(x) = \sum_{j=0}^{q} \frac{(p+q-j)!\,q!}{(p+q)!\,j!\,(q-j)!}(-x)^j.$$

Here $R_{pq}(x)$ is the unique rational function of the form $u(x)/v(x)$ satisfying $u \in \mathbb{C}_p[x]$, $v \in \mathbb{C}_q[x]$, and

$$\exp(x) = \frac{u(x)}{v(x)} + \mathcal{O}(x^{p+q+1})$$

as $x \to 0$, see, e.g., [13, Chapter 1]. Usually the diagonal Padé approximant (i.e., with $p = q$) is chosen so that the denominator $D_{pp}(A)$, which then equals $N_{pp}(-A)$, can be computed simultaneously when evaluating $N_{pp}(A)$. Certainly, preprocessing steps such as shifting and scaling should also be applied, i.e.,

$$\exp(A) \approx \exp(\alpha)R_{pp}\left(\frac{A-\alpha I}{2^k}\right)^{2^k}.$$

It can be shown [72, 109] that a moderately small pair of (p,k) is already sufficient for bounding the backward error below the IEEE double precision machine epsilon.

Computing $\exp(A)$ using Padé approximation built on scaling and squaring is the most popular method in practice and has been extensively studied. The `expm` function in MATLAB (since version 7.2(R2006a)) implements an advanced version of this method, see [72, 74] for details. In general, the diagonal Padé approximation $R_{pp}(A)$ produces far smaller truncation error than the truncated Taylor series $T_m(A)$ when $p = m$, i.e., the same degree of polynomials are involved [109]. However, in Chapter 5 we discuss a problem where the truncated Taylor series method is preferred.

ODE methods. Since $\exp(A)$ is the solution of the initial value problem (4.1.6) at $t = 1$, every numerical method for solving ordinary differential equations can be adopted to compute $\exp(A)$. We restrict our discussion to single-step methods with fixed step size. For example,

the Euler method with step size n^{-1} reads

$$X^{(0)} = I, \qquad , X^{(j)} = X^{(j-1)} + \frac{1}{n} A X^{(j-1)}, (j = 1, 2, \ldots, n)$$

and eventually yields $\exp(A) \approx X^{(n)} = (I + A/n)^n$. Similarly the backward Euler method yields $\exp(A) \approx (I - A/n)^{-n}$. The convergence of both Euler methods is evident from (4.1.7). To avoid n being too large, higher order methods such as Runge-Kutta methods can be applied [66]. In fact, it can be verified [162] that an m-stage explicit Runge-Kutta method of order m with step size n^{-1} for solving (4.1.6) mathematically produces

$$\exp(A) \approx X^{(n)} = T_m(A/n)^n,$$

due to the fact that $T_m(x)$ is the unique polynomial up to degree m which approximates $\exp(x)$ to order m.

The direct application of single-step ODE methods is generally inefficient [109] for evaluating $\exp(A)$. However, as we are only interested in $\exp(A)$ rather than $X(t) = \exp(tA)$ on the whole interval $[0, 1]$, there is no need to evaluate all $X^{(j)}$'s. If we choose $n = 2^k$ and only compute $X^{(2^j)}$ for $j = 0, 1, \ldots, k$, the scaling and squaring method can be regarded as a special case of these ODE methods [106, 109, 162]. We will see in Chapter 5 that such variants of ODE methods can actually be very efficient.

5 Aggressively Truncated Taylor Series Method

In this chapter, we discuss how to compute $\exp(A)$ to high componentwise relative accuracy when $A \in \mathbb{R}^{N\times N}$ is essentially nonnegative. By *essentially nonnegative*, we mean that all off-diagonal entries of A are nonnegative [22, 73]. Such matrices are also known as *Metzler matrices* [64, 111]. Let

$$s(A) = \min_i a_{ii}, \qquad \widehat{A} = A - s(A)I. \tag{5.0.1}$$

Then by Theorem 4.1.2, we have

$$\exp(A) = \exp[s(A)]\exp(\widehat{A}) = \exp[s(A)]\sum_{k=0}^{\infty}\frac{\widehat{A}^k}{k!} \geq 0 \tag{5.0.2}$$

since $\widehat{A}$ is nonnegative. The notation $s(A)$ and $\widehat{A}$ will be used throughout this chapter. The exponential of an essentially nonnegative matrix has important applications in continuous-time Markov processes and positive linear dynamical systems [64, 111, 144]. Recently, the exponential of the adjacency matrix (which is symmetric and nonnegative) is shown to be involved in the measurement analysis of networks [46]. In addition, in some applications it is required to compute all entries of $\exp(A)$ accurately, see [167]. Thus, it is desirable to design algorithms which compute the matrix exponential with high componentwise relative accuracy. Componentwise relative perturbation bounds of $\exp(A)$ have been established in [168, 166], which offer the possibility of accurate computation.

The computation of the exponential of an essentially nonnegative matrix has been studied in, e.g., [7, 43, 107, 167]. In [7, 107], scaling and squaring methods built on diagonal Padé approximation are used. These approaches only produce small normwise errors while componentwise errors on small entries are not taken into account. The truncated Taylor series method is adopted in [43, 167] to avoid cancellations in the computation and to achieve componentwise accuracy by preserving the nonnegativity. Another polynomial approximation method has been proposed in [167], aiming at reducing the computational cost when A is sparse. Among these attempts to this problem, the truncated Taylor series method seems to

be the most promising one so far when A is dense. More specifically, let

$$L_{m,n}(A) = \exp[s(A)]\, T_m\Big(\frac{\widehat{A}}{n}\Big)^n. \tag{5.0.3}$$

The existing Taylor series method presented in [43, 167] chooses n to satisfy $\rho(\widehat{A})/n < 1$, and the desirable componentwise accuracy is achieved by making $T_m(\widehat{A}/n)$ approximate $\exp(\widehat{A}/n)$ with high componentwise relative accuracy. This approach involves a large truncation order m and potentially requires $\mathcal{O}(N)$ matrix multiplications to achieve componentwise accuracy when A is sparse (e.g., A is narrow banded). To illustrate this, we first consider the case when A is tridiagonal and irreducible. Notice that $\widehat{A}^k$ is a $(2k)$-banded matrix (by *b-banded*, we mean that all (i,j)-th entries with $|i-j| > b/2$ are zero, see also Section 6.2). Evidently $\exp(A)$ is a full matrix since

$$\operatorname{struct}\big[\exp(A)\big] = \bigcup_{k=0}^{\infty} \operatorname{struct}\big(\widehat{A}^k\big) = \bigcup_{k=0}^{N-1} \operatorname{struct}\big(\widehat{A}^k\big).$$

The second equality is a consequence of the Cayley-Hamilton theorem. By taking a truncation order $m < N-1$, $T_m(\widehat{A}/n)$ is only $(2m)$-banded and hence does not have any relative accuracy to $\exp(\widehat{A}/n)$ in the $(N,1)$-th entry. For a general sparse matrix A, we consider the directed graph $G(\widehat{A}) = (V,E)$ represented by $\widehat{A}$ (see, e.g., [25]). Once the distance between some pair of nodes (i,j) in $G(\widehat{A})$ is $\Theta(N)$, then to ensure $\operatorname{struct}\big[T_m(\widehat{A}/n)\big] = \operatorname{struct}\big[\exp(\widehat{A}/n)\big]$, which is a necessary condition of achieving high componentwise relative accuracy, the order m cannot be chosen less than $\Theta(N)$. Therefore, the existing Taylor series method requires at least $\Theta(N^4)$ operations in some cases. We will show in Section 5.3 that $\mathcal{O}(N^4)$ is also enough.

To improve the efficiency of the truncated Taylor series method, we make an observation that high componentwise relative accuracy of $T_m(\widehat{A}/n)$ to $\exp(\widehat{A}/n)$ is *not* necessarily needed to attain high componentwise relative accuracy of $L_{m,n}(A)$ to $\exp(A)$. This is inspired by the Euler method which produces $\exp(\widehat{A}) \approx L_{1,n}(\widehat{A})$. Though $T_1(\widehat{A}/n) = I + \widehat{A}/n$ may have no componentwise relative accuracy to $\exp(\widehat{A}/n)$ at all, $\big[T_1(\widehat{A}/n)\big]^n$ can still have high componentwise relative accuracy to $\exp(\widehat{A})$ with n sufficiently large. A similar observation is made for higher order Runge-Kutta methods. This suggests to select the scale factor n a bit larger to improve the efficiency of the algorithm. We will establish novel error estimates for $L_{m,n}(A)$ so that the componentwise accuracy can be guaranteed by a moderate truncation order m and only $\mathcal{O}(\log N)$ matrix multiplications are required. In this sense, we call our method the *aggressively truncated Taylor series method*.

The rest of this chapter is largely based on the manuscript [134] submitted for publication in *SIAM J. Matrix Anal. Appl.*. It is organized as follows. In Section 5.1, we discuss the conditioning for the computation of exponentials of essentially nonnegative matrices. Several componentwise relative truncation error bounds are developed in Section 5.2. Section 5.3 establishes the aggressive truncated Taylor series method based on these truncation error bounds. Rounding errors are analyzed in Section 5.4 to show that our algorithms are highly

accurate in floating-point arithmetic. Finally in Section 5.5, numerical examples are presented to demonstrate the efficiency and accuracy of the aggressively truncated Taylor series method.

5.1 Componentwise perturbation analysis

For a given essentially nonnegative matrix A, we would like to compute every entry of $\exp(A)$ accurately. In this case the normwise condition number $\kappa_{\exp}(A)$ defined in (4.1.9) does not reflect the sensitivity of the problem. Hence a tailored componentwise perturbation analysis is required. Recently, it has been shown in [166] that small relative errors in the off-diagonal entries of A cause small relative errors in the entries of $\exp(A)$. The main theorem in [166] is listed below. We remark that the perturbations on diagonal entries and off-diagonal entries are treated separately, since there is a natural difference between diagonal entries and off-diagonal entries by the definition of essentially nonnegative matrices. For a unified treatment of these perturbations, we refer to [166, Theorem 2].

Theorem 5.1.1 ([166])**.** *Let $A \in \mathbb{R}^{N\times N}$ be essentially nonnegative. Suppose that $\Delta A \in \mathbb{R}^{N\times N}$ satisfies $|\Delta a_{ij}| \le \varepsilon_1 a_{ij}$ (for $i \neq j$) and $|\Delta a_{ii}| \le \varepsilon_2$, where $\varepsilon_1 \in [0,1)$, $\varepsilon_2 \ge 0$. Then for any $t \ge 0$ we have*

$$\left|\exp[t(A+\Delta A)] - \exp(tA)\right| \le \delta(t)\,\exp[\delta(t)]\exp(tA),$$

where

$$\delta(t) = t\varepsilon_2 + \left[N - 1 + \rho(\widehat{A})\,t\right]\frac{\varepsilon_1}{1-\varepsilon_1}.$$

Proof. See [166, Theorem 1]. □

Here we are mainly interested in the case $t = 1$. We define

$$C(A) = N - 1 + \rho(\widehat{A}), \tag{5.1.1}$$

which can be regarded as an upper bound of the *condition number* of $\exp(A)$ in the sense of relative componentwise perturbations. This number depends on the quantity $\rho(\widehat{A})$, i.e., the spectral radius of the shifted matrix $\widehat{A}$. The following result provides a bound for $\rho(\widehat{A})$ under the assumption that entries of $\exp(A)$ can be represented by floating-point numbers without overflow.

Theorem 5.1.2. *Let $A \in \mathbb{R}^{N\times N}$ be essentially nonnegative. If there is no overflow in* $\mathrm{fl}[\exp(A)]$*, then*

$$\rho(\widehat{A}) \le \ln N + \ln R_{\max} - s(A), \tag{5.1.2}$$

where $R_{\max}$ is the largest positive finite floating-point number.

Proof. As $\exp(A)$ does not cause overflow in its floating-point representation, we have

$$R_{\max} \geq \max_{i,j} \big|[\exp(A)]_{ij}\big| \geq \frac{1}{N} \big\|\exp(A)\big\|_\infty = \frac{\exp[s(A)]}{N} \big\|\exp(\widehat{A})\big\|_\infty \geq \frac{\exp[s(A)]}{N} \rho\big(\exp(\widehat{A})\big).$$

Since $\widehat{A}$ is nonnegative, applying the Perron-Frobenius theorem, we obtain

$$\rho\big(\exp(\widehat{A})\big) = \exp\big(\max_{\lambda \in \Lambda(\widehat{A})} \Re(\lambda)\big) = \exp[\rho(\widehat{A})].$$

Therefore,

$$R_{\max} \geq \frac{\exp[s(A)] \exp[\rho(\widehat{A})]}{N},$$

which implies (5.1.2). □

***Remark* 5.1.3.** Unless $s(A)$ is small, Theorem 5.1.2 implies that $\rho(\widehat{A})$ cannot be too large. For example, if $A \in \mathbb{R}_+^{10000\times 10000}$ and $\exp(A)$ can be represented by IEEE-754 double precision floating-point numbers ($R_{\max} \approx 1.8 \times 10^{308}$, see [80]), then

$$\rho(\widehat{A}) \leq \rho(A) \leq \ln 10000 + \ln R_{\max} < 719.$$

Theorem 5.1.2 indicates that the matrix exponential problem is always well-conditioned for an essentially nonnegative matrix A if $s(A)$ is not too small, under the assumption that $\exp(A)$ does not cause overflow. If we assume $|s(A)| = \mathcal{O}(N)$, then a consequence of Theorem 5.1.2 is that $C(A) = \mathcal{O}(N)$. Some difficult problems, e.g., when $\exp(A)$ overflows or $|s(A)| = \omega(N)$, are beyond the scope of this thesis. Consequently, for the problems we are interested, those absolute perturbations on diagonal entries in Theorem 5.1.1 can also be regarded as relative ones, since $|a_{ii}|$'s are all moderately small. As another remark, we do not take into account relative accuracy on really small entries which are close to or below $R_{\min}$, since this is an infeasible requirement in practice.

5.2 Approximation using truncated Taylor series

In this section, we establish some novel error estimates for the truncated Taylor series approximation of the matrix exponential. Both a priori and a posteriori estimates are established. These error estimates will be used to develop componentwise accurate algorithms for computing $\exp(A)$ for an essentially nonnegative A in the next section.

5.2.1 Lower bound for $\exp(A)$

It is evident from the definition of $L_{m,n}(A)$ in (5.0.3) that $L_{m,n}(A) \leq \exp(A)$ when A is essentially nonnegative. Several normwise a priori error bounds for $T_m(A/n)^n$ can be found in

the existing literatures (see, e.g., [34, 48, 103]). These bounds only yield

$$\left\|\exp\Big(\frac{A}{n}\Big)-T_m\Big(\frac{A}{n}\Big)\right\| \le \frac{\|A/n\|^{m+1}}{(m+1)!}\Big(1-\frac{\|A/n\|}{m+2}\Big)^{-1}$$

when $\|A\| < n(m+2)$, but do not take advantage of the squaring procedure. In [109], a normwise backward error bound which makes use of squaring is provided. But the full understanding of $\kappa_{\exp}(A)$, which is still open [73], is required when turning it to a forward error bound. In [139], a componentwise error bound

$$\left|\exp(A)-T_m\Big(\frac{A}{n}\Big)^n\right| \le \frac{1}{2}\left[\Big(1+\frac{(2|A|/n)^{m+1}}{(m+1)!}\Big)^n-1\right]$$

is established for an *intensity matrix* A.[1] Despite that this estimate only bounds absolute errors rather than relative ones, it seems to be so far the closest to our target. To our knowledge, there is no existing a priori componentwise error bound available for general essentially nonnegative matrices. In the following, we provide such an a priori componentwise error bound. We first bound the error $\exp(A)-L_{m,n}(A)$ in terms of $\widehat{A}^{m+1}\exp(A)$.

Lemma 5.2.1. *If $A\in\mathbb{R}^{N\times N}$ is essentially nonnegative, then for any positive integers m and n we have*

$$\lim_{n\to\infty} L_{m,n}(A) = \lim_{m\to\infty} L_{m,n}(A) = \exp(A). \tag{5.2.1}$$

Furthermore, the truncation error is bounded by

$$\exp[s(A)]\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1} \le \exp(A)-L_{m,n}(A) \le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(A). \tag{5.2.2}$$

Proof. Since (5.2.1) is an immediate consequence of (5.2.2), it suffices to verify (5.2.2) only. Let

$$R_m(x) = \exp(x)-T_m(x) = \sum_{k=m+1}^{\infty}\frac{x^k}{k!}.$$

Then we have

$$R_m(X) = \sum_{k=m+1}^{\infty}\frac{X^k}{k!} \ge \frac{X^{m+1}}{(m+1)!} \ge 0$$

for any nonnegative matrix X, and thus

$$\exp\Big(\frac{\widehat{A}}{n}\Big) \ge T_m\Big(\frac{\widehat{A}}{n}\Big).$$

1. An intensity matrix is an essentially nonnegative matrix with zero row sums.

Using the identity $x^n - y^n = (x-y)\sum_{k=0}^{n-1} x^k y^{n-1-k}$ when $xy = yx$, we obtain

$$\begin{aligned}\exp(A) - L_{m,n}(A) &= \exp[s(A)]\left[\exp(\widehat{A}) - T_m\Big(\frac{\widehat{A}}{n}\Big)^n\right] \\ &= \exp[s(A)]\left[\exp\Big(\frac{\widehat{A}}{n}\Big)^n - T_m\Big(\frac{\widehat{A}}{n}\Big)^n\right] \\ &= \exp[s(A)]R_m\Big(\frac{\widehat{A}}{n}\Big)\sum_{k=0}^{n-1}\exp\Big(\frac{\widehat{A}}{n}\Big)^k T_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1-k}.\end{aligned}$$

On the one hand, the inequality

$$\exp(A) - L_{m,n}(A) \geq \exp[s(A)]\frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\cdot n T_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1} = \exp[s(A)]\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1}$$

obviously holds. On the other hand, notice that

$$R_m\Big(\frac{\widehat{A}}{n}\Big) = \sum_{k=0}^{\infty}\frac{(\widehat{A}/n)^{m+1+k}}{(m+1+k)!} \leq \sum_{k=0}^{\infty}\frac{(\widehat{A}/n)^{m+1+k}}{(m+1)!\,k!} = \frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\exp\Big(\frac{\widehat{A}}{n}\Big).$$

Thus, we obtain

$$\begin{aligned}\exp(\widehat{A}) - L_{m,n}(\widehat{A}) &\leq \exp[s(A)]\frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\exp\Big(\frac{\widehat{A}}{n}\Big)\cdot n\exp\Big(\frac{\widehat{A}}{n}\Big)^{n-1} \\ &= \exp[s(A)]\frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp\Big(\frac{\widehat{A}}{n}\Big)^n \\ &= \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(A).\end{aligned}$$

□

Lemma 5.2.1 illustrates the convergence speed of $L_{m,n}(A)$ to $\exp(A)$ and provides a componentwise error estimate. But it is still not an a priori componentwise relative truncation error for $L_{m,n}(A)$. To this end, we have to bound $\widehat{A}^{m+1}\exp(A)$ in terms of $\exp(A)$ and need the following tools.

Lemma 5.2.2 ([166]). *If $A \in \mathbb{R}_+^{N\times N}$, then*

$$A\exp(A) \leq C(A)\exp(A).$$

Proof. See [166, Lemma 2]. □

When A is essentially nonnegative, then Lemma 5.2.2 yields that

$$\widehat{A}\exp(\widehat{A}) \leq C(\widehat{A})\exp(\widehat{A}) = C(A)\exp(\widehat{A}). \tag{5.2.3}$$

A more general consequence is the following corollary.

Corollary 5.2.3. *Let A be essentially nonnegative. Then*

$$\widehat{A}^k \exp(A) \le C(A)^k \exp(A), \qquad (k \in \mathbb{N}). \tag{5.2.4}$$

More generally, let $F(x) = \sum_{k=0}^{\infty} \alpha_k x^k$ with $\alpha_k \ge 0$ $(k = 0, 1, 2, \dots)$ be an analytic function in the disk $\{z \in \mathbb{C} : |z| < R_0\}$. If $C(A) < R_0$, then

$$F(\widehat{A}) \exp(A) \le F[C(A)] \exp(A). \tag{5.2.5}$$

Proof. We first show (5.2.4) by induction. It is a trivial case for $k = 0$; and multiplying $\exp[s(A)]$ to both sides of (5.2.3) yields the conclusion for $k = 1$. For $k > 1$, by induction, we have

$$\widehat{A}^k \exp(A) \le \widehat{A}\big[\widehat{A}^{k-1} \exp(A)\big] \le C(A)^{k-1} \widehat{A} \exp(A) \le C(A)^k \exp(A).$$

This completes the proof of (5.2.4). Consequently, we obtain

$$F(\widehat{A}) \exp(A) = \sum_{k=0}^{\infty} \alpha_k \widehat{A}^k \exp(A) \le \sum_{k=0}^{\infty} \alpha_k C(A)^k \exp(A) = F[C(A)] \exp(A). \qquad \square$$

Now using these tools, we easily obtain the componentwise truncation error estimate as follows.

Theorem 5.2.4. *Let $A \in \mathbb{R}^{N \times N}$ be an essentially nonnegative matrix. Then*

$$0 \le \exp(A) - L_{m,n}(A) \le \frac{C(A)^{m+1}}{n^m (m+1)!} \exp(A) \tag{5.2.6}$$

holds for any positive integers m and n.

Proof. The conclusion (5.2.6) is a direct consequence from applying (5.2.4) to (5.2.2). $\square$

5.2.2 Upper bound for $\exp(A)$

We have already seen in Theorem 5.2.4 that $L_{m,n}(A)$ is a lower bound of $\exp(A)$. Sometimes an upper bound of $\exp(A)$ is of interest. Hence we define

$$\begin{aligned} \widetilde{T}_m(x) &= T_m(x) + \frac{x^{m+1}}{m!\, m} \left(1 - \frac{x}{m}\right)^{-1} \\ &= 1 + x + \cdots + \frac{x^m}{m!} + \frac{x^{m+1}}{m!\, m} + \frac{x^{m+2}}{m!\, m^2} + \cdots \qquad (|x| < m) \end{aligned}$$

and

$$U_{m,n}(A) = \exp[s(A)]\, \widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n. \tag{5.2.7}$$

Notice that $\widetilde{T}_m(x)$ is an mth order rational approximation of $\exp(x)$ which can be expressed in the form

$$\widetilde{T}_m(x) = \frac{\left(1-\frac{x}{m}\right)T_{m-2}(x) + \frac{x^{m-1}}{(m-1)!}}{1-\frac{x}{m}}.$$

By the uniqueness of the Padé approximant [13], $\widetilde{T}_m(x)$ is in fact the $(m-1,1)$ Padé approximant of $\exp(x)$, i.e.,

$$\widetilde{T}_m(x) = R_{m-1,1}(x).$$

The negative of the corresponding remainder is denoted as

$$\widetilde{R}_m(x) = \widetilde{T}_m(x) - \exp(x) = \sum_{k=1}^{\infty}\Big(\frac{1}{m!\,m^k} - \frac{1}{(m+k)!}\Big)x^{m+k},$$

which has nonnegative coefficients in its Maclaurin series expansion since $(m+k)! > m!\,m^k$ for any positive integer k. Consequently, for an essentially nonnegative matrix A with $\rho(\widehat{A}) < mn$,

$$\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big) = \sum_{k=1}^{\infty}\Big(\frac{1}{m!\,m^k} - \frac{1}{(m+k)!}\Big)\Big(\frac{\widehat{A}}{n}\Big)^{m+k}$$

is always nonnegative and then

$$U_{m,n}(A) \geq \exp[s(A)]\exp\Big(\frac{\widehat{A}}{n}\Big)^n = \exp(A).$$

Here we require the condition $\rho(\widehat{A}) < mn$, which allows us to make use of the Maclaurin series expansion of $\widetilde{R}_m(x)$ when evaluating $\widetilde{R}_m(\widehat{A}/n)$. Similar to Theorem 5.2.4, we obtain a componentwise a priori error bound for $U_{m,n}(A)$ as follows.

Theorem 5.2.5. *Let $A \in \mathbb{R}^{N\times N}$ be an essentially nonnegative matrix. Then for any positive integers m and n satisfying $mn > C(A)$, we have*

$$0 \leq U_{m,n}(A) - \exp(A) \leq \left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1\right]\exp(A). \tag{5.2.8}$$

Proof. It is straightforward to verify by the definition of $U_{m,n}(A)$ that

$$U_{m,n}(A) - \exp(A) = \exp[s(A)]\left[U_{m,n}(\widehat{A}) - \exp(\widehat{A})\right].$$

As we have already shown that $U_{m,n}(A) \geq \exp(A)$, it suffices to show

$$U_{m,n}(\widehat{A}) - \exp(\widehat{A}) = \widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n - \exp(\widehat{A}) \leq \left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1\right]\exp(\widehat{A}).$$

For any nonnegative matrix X with $\rho(X/m) < 1$, we have $\widetilde{R}_m(X) \geq 0$. Using a rough estimate that $\exp(X) = I + \sum_{k=1}^{\infty} X^k/k! \geq I$, we obtain

$$\widetilde{T}_m(X) = \exp(X) + \widetilde{R}_m(X) \leq \exp(X)\left[I + \widetilde{R}_m(X)\right].$$

Substituting X by $\widehat{A}/n$ into this inequality leads to

$$\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n - \exp(\widehat{A}) \leq \left[\exp\Big(\frac{\widehat{A}}{n}\Big)\Big(I + \widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\Big)\right]^n - \exp(\widehat{A}) = \left[\Big(I + \widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\Big)^n - I\right]\exp(\widehat{A}). \quad (5.2.9)$$

Notice that

$$\left[1 + \widetilde{R}_m\Big(\frac{x}{n}\Big)\right]^n - 1 = \sum_{k=1}^{n}\binom{n}{k}\widetilde{R}_m\Big(\frac{x}{n}\Big)^k$$

has nonnegative coefficients in its Maclaurin series expansion. Then by Corollary 5.2.3, we have

$$\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n - \exp(\widehat{A}) \leq \left[\Big(I + \widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\Big)^n - I\right]\exp(\widehat{A}) \leq \left[\Big(1 + \widetilde{R}_m\Big(\frac{C(\widehat{A})}{n}\Big)\Big)^n - 1\right]\exp(\widehat{A}),$$

because $C(\widehat{A}) < mn$. Thus, the conclusion is proved. □

Notice that

$$\widetilde{R}_m(x) = \Big(\frac{1}{m!\,m} - \frac{1}{(m+1)!}\Big)x^{m+1} + \mathcal{O}(x^{m+2}) = \frac{x^{m+1}}{(m+1)!\,m} + \mathcal{O}(x^{m+2})$$

as $x \to 0$. Then once $\widetilde{R}_m\big(C(A)/n\big)$ is small, we have

$$\left[\Big(1 + \widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\Big)^n - 1\right]\exp(A) \approx \frac{C(A)^{m+1}}{n^m(m+1)!\,m}\exp(A),$$

which is similar to (5.2.6) in Theorem 5.2.4.

5.2.3 A posteriori error bounds for $L_{m,n}(A)$ and $U_{m,n}(A)$

The a priori error bounds provided in Theorems 5.2.4 and 5.2.5 rely on Lemma 5.2.2 and Corollary 5.2.3. However, we remark that these bounds are not always satisfactory since the estimates (5.2.4) and (5.2.5) can sometimes overestimate the error a lot. For example, when $A = I$, the inequality (5.2.4) overestimates the bound by a factor of $(N-1)^k$, which is certainly not negligible when either N or k is large. Thus, it is desirable to derive some more accurate error estimates. Here we do not attempt to improve the a priori bounds we have established in this section. Instead we will develop some a posteriori bounds, which are also useful in practice.

We first derive an a posteriori estimate for $L_{m,n}(A)$. In some existing Taylor series meth-

ods [43, 167], the a posteriori error is estimated via

$$\exp\Big(\frac{\widehat{A}}{n}\Big) - T_m\Big(\frac{\widehat{A}}{n}\Big) \le \frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\left[I - \frac{\widehat{A}}{n(m+2)}\right]^{-1}$$

and hence a stopping criterion

$$\frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\left[I - \frac{\widehat{A}}{n(m+2)}\right]^{-1} \le \tau \cdot T_m\Big(\frac{\widehat{A}}{n}\Big), \tag{5.2.10}$$

which similar to the normwise one (4.2.2), can be applied, where τ is the desired (relative) accuracy prescribed by the user. But a drawback of this estimate is that the corresponding stopping criterion can be difficult to satisfy since the effectiveness of the squaring stage is completely neglected in this estimate. Therefore we will take into account the squaring stage and establish an estimate for $T_m(\widehat{A}/n)^n$ rather than $T_m(\widehat{A}/n)$. Notice that (5.2.2) is already quite close to our purpose, except that $\exp(A)$ is of course unknown in practice. Hence we aim at bounding $\exp(A)$ from above in terms of $T_m(\widehat{A}/n)$. To this end, we assume that m is chosen such that

$$\frac{C(A)^{m+1}}{n^m(m+1)!} \le \frac{1}{2},$$

which is relatively easy to satisfy. Under this extra assumption, Theorem 5.2.4 implies

$$T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \exp(\widehat{A}) \le 2\,T_m\Big(\frac{\widehat{A}}{n}\Big)^n,$$

indicating that $T_m(\widehat{A}/n)^n$ is not very far away from $\exp(\widehat{A})$. Then using (5.2.2), we obtain

$$\exp(A) - L_{m,n}(A) \le \frac{2\widehat{A}^{m+1}}{n^m(m+1)!}\,T_m\Big(\frac{\widehat{A}}{n}\Big)^n, \tag{5.2.11}$$

which is a good a posteriori estimate when $T_m(\widehat{A}/n)^n$ or $L_{m,n}(A)$ is already available. Since the derivation here does not heavily rely on (5.2.4), we expect (5.2.11) to be more accurate compared to (5.2.6).

The a posteriori bound for $U_{m,n}(A)$ is simpler compared to that for $L_{m,n}(A)$. We first show the following lemma which is similar to Lemma 5.2.1.

Lemma 5.2.6. *If $A \in \mathbb{R}^{N\times N}$ is essentially nonnegative, then for positive integers m and n satisfying $mn > \rho(\widehat{A})$, we have*

$$\begin{aligned}\exp[s(A)]\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\exp\Big(\frac{n-1}{n}\widehat{A}\Big) &\le U_{m,n}(A) - \exp(A)\\ &\le \exp[s(A)]\frac{\widehat{A}^{m+1}}{n^m m!\,m}\Big(I - \frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1}.\end{aligned} \tag{5.2.12}$$

Proof. Because $U_{m,n}(A) = \exp[s(A)]U_{m,n}(\widehat{A})$, the conclusion is equivalent to

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\exp\Big(\frac{n-1}{n}\widehat{A}\Big) \le U_{m,n}(\widehat{A}) - \exp(\widehat{A}) \le \frac{\widehat{A}^{m+1}}{n^m m!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1}.$$

Using the same technique as in Lemma 5.2.1, we obtain

$$U_{m,n}(\widehat{A}) - \exp(\widehat{A}) = \widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n - \exp\Big(\frac{\widehat{A}}{n}\Big)^n = \widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\sum_{k=0}^{n-1}\exp\Big(\frac{\widehat{A}}{n}\Big)^k\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1-k}.$$

Since

$$\widetilde{R}_m(x) = \frac{x^{m+1}}{(m+1)!\,m} + \mathscr{O}(x^{m+2}), \qquad (x\to 0)$$

and all coefficients in its Maclaurin series expansion are nonnegative, we have

$$\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big) \ge \frac{(\widehat{A}/n)^{m+1}}{(m+1)!\,m}.$$

Combining with the fact that

$$\exp\Big(\frac{\widehat{A}}{n}\Big) \le \widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big),$$

we obtain

$$U_{m,n}(\widehat{A}) - \exp(\widehat{A}) \ge \frac{(\widehat{A}/n)^{m+1}}{(m+1)!\,m}\cdot n\exp\Big(\frac{\widehat{A}}{n}\Big)^{n-1} = \frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\exp\Big(\frac{n-1}{n}\widehat{A}\Big).$$

For the upper bound of the error, we also make use of the series expansion

$$\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big) = \sum_{k=1}^{\infty}\Big(\frac{1}{m!\,m^k} - \frac{1}{(m+k)!}\Big)\Big(\frac{\widehat{A}}{n}\Big)^{m+k} \le \sum_{k=1}^{\infty}\frac{1}{m!\,m^k}\Big(\frac{\widehat{A}}{n}\Big)^{m+k} = \frac{(\widehat{A}/n)^{m+1}}{m!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}.$$

Then

$$\begin{aligned} U_{m,n}(\widehat{A}) - \exp(\widehat{A}) &\le \frac{(\widehat{A}/n)^{m+1}}{m!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\cdot n\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1} \\ &= \frac{\widehat{A}^{m+1}}{n^m m!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{n-1}{n}\widehat{A}\Big). \end{aligned}$$

The proof of the lemma is now complete. □

5.3 Aggressively truncated Taylor series method

In this section, we develop several algorithms based on the error estimates presented in Section 5.2.

5.3.1 Algorithms based on a priori estimates

Theorem 5.2.4 offers a useful a priori componentwise error estimate for the development of an algorithm.Let τ denote the desired accuracy provided by the user. Then it is natural to choose m and n satisfying

$$\frac{C(A)^{m+1}}{n^m(m+1)!} \leq \tau \tag{5.3.1}$$

and then use $L_{m,n}(A)$ to approximate $\exp(A)$. Figure 5.1 shows some sample values of the coefficient in (5.2.6), where $C(A) = 32$, 128, 512, and 2048, respectively. The computational cost for evaluating $L_{m,n}(A)$ as defined in (5.0.3) is roughly $m + \log_2 n$ matrix multiplications [109]. Loosely speaking, if we would like to minimize the computational cost, it is sensible to choose the tangent line with slope equal to one and pick the point of tangency on the contour line. A more sophisticated choice for the parameters will be presented below in Algorithm 5.1.

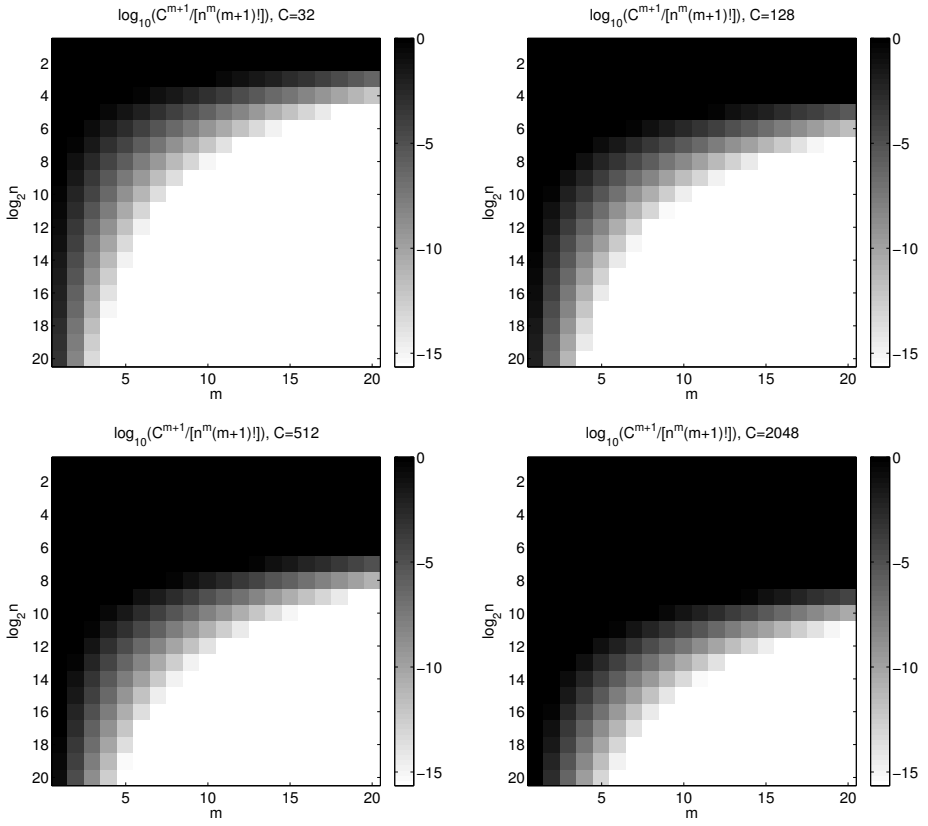


Figure 5.1 – Sample values of $C(A)^{m+1}/[n^m(m+1)!]$. The color scale corresponds to $\log_{10}\left[C(A)^{m+1}/[n^m(m+1)!]\right]$. Bright values correspond to small truncation errors, whereas dark values correspond to large errors.

***Remark* 5.3.1.** From the plots in Figure 5.1, the optimal value (for the worst case) of both m and $\log_2 n$ are of the magnitude 10^1 for a wide range of matrices. Preferable values of m and $\log_2 n$ are close to the boundary of the white area.

Table 5.1 – Number of matrix multiplications, π_m, required for evaluating $T_m(x)$.

m	2	3	4	5	6	7	8	9	10	11
π_m	1	2	2	3	3	4	4	4	5	5
m	12	13	14	15	16	17	18	19	20	21
π_m	5	6	6	6	6	7	7	7	7	8

Similar suggestions for m and n have been proposed in [109] based on normwise error estimates. Here we conclude that both m and $\log_2 n$ are still small even if componentwise accuracy is desired. For example, if $m = 13$ is chosen, $n = 4C(A)$ is usually sufficient for attaining double precision accuracy which indicates that overscaling never occurs in practice. Moreover, both Theorem 5.2.4 and the rounding error analysis for repeated squaring (see Lemma 5.4.2 in Section 5.4) are pessimistic [109]. Thus the true error can be even smaller.

There are several methods to evaluate the matrix polynomial $T_m(X)$. A simple approach $T_m(X) \leftarrow T_{m-1}(X) + X^m/m!$ and Horner's method both require $\Theta(m)$ matrix multiplications. Compared to existing Taylor series methods, an advantage of knowing m a priori is that there exist cheaper alternatives for evaluating $T_m(X)$ (see, e.g., [119]). For example,

$$\begin{aligned} T_6(X) &= \left(I + X + \frac{X^2}{2!}\right) + X^3\left(\frac{I}{3!} + \frac{X}{4!} + \frac{X^2}{5!} + \frac{X^3}{6!}\right), \\ T_7(X) &= \left(I + \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!}\right) + X\left(I + \frac{X^2}{3!} + \frac{X^4}{5!} + \frac{X^6}{7!}\right). \end{aligned} \tag{5.3.2}$$

They both require two matrix multiplications less than the naive approach. The upper bound [2] of computational cost for $1 \le m \le 21$ is summarized in Table 5.1. We remark that the numbers in Table 5.1 are smaller than these values in [74, Table 2.2], since there is no need to treat odd powers and even powers separately when evaluating the truncated Taylor series. With the reformulation technique in (5.3.2), it is sensible to choose m and n minimizing $\pi(m) + \log_2 n$ so that the computational cost of evaluating $L_{m,n}(A)$ is minimized. Certainly, by taking into account the rounding error due to finite precision arithmetic, a smaller n is preferred once there are multiple choices for the optimal parameter setting.

Now we are ready to derive our aggressively truncated Taylor series method (shown in Algorithm 5.1) based on the above discussions. Some remarks on the algorithm are also provided below.

***Remark* 5.3.2.** An important feature of Algorithm 5.1 is that $\pi(m) + \log_2 n$ is of order $\mathcal{O}(\log N)$ when $\mathrm{fl}[\exp(A)]$ does not overflow and $|s(A)| = \mathcal{O}(N)$. To see this, we first conclude from (5.1.1) and (5.1.2) that $C(A) = \mathcal{O}(N)$. Then by Theorem 5.2.4, it is sufficient to choose $m_0 = 1$ and $n_0 = \lceil C(A)^2/(2\tau) \rceil$ in order to pass the a priori test (5.3.1). Hence

$$\min\left[\pi(m) + \log_2 n\right] \le \pi(m_0) + \log_2 n_0 = \mathcal{O}(\log C(A)^2) = \mathcal{O}(\log N).$$

2. The numbers are obtained by using the optimal settings in the Paterson-Stockmeyer method [67, 119].

Algorithm 5.1 Aggressively truncated Taylor series method for $\exp(A)$, with A essentially nonnegative

Input: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $\tau > \mathbf{u}$.

1: (optional) Balancing.
2: Estimate $\rho(\widehat{A})$ (e.g., via power method).
3: Solve the discrete optimization problem

$$\pi(m) + \log_2 n = \min, \quad \text{s.t. } \frac{C(A)^{m+1}}{n^m (m+1)!} \le \tau$$

by enumerating over m, $\log_2 n \in \{1, 2, \ldots, 21\}$.
4: $E \leftarrow L_{m,n}(A)$.
5: (optional) Reversed balancing.

As a consequence, the complexity of Algorithm 5.1 is $\mathscr{O}(N^3 \log N)$. In contrast, to achieve componentwise high relative accuracy, the Taylor series methods presented in [43, 167] for essentially nonnegative matrices require $\mathscr{O}(N^4)$ operations as $m = \mathscr{O}(N)$ and $\log_2 n = \mathscr{O}(\log\log N)$ (e.g., choosing $m = \lceil 2\mathrm{e}C(A)\rceil = \mathscr{O}(N)$ is already sufficient for a modest τ). This is the main advantage of our new method. When the required accuracy is modestly low (e.g., only four correct digits are of interest), the new method becomes even more attractive while existing Taylor series methods still require $\mathscr{O}(N^4)$ operations to produce all nonzero entries in $\exp(A)$.

***Remark* 5.3.3.** Some extra care must be taken in the shifting stage. If we compute $\exp(\widehat{A})$ and $\exp[s(A)]$ straightforwardly, it is possible that $\exp(\widehat{A})$ causes overflow but $\exp[s(A)]$ is small enough so that the desired result $\exp(A)$ is still representable with floating-point numbers. Therefore shifting needs to be handled in a nontrivial way when $s(A) < 0$. Since

$$\exp(A) = \left[\exp\Big(\frac{A}{n}\Big)\right]^n = \left[\exp\Big(\frac{s(A)}{n}\Big)\exp\Big(\frac{\widehat{A}}{n}\Big)\right]^n,$$

the shift $s(A)/n$ on A/n becomes much safer when n is large. The same observation can also be applied to $L_{m,n}(A)$, i.e.,

$$L_{m,n}(A) = \left[\exp\Big(\frac{s(A)}{n}\Big) T_m\Big(\frac{\widehat{A}}{n}\Big)\right]^n, \tag{5.3.3}$$

By applying scaling before shifting, we are most likely able to avoid unnecessary overflow even when $s(A)$ is small. This trick is commonly used in computations of Markov models (e.g., see [107, 136, 167]). Moreover, we suggest that the shifting strategy (5.3.3) is always used even for $A \ge 0$ since some entries of $\exp(\widehat{A})$ might underflow. In case that $|s(A)|$ is extremely large so that n has to be large to avoid overflow/underflow, we conclude from Section 5.1 that $C(A)$ is also large. Such problems need special attention, and are not investigated in this thesis.

***Remark* 5.3.4.** Balancing is a commonly used technique when computing the matrix exponential. Notice that balancing does not change $\rho(\widehat{A})$ in our problem so it does not help reducing $C(A)$ and the computational cost. Therefore, we provide balancing as an optional step. For

Algorithm 5.2 A priori upper bound algorithm for $\exp(A)$, with A essentially nonnegative

Input: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $\tau > \mathbf{u}$.

1: (optional) Balancing.
2: Estimate $\rho(\widehat{A})$ (e.g., via power method).
3: Solve the discrete optimization problem

$$\pi(m-1)+\log_2 n = \min, \quad \text{s.t. } mn > 2\rho(\widehat{A}) \text{ and } \left[1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right]^n - 1 \le \tau$$

by enumerating over $m, \log_2 n \in \{1,2,\ldots,21\}$.
4: $E \leftarrow U_{m,n}(A)$.
5: (optional) Reversed balancing.

essentially nonnegative matrices, since $\|A\|_\infty$ can be reduced by balancing, the technique potentially reduces the chance of overflow/underflow on intermediate results during the computation. Although balancing is not always safe for some eigenvalue problems [159], extensive numerical experiments in [138] show that it is unlikely to be harmful when computing matrix functions.

Similar to the algorithm for $L_{m,n}(A)$, it is straightforward to develop an algorithm for the upper bound $U_{m,n}(A)$ based on Theorem 5.2.5. Algorithm 5.2 is an upper bound version in analogy to Algorithm 5.1. A notable difference between $U_{m,n}(A)$ and $L_{m,n}(A)$ is that the matrix inverse is needed for evaluating the upper bound. Since $I - \widehat{A}/(mn)$ is an *M-matrix*,[3] its inverse can be calculated accurately using GTH-type algorithms [5]. Actually by increasing the scale factor to $2n$, $\left[I - \widehat{A}/(2mn)\right]^{-1}$ can be safely computed even by standard Gaussian elimination without pivoting because it is always well-conditioned in the sense of componentwise perturbation (see Lemma 5.4.5 in Section 5.4). Subtractions on the diagonal never cause severe cancellation and hence only introduce small relative backward errors. Both GTH-type algorithms and Gaussian elimination can adopt block variants so that most computations are performed in level 3 BLAS. This feature is useful when performance is important.

5.3.2 Algorithms based on a posteriori estimates

The a priori error estimates provided in Theorems 5.2.4 and 5.2.5 lead to simple and efficient approaches for computing $\exp(A)$. However, rounding error is not taken into account when determining the parameters. Also, we have seen in Section 5.2.3 that sometimes Corollary 5.2.3 can overestimate the truncation error quite a lot. Therefore the choice of m and n might be too large. As an extreme case, if the user has an unfeasibly high requirement on the accuracy (e.g., $\tau = 10^{-12}$ using IEEE single precision arithmetic), the parameters m and n which bound the truncation error well below τ may cause significant rounding error. A sensible choice is to return a nearly optimal solution in the working precision and to report a

3. A square matrix A is called an M-matrix if $-A$ is essentially nonnegative and A^{-1} is nonnegative.

warning message to the user. From Section 5.3.1, we have already seen that both m and $\log_2 n$ are of order 10^1 in the aggressively truncated Taylor series method for a wide range of matrices. Since the rounding error depends more sensitively on $\log_2 n$ compared to m, we can use a fixed truncation order m (e.g., $m = 13$) and seek for an appropriate scale factor n to avoid overscaling. In the following, we derive such kind of iterative approaches based on a posteriori error estimates we have developed in Section 5.2.3.

In existing Taylor series methods, the truncation order m is determined by an a posteriori estimate (5.2.10). This criterion is generally too conservative to be satisfied when being applied to the aggressively truncated method since $T_m(\widehat{A}/n)$ might not have any relative accuracy to $\exp(\widehat{A}/n)$. The estimate (5.2.11) is more useful when the truncation order m is relatively small since the scale factor n is also taken into account. A simple a posteriori error estimate

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \frac{\tau}{2}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \tag{5.3.4}$$

can be used as a criterion which ensures $\exp(A) - L_{m,n}(A) \le \tau \exp(A)$. In fact,

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \frac{\tau}{1+2\tau}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \tag{5.3.5}$$

is a better one which also guarantees $\exp(A) - L_{m,n}(A) \le \tau \exp(A)$. To see this, we conclude from (5.3.5) and Lemma 5.2.1 that

$$\begin{aligned}\exp(\widehat{A}) - T_m\Big(\frac{\widehat{A}}{n}\Big)^n &\le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(\widehat{A}) \\ &\le \frac{2\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \frac{2\tau}{1+2\tau}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \frac{2\tau}{1+2\tau}\exp(\widehat{A}).\end{aligned}$$

Repeating the same procedure with the new bound, we arrive at

$$\begin{aligned}\exp(\widehat{A}) - T_m\Big(\frac{\widehat{A}}{n}\Big)^n &\le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(\widehat{A}) \\ &\le \frac{1}{1-\frac{2\tau}{1+2\tau}}\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \frac{\frac{\tau}{1+2\tau}}{1-\frac{2\tau}{1+2\tau}}\exp(\widehat{A}) = \tau\exp(\widehat{A}).\end{aligned}$$

Condition (5.3.5) is relaxed roughly by half compared to (5.3.4). This has the potential benefit to reduce $\log_2 n$ by one.

The condition (5.3.5) has several advantages over (5.2.10). For example, (5.3.5) is inverse-free and hence cheaper than (5.2.10). A more important feature is that even if (5.3.5) is not satisfied for a given choice of (m,n), say

$$\varepsilon_{m,n} = \min\left\{\varepsilon\colon \frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\Big(\frac{\widehat{A}}{n}\Big)^n \le \varepsilon T_m\Big(\frac{\widehat{A}}{n}\Big)^n\right\} > \frac{\tau}{1+2\tau},$$

Algorithm 5.3 Lower bound iteration for $\exp(A)$, with A essentially nonnegative

Input: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, MAXITER $\in \mathbb{N}$.

(optional) Balancing.
$k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
$\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
while $\varepsilon_0 \geq \varepsilon \geq \frac{\tau}{1+2\tau}$ **and** $k \leq$ MAXITER **do**
  $n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
  $L \leftarrow L_{m,n}(A)$
  $W \leftarrow \dfrac{n(\widehat{A}/n)^{m+1}}{(m+1)!} L$
  $\varepsilon \leftarrow \max\left\{ \dfrac{W(i,j)}{L(i,j)} : W(i,j) > 0 \right\}$
  $k \leftarrow k + \dfrac{1}{m} \log_2 \dfrac{\varepsilon}{\tau}$
end while
$E \leftarrow L$
if $\varepsilon \geq \tau$ **then**
  Report failure.
end if
(optional) Reversed balancing.

it still suggests an appropriate choice of the parameters. For example, it is expected that $\varepsilon_{m,2n} \sim 2^{-m}\varepsilon_{m,n}$ when $\varepsilon_{m,n}$ is small because the convergence rate provided in Lemma 5.2.1 is reasonably accurate. This kind of heuristics often rapidly leads to a suitable choice of (m,n). The typical situation in practice is that an appropriate parameter setting can be found immediately after an initial guess. As a consequence of the above discussion, an iterative method based on the a posteriori error estimate is presented in Algorithm 5.3. As pointed out in Section 4.2, this algorithm can also be regarded as an mth order Runge-Kutta method. In practice we recommend that $m = 13$ is used so that $n < 4C(A)$ is sufficient for $N \leq 2048$. Usually, no more than two iterations are required in Algorithm 5.3. Therefore the computational cost, which is also $\mathcal{O}(N^3 \log N)$, is not much more expensive than Algorithm 5.1. Therefore the computational cost is also $\mathcal{O}(N^3 \log N)$, and Algorithm 5.3 is not much more expensive than Algorithm 5.1.

***Remark* 5.3.5.** If $\exp(A)v$ (for some vector $v \geq 0$) instead of $\exp(A)$ is of interest, the stopping criterion can be adjusted to

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!} T_m\left(\frac{\widehat{A}}{n}\right)^n v \leq \frac{\tau}{1+2\tau} T_m\left(\frac{\widehat{A}}{n}\right)^n v,$$

which might be easier to satisfy than (5.3.5).

Similarly, we can establish an iterative approach for $U_{m,n}(A)$, based on Lemma 5.2.6. We

can simply use

$$\frac{\widehat{A}^{m+1}}{n^m m!\, m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1} \le \frac{\tau}{1+\tau}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n} \tag{5.3.6}$$

or

$$\frac{\widehat{A}^{m+1}}{n^m m!\, m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n} \le \frac{\tau}{1+\tau}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n}, \tag{5.3.7}$$

which both imply

$$\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n} - \exp(\widehat{A}) \le \tau \exp(\widehat{A})$$

and hence

$$U_{m,n}(A) - \exp(A) \le \tau \exp(A).$$

Notice that $\big[I-\widehat{A}/(mn)\big]^{-1}$ has already been calculated when evaluating $\widetilde{T}_m(\widehat{A}/n)$, so that these stopping criteria only require matrix multiplications. We remark that if (5.3.6) is used, it is advisable to choose n in the form $n=2^k+1$ so that $\widetilde{T}_m(\widehat{A}/n)^{n-1}$ can be cheaply computed. The iterative algorithm for $U_{m,n}(A)$ is listed in Algorithm 5.4.

5.3.3 Interval algorithms with improved accuracy

When both $L_{m,n}(A)$ and $U_{m,n}(A)$ are computed, it is possible to make use of these bounds to gain a more accurate approximation of $\exp(A)$. In the following, we demonstrate such a technique which potentially improves the accuracy of the solution using these bounds. In addition, we present a novel high accuracy interval-type algorithm based on these bounds.

Sometimes both an upper bound and a lower bound of $\exp(A)$ are of interest. Certainly we can calculate them separately using the algorithms we have presented. But we expect more useful outputs by merging them. Notice that

$$U_{m,n}(A) - L_{m,n}(A) = \big[U_{m,n}(A) - \exp(A)\big] + \big[\exp(A) - L_{m,n}(A)\big].$$

The difference between the upper bound and the lower bound is also componentwise small compared to $\exp(A)$. Thus, once both $L_{m,n}(A)$ and $U_{m,n}(A)$ have already been calculated, a simpler a posteriori error estimate is automatically available. In principle, any matrix X satisfying $L_{m,n}(A) \le X \le U_{m,n}(A)$ is a good approximation of $\exp(A)$ when both bounds are sufficiently accurate. We seek for a higher order approximation of the form $\eta L_{m,n}(A) + (1-\eta)U_{m,n}(A)$ where $\eta \in [0,1]$, i.e., a convex combination of $L_{m,n}(A)$ and $U_{m,n}(A)$. It is straightforward to verify that

$$\lim_{x\to 0}\frac{\exp(x)-T_m(x/n)^n}{x^{m+1}} = m\lim_{x\to 0}\frac{\widetilde{T}_m(x/n)^n-\exp(x)}{x^{m+1}} = \frac{1}{n^m(m+1)!}.$$

Algorithm 5.4 Upper bound iteration for $\exp(A)$, with A essentially nonnegative

Input: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, $\texttt{MAXITER} \in \mathbb{N}$.

(optional) Balancing.
$k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
$\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
while $\varepsilon_0 \geq \varepsilon \geq \frac{\tau}{1+\tau}$ **and** $k \leq \texttt{MAXITER}$ **do**
  $n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
  Try $U \leftarrow U_{m,n}(A)$
  if $\rho(\widehat{A}) \geq mn$ **then**
    $W \leftarrow \dfrac{n(\widehat{A}/n)^{m+1}}{(m+1)!\,m}\left(I - \dfrac{\widehat{A}}{mn}\right)^{-1} U$
    $\varepsilon \leftarrow \max\left\{\dfrac{W(i,j)}{U(i,j)} : U(i,j) > 0\right\}$
    $k \leftarrow k + \dfrac{1}{m}\log_2\dfrac{\varepsilon}{\tau}$
  else
    $k \leftarrow k + 1$
  end if
end while
$E \leftarrow U$
if $\varepsilon \geq \tau$ **then**
  Report failure.
end if
(optional) Reversed balancing.

Therefore,

$$E_{m,n}(A) = \frac{1}{m+1}L_{m,n}(A) + \frac{m}{m+1}U_{m,n}(A)$$

is an $(m+1)$-th order approximation of $\exp(A)$. The idea here is closely related to Richardson extrapolation [124]. A method based on Richardson extrapolation on $L_{1,n}(A)$ has been proposed in [154]. In contrast to that method, which is numerically unstable [115], our approach based on interpolation is stable since it preserves nonnegativity in floating-point arithmetic. Using the same techniques for deriving Theorems 5.2.4 and 5.2.5, we establish the following error estimate.

Theorem 5.3.6. *Let $A \in \mathbb{R}^{N\times N}$ be essentially nonnegative. Then*

$$\begin{aligned} 0 \leq E_{m,n}(A) - \exp(A) \\ \leq \frac{m}{m+1}\left[\left(1 + \widetilde{R}_m\left(\frac{C(A)}{n}\right)\right)^n - 1 + \frac{C(A)^{m+2} - C(A)^{m+1}}{n^m(m+1)!\,m}\right]\exp(A) \end{aligned} \quad (5.3.8)$$

holds for any positive integers m and n satisfying $mn > C(A)$.

Proof. Multiplying $\exp[-s(A)]$ to (5.3.8) leads to an equivalent inequality

$$0 \le E_{m,n}(\widehat{A}) - \exp(\widehat{A}) \le \frac{m}{m+1}\left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1 + \frac{C(A)^{m+2} - C(A)^{m+1}}{n^m(m+1)!\,m}\right]\exp(\widehat{A}).$$

In the following we prove this version which involves $\widehat{A}$ only. On the one hand, from the proofs of Lemmas 5.2.1 and 5.2.6, we have

$$\begin{aligned} L_{m,n}(\widehat{A}) - \exp(\widehat{A}) &\ge -nR_m\Big(\frac{\widehat{A}}{n}\Big)\exp\Big(\frac{\widehat{A}}{n}\Big)^{n-1}, \\ U_{m,n}(\widehat{A}) - \exp(\widehat{A}) &\ge n\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\exp\Big(\frac{\widehat{A}}{n}\Big)^{n-1}. \end{aligned}$$

Then

$$\begin{aligned} E_{m,n}(\widehat{A}) - \exp(\widehat{A}) &= \frac{m}{m+1}\big[U_{m,n}(\widehat{A}) - \exp(\widehat{A})\big] + \frac{1}{m+1}\big[L_{m,n}(\widehat{A}) - \exp(\widehat{A})\big] \\ &\ge \frac{1}{m+1}\left[m\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big) - R_m\Big(\frac{\widehat{A}}{n}\Big)\right]\exp\Big(\frac{n-1}{n}\widehat{A}\Big). \end{aligned}$$

Notice that the function

$$f(k) = \frac{1}{m!\,m^{k-1}} \Big/ \frac{m+1}{(m+k)!}$$

satisfies that $f(1) = 1$ and $f(k+1)/f(k) = (m+k+1)/m > 1$ for $k > 1$. Therefore, we have $f(k) \ge 1$ for any positive integer k and then

$$m\widetilde{R}_m(x) - R_m(x) = \sum_{k=1}^{\infty}\left[\frac{1}{m!\,m^{k-1}} - \frac{m+1}{(m+k)!}\right]x^{m+k}$$

has nonnegative coefficients in the Maclaurin series expansion. Hence we obtain

$$m\widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big) - R_m\Big(\frac{\widehat{A}}{n}\Big) \ge 0,$$

and then

$$E_{m,n}(\widehat{A}) - \exp(\widehat{A}) \ge 0.$$

On the other hand, Lemma 5.2.1 implies that

$$\begin{aligned} L_{m,n}(\widehat{A}) - \exp(\widehat{A}) &\le -\frac{\widehat{A}^{m+1}}{n^m(m+1)!} \\ &= \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\big[\exp(\widehat{A}) - L_{1,0}(\widehat{A})\big] - \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(\widehat{A}) \\ &\le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\big[\widehat{A}\exp(\widehat{A})\big] - \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\exp(\widehat{A}) \end{aligned}$$

$$= \frac{\widehat{A}^{m+2} - \widehat{A}^{m+1}}{n^m (m+1)!} \exp(\widehat{A}).$$

Therefore, combining with (5.2.9), we have

$$\begin{aligned} E_{m,n}(\widehat{A}) - \exp(\widehat{A}) &= \frac{m}{m+1} \left[\big(U_{m,n}(\widehat{A}) - \exp(\widehat{A})\big) + \frac{1}{m}\big(L_{m,n}(\widehat{A}) - \exp(\widehat{A})\big) \right] \\ &\le \frac{m}{m+1} \left[\left(1 + \widetilde{R}_m\Big(\frac{\widehat{A}}{n}\Big)\right)^n - 1 + \frac{\widehat{A}^{m+2} - \widehat{A}^{m+1}}{n^m (m+1)!\, m} \right] \exp(\widehat{A}). \end{aligned}$$

To prove the desired inequality

$$E_{m,n}(\widehat{A}) - \exp(\widehat{A}) \le \frac{m}{m+1} \left[\left(1 + \widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1 + \frac{C(A)^{m+2} - C(A)^{m+1}}{n^m (m+1)!\, m} \right] \exp(\widehat{A}),$$

it remains to verify that

$$g(x) = \left[1 + \widetilde{R}_m\Big(\frac{x}{n}\Big)\right]^n - 1 + \frac{x^{m+2} - x^{m+1}}{n^m (m+1)!\, m} =: \sum_{k=0}^{\infty} \alpha_k x^k$$

has nonnegative coefficients in its Maclaurin series expansion. Actually, all α_k's except α_{m+1} are obviously nonnegative as

$$g(x) = \sum_{k=1}^{n} \binom{n}{k} \widetilde{R}_m\Big(\frac{x}{n}\Big)^k + \frac{x^{m+2}}{n^m (m+1)!\, m} - \frac{x^{m+1}}{n^m (m+1)!\, m}.$$

For α_{m+1}, we use the fact that

$$\widetilde{R}_m(x) = \frac{x^{m+1}}{(m+1)!\, m} + \mathcal{O}(x^{m+2}), \qquad (x \to 0)$$

to conclude that $g(x) = \mathcal{O}(x^{m+2})$ as $x \to 0$, and thus $\alpha_{m+1} = 0$. This completes the proof of the theorem. □

***Remark* 5.3.7.** From the definition of $\widetilde{T}_m(x)$, we can see that terms up to mth order are shared when evaluating $T_m(\widehat{A}/n)$ and $\widetilde{T}_m(\widehat{A}/n)$ simultaneously. This observation is also valid for some alternative approaches such as (5.3.2).

In applications [14] where accurate bounds are extremely important, it is desirable that

$$\underline{\mathrm{fl}}[L_{m,n}(A)] \le L_{m,n}(A) \le \exp(A) \le U_{m,n}(A) \le \overline{\mathrm{fl}}[U_{m,n}(A)] \tag{5.3.9}$$

is guaranteed regardless of rounding errors. In this case $L_{m,n}(A)$ and $U_{m,n}(A)$ need to be calculated separately using different rounding modes and no computational cost can be saved. The technique of convex combination can still be applied although $\mathrm{fl}\big[E_{m,n}(A)\big]$ is not guaranteed to be another upper bound of $\exp(A)$. A complete algorithm is summarized as Algorithm 5.5 which provides both lower and upper bounds and a higher order approximation

Algorithm 5.5 Interval algorithm for $\exp(A)$, with A essentially nonnegative

Input: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, $\texttt{MAXITER} \in \mathbb{N}$.

(optional) Balancing.
$k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
$\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
$L \leftarrow A$, $U \leftarrow +\infty$
while $\varepsilon_0 \geq \varepsilon \geq \tau$ **and** $k \leq \texttt{MAXITER}$ **do**
  $n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
  Evaluate $L_0 \leftarrow \underline{\mathrm{fl}}[L_{m,n}(A)]$ under the rounding mode *round towards* $-\infty$
  $L \leftarrow \max\{L, L_0\}$
  (Try) Evaluate $U_0 \leftarrow \overline{\mathrm{fl}}[U_{m,n}(A)]$ under the rounding mode *round towards* $+\infty$
  $U \leftarrow \min\{U, U_0\}$
  $\varepsilon \leftarrow \max\left\{\dfrac{U(i,j) - L(i,j)}{L(i,j)} : U(i,j) > 0\right\}$
  $k \leftarrow k + \dfrac{1}{m}\log_2\dfrac{\varepsilon}{\tau}$
end while
$E \leftarrow \dfrac{1}{m+1}L + \dfrac{m}{m+1}U$
if $\varepsilon \geq \tau$ **then**
  Report failure.
end if
(optional) Reversed balancing.

in between. We will verify in Section 5.4.3 that the property (5.3.9) can indeed be achieved. Because of this property, Algorithm 5.5 can be regarded as an interval algorithm [110] but without any explicit use of interval arithmetic. As a remark, some different interval algorithms for computing $\exp(A)$ (A is not necessarily essentially nonnegative) can be found in [27, 56]. These methods bound the remainders of truncated Taylor series [56] or Padé approximant [27], and both involve interval arithmetic. The rounding error analysis in Section 5.4 demonstrates that this interval algorithm does not severely overestimate the error. Generally the a posteriori error $U_{m,n}(A) - L_{m,n}(A)$ is also much smaller than the a priori error based on truncation and rounding error analysis. In practice, Algorithm 5.5 is more robust than other algorithms presented in this section, while it is also more costly.

5.4 Rounding error analysis

In this section we provide error bounds taking into account effects of finite precision arithmetic. Although these bounds usually overestimate the errors in practice, we do not attempt to refine them as tightly as possible since the main purpose is to provide evidence for the forward stability of our algorithms. We assume that there is no overflow or (gradual) underflow in the calculation so that the rounding model

$$\mathrm{fl}(\alpha \circ \beta) = (\alpha \circ \beta)(1+\epsilon), \qquad |\epsilon| \leq \mathbf{u} \tag{5.4.1}$$

is applicable, where "$\circ$" is $+$, $-$, $\times$, or $\div$, and $\mathbf{u}$ is the unit roundoff, see Appendix B for details. In addition,

$$\mathrm{fl}\big[\exp(\alpha)\big] = \exp(\alpha)(1+\epsilon), \qquad |\epsilon| \le \mathbf{u}$$

is assumed. We further assume that $N^2\mathbf{u} \ll 1$, which is a reasonable requirement in practice, so that all $\mathscr{O}(\mathbf{u}^2)$ terms are negligible.

5.4.1 Rounding error in $\mathrm{fl}\big[L_{m,n}(A)\big]$

Firstly, we show that the matrix polynomial $T_m(A/n)$ can be computed stably when A is nonnegative. We assume that $T_m(A/n)$ is evaluated via

$$\mathrm{fl}\left[T_m\Big(\frac{A}{n}\Big)\right] \leftarrow \mathrm{fl}\left[T_{m-1}\Big(\frac{A}{n}\Big)\right] + \frac{A}{mn}\cdot\mathrm{fl}\left[\frac{1}{(m-1)!}\Big(\frac{A}{n}\Big)^{m-1}\right]. \tag{5.4.2}$$

Those cheaper alternatives for evaluating $T_m(A/n)$ such as (5.3.2) require fewer matrix multiplications than the simple scheme (5.4.2) and hence very likely have smaller rounding error bounds (see, e.g., [73, Theorem 4.5]). It is not difficult to prove the following conclusion.

Lemma 5.4.1. *If $A \in \mathbb{R}_+^{N\times N}$, then for any positive integers m and n, we have*

$$\left|\mathrm{fl}\left[T_m\Big(\frac{A}{n}\Big)\right] - T_m\Big(\frac{A}{n}\Big)\right| \le \big[(m+1)(N+2)\mathbf{u} + \mathscr{O}(\mathbf{u}^2)\big]\,T_m\Big(\frac{A}{n}\Big).$$

Proof. The proof can be found in [48, Theorem 1.1.1]. We remark that our bound is slightly different compared to the one in [48] because we require two more operations here—the division by n in $\mathrm{fl}(A/n) \leftarrow A/n$ and the division by m in $\mathrm{fl}\big[(A/n)^m/m!\big] \leftarrow \mathrm{fl}\big[(A/n)\cdot[(A/n)^{m-1}/(m-1)!]\big]/m$. □

For the scaling procedure, Lemma 5.4.2 below is in a slightly different form compared to Theorem 4.9 in [7]. We also provide a different proof for this lemma. Compared to (4.2.1), the error bound here is much tighter. Therefore, the squaring stage for nonnegative matrices is already much safer than that for general matrices. Although the error bound for repeated squaring is nearly attainable in the worst case, it is often too pessimistic in practice. We have already seen in Section 5.3 that sometimes perturbations on a nonnegative matrix B can even decay during the repeated squaring process. The full understanding of the squaring phase is still an open problem [109]. It is good to keep in mind that repeated squaring can be dangerous, but that this is not always the case.

Lemma 5.4.2. *Let $\Delta B \in \mathbb{R}^{N\times N}$ be the perturbation of a nonnegative matrix B satisfying $|\Delta B| \le \varepsilon B$ for some $\varepsilon \in [0,1)$. Then for $n = 2^k$, we have*

$$\big|\mathrm{fl}\big[(B+\Delta B)^n\big] - B^n\big| \le \big[n(N\mathbf{u}+\varepsilon) + \mathscr{O}(\varepsilon^2 + \varepsilon\mathbf{u} + \mathbf{u}^2)\big]B^n.$$

Proof. First, the condition $|\Delta B| \le \varepsilon B$ is equivalent to

$$(1-\varepsilon)B \le B + \Delta B \le (1+\varepsilon)B.$$

Then by induction, we obtain

$$(1-\varepsilon)^n (1-N\mathbf{u})^{n-1} B^n \le \mathrm{fl}\big[(B+\Delta B)^n\big] \le (1+\varepsilon)^n (1+N\mathbf{u})^{n-1} B^n.$$

Therefore

$$\big|\mathrm{fl}\big[(B+\Delta B)^n\big] - B^n\big| \le \big[n(N\mathbf{u}+\varepsilon) + \mathscr{O}(\varepsilon^2 + \varepsilon\mathbf{u} + \mathbf{u}^2)\big] B^n. \qquad \square$$

Making use of Lemmas 5.4.1 and 5.4.2, we obtain Theorem 5.4.3 which provides the rounding error bound for Algorithms 5.1 and 5.3. We remark that a similar theorem can be found in [48]. Our estimate, which is tailored to nonnegative matrices, is more concise and has a simpler proof compared to [48, Theorem 1.1.9]. The error estimate in Theorem 5.4.3 is satisfactory, since the scale factor n used by Algorithms 5.1, which appears in the error bound, is proportional to $C(A)$.

Theorem 5.4.3. *Let A be an $N \times N$ essentially nonnegative matrix and m, n be positive integers satisfying* $\exp(A) - L_{m,n}(A) \le \tau \exp(A)$. *Then*

$$\big|\mathrm{fl}\big[L_{m,n}(A)\big] - \exp(A)\big| \le \big[\tau + n(m+2)(N+3)\mathbf{u} + \mathscr{O}(\mathbf{u}^2)\big] \exp(A).$$

Proof. Let $B = \exp[s(A)/n]\, T_m(\widehat{A}/n)$, $\Delta B = \mathrm{fl}\big[\exp[s(A)/n]\, T_m(\widehat{A}/n)\big] - B$, and $\varepsilon = (m+1)(N+3)\mathbf{u} + O(\mathbf{u}^2)$. Then by Lemma 5.4.2, we have

$$\big|\mathrm{fl}\big[L_{m,n}(A)\big] - L_{m,n}(A)\big| \le \big[n(m+2)(N+3)\mathbf{u} + \mathscr{O}(\mathbf{u}^2)\big] \exp(A).$$

Taking into account the truncation error bound τ yields the conclusion. $\square$

5.4.2 Rounding error in $\mathrm{fl}\big[U_{m,n}(A)\big]$

To analyze the rounding error for $\mathrm{fl}\big[U_{m,n}(A)\big]$, the inverse of an M-matrix is the only different part compared to $\mathrm{fl}\big[L_{m,n}(A)\big]$. Let M be an $N \times N$ nonsingular M-matrix with the Jacobi splitting $M = D - K$ where D is diagonal and K has zero diagonal entries. Then Lemma 5.4.4 characterizes the sensitivity of M^{-1} in terms of $\gamma = \rho\big(D^{-1}K\big)$.

Lemma 5.4.4 ([165])**.** *Assume that M is an $N \times N$ nonsingular M-matrix. Let ΔM be a perturbation to M satisfying $|\Delta M| \le \varepsilon |M|$ where $0 \le \varepsilon < (1-\gamma)/(1+\gamma)$. Then*

$$\big|(M+\Delta M)^{-1} - M^{-1}\big| \le \left[\frac{2-\gamma}{1-\gamma} N\varepsilon + \mathscr{O}(\varepsilon^2)\right] M^{-1}.$$

Proof. See [165, Theorem 2.4]. $\square$

When $mn > 2\rho(\widehat{A})$, the M-matrix $M = I - \widehat{A}/(mn)$ is well-conditioned in the componentwise sense with $\gamma < 1/2$. This can be verified through

$$\rho\big(D^{-1}K\big) \le \frac{1}{2}\rho\left[\left(D - \frac{1}{2}I\right)^{-1}K\right] < \frac{1}{2},$$

due to the fact that $I/2 - \widehat{A}/(mn) = (D - I/2) - K$ is also an M-matrix. Moreover, the LU factorization of M satisfies $s(U) \ge 1/2$, indicating that M^{-1} can likely be calculated componentwise accurately via standard Gaussian elimination. Lemma 5.4.5 below confirms this and provides a rounding error bound for the following algorithm.

(1) Perform one step Gaussian elimination without pivoting for calculating the LU factorization:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = \begin{bmatrix} 1 & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} \\ & S_{22} \end{bmatrix}, \tag{5.4.3}$$

where S_{22} is the Schur complement.

(2) Compute $\mathrm{fl}\big(S_{22}^{-1}\big) \leftarrow \mathrm{fl}\big[\mathrm{fl}(S_{22})^{-1}\big]$ recursively.

(3) Compute $\mathrm{fl}\big(M^{-1}\big)$ through

$$\mathrm{fl}\big(M^{-1}\big) = \mathrm{fl}\left(\begin{bmatrix} M_{11}^{-1} + M_{11}^{-1}M_{12}\,\mathrm{fl}\big(S_{22}^{-1}\big)L_{21} & M_{11}^{-1}M_{12}\,\mathrm{fl}\big(S_{22}^{-1}\big) \\ -\mathrm{fl}\big(S_{22}^{-1}\big)L_{21} & \mathrm{fl}\big(S_{22}^{-1}\big) \end{bmatrix}\right).$$

Lemma 5.4.5. *Let $B \in \mathbb{R}_+^{N\times N}$ satisfying $\rho(B) < 1/2$. Then the inverse of $M = I - B$ calculated by the algorithm above satisfies*

$$\left|\mathrm{fl}\big(M^{-1}\big) - M^{-1}\right| \le \big[10N(N+1)\mathbf{u} + \mathscr{O}(\mathbf{u}^2)\big]M^{-1}. \tag{5.4.4}$$

Proof. The conclusion holds trivially for $N = 1$. For $N > 1$, we analyze the rounding error by induction.

In Step (1), the computed vector $\mathrm{fl}(L_{21})$ satisfies

$$\mathrm{fl}(L_{21}) = \frac{M_{21}}{M_{11}} + x,$$

where

$$|x| \le \mathbf{u}\left|\frac{M_{21}}{M_{11}}\right| = \mathbf{u}\,|L_{21}|.$$

Similarly, $\mathrm{fl}(S_{22})$ satisfies

$$\mathrm{fl}(S_{22}) = M_{22} - \mathrm{fl}(L_{21})M_{12} + Y,$$

where

$$|Y| \le \big[2\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\big(|M_{22}| + \mathrm{fl}(L_{21})M_{12}\big) = \big[2\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\big(|M_{22}| + L_{21}M_{12}\big).$$

Notice that all diagonal entries of S_{22} lie in the interval $[1/2, 1]$ as $\rho(B) < 1/2$. It can be easily verified that $|M_{22}| + L_{21}M_{12} \le 3|S_{22}|$ and hence

$$|\mathrm{fl}(S_{22}) - S_{22}| \le \big[6\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\,|S_{22}|.$$

In Step (2), we use the induction hypothesis

$$\big|\mathrm{fl}\big[\mathrm{fl}(S_{22})^{-1}\big] - \mathrm{fl}(S_{22})^{-1}\big| \le \big[10N(N-1)\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\,\mathrm{fl}(S_{22})^{-1}.$$

Therefore, applying Lemma 5.4.4 yields

$$\begin{aligned}\big|\mathrm{fl}\big[\mathrm{fl}(S_{22})^{-1}\big] - S_{22}^{-1}\big| &\le \big|\mathrm{fl}\big[\mathrm{fl}(S_{22})^{-1}\big] - \mathrm{fl}(S_{22})^{-1}\big| + \big|\mathrm{fl}(S_{22})^{-1} - S_{22}^{-1}\big| \\ &\le \big[10N(N-1)\mathbf{u} + 18(N-1)\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]S_{22}^{-1} \\ &\le \big[(10N^2 + 8N)\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]S_{22}^{-1}.\end{aligned}$$

Finally, in Step (3), the relative error propagated is bounded by $2N\mathbf{u} + \mathcal{O}(\mathbf{u}^2)$. Therefore, combining this error with the error in $\mathrm{fl}\big(S_{22}^{-1}\big)$ yields the conclusion. □

Based on the lemmas we have developed, it is now ready to estimate the rounding error for $U_{m,n}(A)$ similar to Theorem 5.4.3.

Theorem 5.4.6. *Let A be an $N \times N$ essentially nonnegative matrix and m, n be positive integers satisfying $mn > 2\rho(\widehat{A})$ and $U_{m,n}(A) - \exp(A) \le \tau \exp(A)$. Then*

$$\big|\mathrm{fl}[U_{m,n}(A)] - \exp(A)\big| \le \big[\tau + n(m+10N)(N+3)\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\exp(A).$$

Proof. Using the same technique for Lemma 5.4.1, it can be verified that

$$\left|\mathrm{fl}\left[\exp\Big[\frac{s(A)}{n}\Big]\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)\right] - \exp\Big[\frac{s(A)}{n}\Big]\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)\right| \le \big[(m+10N)(N+3)\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big]\,\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big).$$

Then let $B = \exp[s(A)/n]\,\widetilde{T}_m(\widehat{A}/n)$, $\Delta B = \mathrm{fl}\big[\exp[s(A)/n]\,\widetilde{T}_m(\widehat{A}/n)\big] - B$, and $\varepsilon = (m+10N)(N+3)\mathbf{u} + O(\mathbf{u}^2)$. The conclusion is obtained by applying Lemma 5.4.2. □

5.4.3 Rounding error under biased rounding modes

We have confirmed in the previous subsections that Algorithms 5.1–5.4 have high componentwise accuracy. Algorithm 5.5 requires calculations under biased rounding modes to achieve the property (5.3.9). In this case the rounding errors can still be modeled by (5.4.1),

while the unit roundoff here is twice as large as the one under the standard rounding mode (i.e., *round towards nearest*) [112]. Therefore, Theorems 5.4.3 and 5.4.6 are valid for the interval algorithm.

The task left is to verify that the property (5.3.9) can indeed be achieved by carefully switching the rounding modes. Obviously, $\underline{\mathrm{fl}}\big[L_{m,n}(A)\big] \le L_{m,n}(A)$ is satisfied by simply switching the rounding mode to *round towards* $-\infty$. For the upper bound, the tricky part is to achieve $\overline{\mathrm{fl}}\big(M^{-1}\big) \ge M^{-1}$, where $M = I - A/(mn)$ is an M-matrix. [4] Firstly, $\mathrm{fl}(M)$ is formed by $\mathrm{fl}(M) = -\overline{\mathrm{fl}}\big[\overline{\mathrm{fl}}[A/(mn) - I]\big] \le M$. Consider one step of Gaussian elimination (5.4.3). The factorization can be computed through

$$\mathrm{fl}(L_{21}) = -\overline{\mathrm{fl}}(|L_{21}|) = -\overline{\mathrm{fl}}\left[\frac{\overline{\mathrm{fl}}(|M_{21}|)}{\underline{\mathrm{fl}}(M_{11})}\right] \le L_{21},$$
$$\mathrm{fl}(S_{22}) = -\overline{\mathrm{fl}}\big[\overline{\mathrm{fl}}(|L_{21}|)\,\overline{\mathrm{fl}}(|M_{12}|) - \underline{\mathrm{fl}}(M_{22})\big] \le S_{22}.$$

By applying the same procedure recursively to S_{22}, we conclude that the LU factorization $M = LU$ satisfies

$$\underline{\mathrm{fl}}(L) = \mathrm{fl}(L) \le L, \qquad \underline{\mathrm{fl}}(U) = \mathrm{fl}(U) \le U.$$

Suppose M^{-1} is computed by solving two triangular linear systems under the rounding mode *round towards* $+\infty$, the solution satisfies

$$\overline{\mathrm{fl}}\big(M^{-1}\big) \ge \underline{\mathrm{fl}}(U)^{-1}\,\underline{\mathrm{fl}}(L)^{-1} \ge U^{-1}L^{-1} = M^{-1}.$$

The conclusion can be generalized to block LU factorization and some algorithmic variants for computing M^{-1} as long as $\mathrm{fl}(L)$ and $\mathrm{fl}(U)$ are true lower bounds of L and U, respectively. However, the analysis does not carry over to GTH-type algorithms since the diagonal entries of S_{22} are computed in a different way. Therefore, if the actual upper bound is important, we prefer standard Gaussian elimination rather than GTH-type algorithms. In practice all calculations can be done without switching the rounding mode (under the rounding mode *round towards* $+\infty$), using the trick $\underline{\mathrm{fl}}(x) = -\overline{\mathrm{fl}}(-x)$.

5.5 Numerical experiments

In this section, we present and discuss some experimental results. All experiments are performed on a Linux machine with an Intel Xeon quadcore 3.10GHz CPU and 8GB memory. We make use of only a single core for the experiments. Both Algorithm 5.1 and Algorithm 5.5 are implemented in C99 and compiled by the GNU C complier. Matrix multiplications are performed by the optimized (single-threaded) GEMM routine from the OpenBLAS library; while other operations, such as block LU factorization and solving triangular systems, are

4. Unfortunately, GTH-type algorithms [5] usually do not have this property.

carefully implemented, see Section 5.4.3. For the parameter setting, we choose $m = 8$ as the truncation order in Algorithm 5.5. It is relatively small compared to the recommended value ($m = 13$) and allows us to observe some features of this algorithm which rarely occur under the recommended setting. The relative tolerances is set to $\tau = 1024N\mathbf{u} \approx 2.3 \times 10^{-13}N$. The maximum iteration number is set to `MAXITER` = 52 which is more than needed. Balancing is not used in the experiments to demonstrate that our algorithm is not sensitive to bad scaling in the original matrix. The switching of rounding modes is accomplished by the standard library function `fesetround()`. As a comparison, we also run the same set of tests on the same machine for MATLAB's `expm` function (version R2012a) which is a general purpose solver. We choose nine commonly used test matrices. All entries in the solutions can be represented by normalized floating-point numbers (i.e., no overflow or (gradual) underflow).

Example 5.1 ([43])**.**

$$A = \begin{bmatrix} -0.01 & 10^{15} \\ 0 & -0.01 + 10^{-6} \end{bmatrix}.$$

Example 5.2. A matrix modified from an example in C. Moler's blog [5]:

$$A = \begin{bmatrix} 0 & e & 0 \\ a+b & -d & a \\ c & 0 & -c \end{bmatrix},$$

where $a = 2 \times 10^{10}$, $b = \frac{2}{3} \times 10^{8}$, $c = \frac{200}{3}$, $d = 3$, $e = 10^{-8}$.

Example 5.3 ([38])**.**

$$A = \begin{bmatrix} -16 & 2^{60} & 2^{60} & 2^{60} \\ 0 & -16 & 2^{60} & 2^{60} \\ 0 & 0 & -1 & 2^{60} \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Example 5.4 ([155])**.** The 10×10 Forsythe matrix

$$A = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 10^{-10} & & & 0 \end{bmatrix}_{10\times 10}.$$

5. C. Moler, A Balancing Act for the Matrix Exponential, http://blogs.mathworks.com/cleve/2012/07/23/a-balancing-act-for-the-matrix-exponential/.

Example 5.5 ([167])**.**

$$A = -T_{50} = \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}_{50\times 50}.$$

Example 5.6. The 128×128 Jordan block with zero eigenvalues

$$A = J_{128}(0) = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}_{128\times 128}.$$

Example 5.7 ([167])**.** The adjacency matrix of a 2-nearest ring network of 200 nodes with four extra shortcuts 16–30, 74–85, 90–128, 138–147, modeling a small-world network. It is generated by `smallw(200,2,0.03)`, where `smallw.m` is from the MATLAB toolbox CONTEST [6] [145].

Example 5.8 ([167])**.** $A = -T_{40,40} = -(T_{40} \otimes I_{40} + I_{40} \otimes T_{40})$.

Example 5.9.

$$A = 1400 \times J_{2048}(-1/2) = \begin{bmatrix} -700 & 1400 & & \\ & \ddots & \ddots & \\ & & \ddots & 1400 \\ & & & -700 \end{bmatrix}_{2048\times 2048}.$$

Table 5.2 lists the componentwise errors

$$\min\left\{\varepsilon \geq 0\colon \left|\mathrm{fl}\left[\exp(A)\right] - \exp(A)\right| \leq \varepsilon \exp(A)\right\}$$

for these testing examples. For Algorithm 5.5, we present the a posteriori error estimate computed from $U_{m,n}(A) - L_{m,n}(A)$, as well as the measured error from the "accurate" solution computed by the MATLAB Symbolic Computation toolbox. (We use `sym` when $\exp(A)$ is known exactly or N is small, and use `vpa` with 250 digits for large matrices.) Algorithm 5.1 successfully solves all these problems to desired accuracy. Algorithm 5.5 also solves all problems except for Example 5.2. Despite that the measured error in Example 5.2 is already smaller than τ, we regard it as a failure since the algorithm cannot identify the true error by itself. However, the algorithm still provides a useful error estimate for the unsatisfactory solution which informs the user about the quality of the solution. The difference between the a posteriori error estimate and the measured error also confirms that the weighted sum $E_{m,n}(A)$ can be more accurate than the upper/lower bounds. As a general purpose solver, MATLAB's `expm`

6. CONTEST: http://www.mathstat.strath.ac.uk/research/groups/numerical_analysis/contest .

function only produces componentwise accuracy for two of these examples. In fact, for the first three examples which have large normwise condition numbers (see (4.1.9) for the definition of $\kappa_{\exp}(A)$, here the F-norm is used), the normwise errors of `expm` are also large (1.0×10^{-2}, 1.0×10^{0}, and 1.0×10^{0}, respectively).

Table 5.2 – Componentwise relative errors of the test examples.

Matrix ID	$C(A)$	$\kappa_{\exp}(A)$	τ	Algorithm 5.1	Algorithm 5.5		`expm`
					estimated	measured	
5.1	1.0×10^{0}	1.7×10^{29}	4.5×10^{-13}	8.9×10^{-16}	1.2×10^{-14}	6.3×10^{-15}	1.0×10^{-2}
5.2	8.6×10^{1}	2.4×10^{19}	6.8×10^{-13}	4.7×10^{-14}	2.5×10^{-12}	6.0×10^{-13}	1.0×10^{0}
5.3	1.8×10^{1}	5.2×10^{69}	9.1×10^{-13}	5.7×10^{-15}	3.8×10^{-13}	1.5×10^{-13}	1.0×10^{0}
5.4	9.1×10^{0}	1.7×10^{0}	2.3×10^{-12}	3.6×10^{-16}	1.1×10^{-12}	6.8×10^{-14}	1.8×10^{-14}
5.5	5.1×10^{1}	5.4×10^{0}	1.1×10^{-11}	2.0×10^{-14}	2.9×10^{-12}	2.2×10^{-12}	2.6×10^{6}
5.6	1.3×10^{2}	1.8×10^{0}	2.9×10^{-11}	9.8×10^{-13}	1.5×10^{-11}	6.9×10^{-14}	9.7×10^{77}
5.7	2.0×10^{2}	7.3×10^{0}	4.5×10^{-11}	2.8×10^{-14}	2.2×10^{-11}	1.9×10^{-11}	3.3×10^{7}
5.8	1.6×10^{3}	6.1×10^{1}	3.6×10^{-10}	6.3×10^{-13}	1.2×10^{-10}	1.0×10^{-10}	4.2×10^{6}
5.9	2.0×10^{3}	2.4×10^{4}	4.6×10^{-10}	6.1×10^{-11}	3.5×10^{-10}	8.4×10^{-12}	1.1×10^{-12}

Table 5.3 – Parameter settings for Algorithm 5.1 and Algorithm 5.5.

Matrix ID	Algorithm 5.1		Algorithm 5.5		
	m	$\log_2 n$	m	$\log_2 n$	`#(iter)`
5.1	12	1	8	3	1
5.2	18	6	8	10	3
5.3	21	3	8	8	2
5.4	21	2	8	5	2
5.5	18	5	8	9	2
5.6	19	6	8	10	2
5.7	18	7	8	9	1
5.8	18	10	8	12	1
5.9	19	10	8	14	2

Table 5.4 – Execution time (in seconds) of the test examples.

Matrix ID	Algorithm 5.1	Algorithm 5.5	`expm`
5.1	8.4×10^{-5}	1.5×10^{-5}	5.4×10^{-4}
5.2	8.9×10^{-5}	4.7×10^{-5}	3.7×10^{-4}
5.3	9.4×10^{-5}	2.9×10^{-5}	6.0×10^{-4}
5.4	8.3×10^{-5}	3.2×10^{-5}	2.5×10^{-3}
5.5	4.2×10^{-4}	1.2×10^{-3}	5.5×10^{-4}
5.6	3.7×10^{-3}	2.0×10^{-2}	3.6×10^{-3}
5.7	1.3×10^{-2}	2.4×10^{-2}	7.2×10^{-3}
5.8	8.6×10^{0}	3.5×10^{1}	3.1×10^{0}
5.9	1.4×10^{1}	1.1×10^{2}	2.4×10^{1}

Example 5.6 is of particular interest. The exact solution is known to be

$$\big[\exp(A)\big]_{ij} = \begin{cases} 0, & i > j, \\ \frac{1}{(j-i)!}, & i \le j, \end{cases}$$

whose nonzero entries are the coefficients in the Maclaurin series of $\exp(x)$. Algorithm 5.5 terminates successfully with $n = 1024$. Therefore the first 128 terms in the Maclaurin series of $T_8\big(x/1024\big)^{1024}$ and $\widetilde{T}_8\big(x/1024\big)^{1024}$ both agree with those terms in $\exp(x)$ within a relative error of 2×10^{-11}. Since $1/128! < 2.6 \times 10^{-216}$, the remaining terms are very tiny. This is an intuitive way to understand why our methods can always produce high relative accuracy. Example 5.9 is a very challenging problem because the magnitude of the nonzero entries in the solution vary from 10^{-304} to 10^{302}. Our algorithms still provide componentwise accurate solutions for this extreme case. Interestingly, `expm` also does a good job for this difficult example but fails for Example 5.6 which should be easier. A brief explanation is that `expm` tries to avoid "unnecessary" scaling in Example 5.6 and thus loses the opportunity to recover small entries in the solution.

Table 5.3 lists the parameters used in Algorithm 5.1 and Algorithm 5.5, respectively. For Examples 5.3, 5.7, and 5.8, the parameter settings used in Algorithm 5.5 would be treated as infeasible in Algorithm 5.1, since they cannot pass the a priori test (5.3.1). Therefore, Theorem 5.2.4 indeed overestimates the truncation errors. Interestingly, because of the overestimation, the parameter setting used in Algorithm 5.5 requires less computational work compared to the "optimal" settings in Algorithm 5.1 for Examples 5.7 and 5.8. Although Algorithm 5.5 is still more expensive since the upper bound $U_{m,n}(A)$ is also calculated, Algorithm 5.3 with the same setting might be potentially cheaper than Algorithm 5.1. These phenomena confirm our motivation for developing iterative methods based on a posteriori error estimates. Algorithm 5.5 uses at most two iterations in almost all examples since the estimated scale factor n based on the convergence rate is quite accurate. Example 5.2 requires three iterations because it is terminated in the third step where the error stops decreasing.

The computational times are provided in Table 5.4 for reference also. Algorithm 5.1 and MATLAB's `expm` have similar performance since they both use a priori estimates to determine the approximation order and the scale factor. As an iterative algorithm, with both upper and lower bounds computed, Algorithm 5.5 is in general slower. This is the cost we need to pay in order to obtain extra robustness.

***Remark* 5.5.1.** In practice, $m = 13$ is recommended as the truncation order for double precision floating-point numbers. Algorithm 5.5 can solve all our examples with this recommended setting. Most of them are solved in one iteration and hence the performance is also largely improved compared to $m = 8$. But the convergence rate can be better understood with a smaller truncation order since we need to predict the scale factor from an improper initial guess. A failure in the testing examples with $m = 8$ also demonstrates the robustness of Algorithm 5.5.

Finally, we provide an example which illustrates the limitation of our methods.

Example 5.10. The discretization of a 1-D Laplacian operator

$$A = -(N+1)^2 T_N = (N+1)^2 \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}_{N\times N},$$

where $N = 50$, 100, 200, and 400.

These matrices satisfy $|s(A)| = 2(N+1)^2$ and $C(A) \approx N - 1 + 2(N+1)^2$. They are considered too difficult for our methods since $C(A)$ is much larger compared to our assumption $C(A) = \mathcal{O}(N)$, see Section 5.1. Consequently, the relative errors of our algorithms are often larger than τ, especially when N is large, see Table 5.5. Only slight improvements in the accuracy have been observed by switching to the recommended truncation order $m = 13$. Not surprisingly, when increasing N, the accuracy of our methods decrease as $C(A)$ grows quickly. However, MATLAB's `expm` works well for this matrix although $\kappa_{\exp}(A)$ is also large. As the entries of $\exp(A)$ do not vary too much for these matrices (from 10^{-11} to 10^{-6}), `expm` also provides small componentwise errors.

Table 5.5 – Componentwise relative errors in Example 5.10.

N	$C(A)$	$\kappa_{\exp}(A)$	τ	Algorithm 5.1	Algorithm 5.5		`expm`
					estimated	measured	
50	5.2×10^{3}	4.5×10^{4}	1.1×10^{-11}	2.8×10^{-12}	9.5×10^{-11}	6.5×10^{-11}	8.4×10^{-13}
100	2.1×10^{4}	2.5×10^{5}	2.3×10^{-11}	1.1×10^{-11}	2.0×10^{-10}	1.3×10^{-10}	2.5×10^{-12}
200	8.1×10^{4}	1.4×10^{6}	4.5×10^{-11}	4.2×10^{-10}	1.3×10^{-9}	9.5×10^{-10}	1.3×10^{-11}
400	3.2×10^{5}	7.9×10^{6}	9.1×10^{-11}	1.3×10^{-10}	4.0×10^{-9}	3.0×10^{-9}	2.1×10^{-11}

5.6 Summary

In this chapter we have presented the aggressively truncated Taylor series method for calculating the exponential of an essentially nonnegative matrix. Truncated Taylor series methods are usually not the first choice for general matrices. However, by carefully choosing the order of truncation and the scale factor, they are preferred for essentially nonnegative matrices. We make use of the nonnegativity and establish componentwise a priori and a posteriori error estimates for both $L_{m,n}(A)$ and $U_{m,n}(A)$. We also show that the weighted average $E_{m,n}(A)$ offers a higher order approximation. These novel theoretical results lead to different variants of the aggressively truncated Taylor series method. Rounding error analysis ensures the stability of our algorithms. The interval algorithm also provides componentwise error estimates regardless of roundoff. Finally we test these algorithms with some commonly used matrices to confirm the efficiency and accuracy.

6 Finite Section Method

In this chapter, we consider another matrix exponential problem. In a number of scientific applications, especially in quantum mechanics, it is desirable to compute $\exp(\mathrm{i}A)$ where A is a self-adjoint operator. For example, $\exp(\mathrm{i}A)$ naturally appears in the solution of the time-dependent Schrödinger equation [35]. We refer to, e.g., [39] for applications from other domains. In practice, the operator A is often given in discretized form, i.e., a doubly-infinite Hermitian matrix under a certain basis, and a finite diagonal block of $\exp(\mathrm{i}A)$ is of interest. Throughout the chapter, we assume that A is self-adjoint and narrow banded. Suppose the $(-m:m,-m:m)$ block of $\exp(\mathrm{i}A)$ is desired. A simple way to solve this problem is illustrated in Figure 6.1. We first compute the exponential of the $(-w:w,-w:w)$ block of A, where w is chosen somewhat larger than m, and then use its central $(2m+1)\times(2m+1)$ diagonal block to approximate the desired solution. In reference to similar methods for solving linear systems [65, 102], we call this approach *finite section method*. The diagonal blocks $(-m:m,-m:m)$ and $(-w:w,-w:w)$ are called the *desired window* and the *computational window*, respectively.

Despite the simplicity of the finite section method, it is crucial to understand how large the computational window needs to be, and whether this truncation produces a sufficiently accurate approximation to the true solution. These questions are relatively easy to answer for bounded matrices, where standard polynomial approximation technique can be applied. But it turns out that the finite section method can also be applied to certain unbounded matrices, and still produces reliable solutions. For example, Figure 6.1 illustrates this for an unbounded Wilkinson-type matrix $W^{-}(1)$, see Section 6.3.1, for which the error decays quickly when the size of the computational window increases. In this chapter, we will explain this phenomenon and establish the finite section method with error estimates for several classes of doubly-infinite Hermitian matrices. Unlike the setting in Chapter 5, we are only interested in normwise accuracy in this chapter. Tiny entries in $\exp(\mathrm{i}A)$ are still taken into account—we make use of these tiny entries to derive the finite section method.

To our knowledge, much of the existing literature on infinite matrices is concerned with solving infinite dimensional linear systems, see e.g., [23, 28, 135] and the references therein.

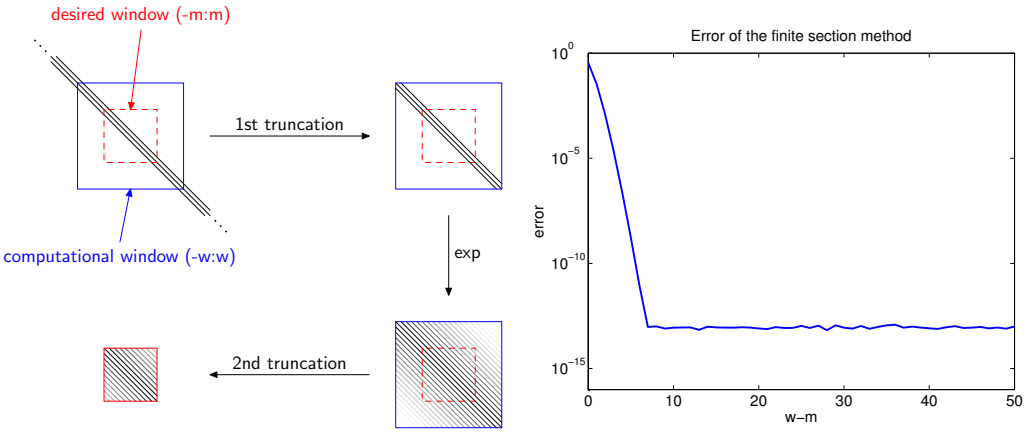


Figure 6.1 – A pictorial illustration of the finite section method. In this case A is the Wilkinson-type matrix $W^-(1)$.

The matrix exponential problem for infinite matrices has also been studied in the literature of exponential integrators, see e.g., [75]. Recently [63] discussed more general functions on operators. As the conjugate transpose operation $A \mapsto A^*$ is naturally an involution, the set of bounded matrices on a Hilbert space form a unital C^*-algebra [44]. Therefore, some existing literature [19, 23, 98] makes use of properties of C^*-algebras to discuss the decay property of matrix inversions, matrix functions, and matrix factorizations. Since we need to handle some unbounded matrices, which do not necessarily form a set closed under multiplication, we will use some techniques in approximation theory and linear algebra to discuss the finite section method for doubly-infinite matrices.

This chapter is largely based on the manuscript [133] submitted for publication in *Linear Algebra Appl.*. The rest of this chapter is organized as follows. In Section 6.2, we discuss the decay property of $\exp(\mathrm{i}A)$ for a bounded Hermitian matrix A and illustrate how this can be used to analyze the finite section method. In Section 6.3, we first analyze decay of entries for Wilkinson-type matrices and derive the corresponding finite section method, and then discuss some extensions to more general unbounded matrices. Finally, numerical experiments are presented in Section 6.4 to demonstrate the reliability of finite section methods.

6.1 Exponentials of doubly-infinite matrices

To set up the stage, let us first provide a formal mathematical formulation of the problem. In the following, we recall some preliminaries in functional analysis. These results can be found in, e.g., [4, 36, 122, 123].

A doubly-infinite matrix is a two dimensional array $A = [a_{ij}]$ of complex numbers with $i, j \in \mathbb{Z}$. It is called Hermitian (or skew-Hermitian) if $a_{ij} = a_{ji}^*$ (or $a_{ij} = -a_{ji}^*$) for all $i, j \in \mathbb{Z}$. If there exists an even number b such that $a_{ij} = 0$ when $|i - j| > b/2$, then A is called *b-banded*, or *banded* in short. Matrix-matrix and matrix-vector multiplications are defined akin to those

operations for finite matrices. That is,

$$\begin{aligned}(AB)_{ij} &= \sum_{k\in\mathbb{Z}} a_{ik}b_{kj},\\ (Ax)_i &= \sum_{k\in\mathbb{Z}} a_{ik}x_k,\end{aligned} \tag{6.1.1}$$

where A, B are doubly-infinite matrices and $x = [x_i]$ is a doubly-infinite vector, provided that these summations converge absolutely. Evidently, multiplications involving banded matrices are always well-defined since all summations are finite. A doubly-infinite matrix A is called *bounded* if

$$\|A\|_2 := \sup_{\substack{x\in l^2(\mathbb{Z})\\ \|x\|_2\le 1}} \|Ax\|_2 < +\infty;$$

in this case A can be interpreted as a continuous linear operator over $l^2(\mathbb{Z})$ [4]. By Gelfand's formula [53], we have

$$\rho(A) = \lim_{k\to\infty} \|A^k\|_2^{\frac{1}{k}} \le \|A\|_2,$$

indicating that the spectrum $\Lambda(A)$ is also bounded. In analogy to (4.1.1) for finite matrices, for an analytic function $F(z)$ defined on a domain Ω enclosing $\Lambda(A)$, the matrix function $F(A)$ is defined as

$$F(A) = \frac{1}{2\pi\mathrm{i}} \oint_{\partial\Omega} F(z)(zI - A)^{-1}\,\mathrm{d}z.$$

In this chapter we are interested in the exponential function $F(z) = \exp(\mathrm{i}z)$.

For unbounded Hermitian matrices, the self-adjointness is important even for defining the matrix exponential. Since unbounded Hermitian matrices do not necessarily represent self-adjoint operators on $l^2(\mathbb{Z})$, careful treatment is required. In this chapter we only consider the class of doubly-infinite banded Hermitian matrices A that can be expressed as the sum of three Hermitian matrices

$$A = D + G + R,$$

where D is diagonal and invertible, R is bounded, and $\|GD^{-1}\|_2 < 1$. We will show that A is self-adjoint, in the sense that it can be represented as a self-adjoint operator by choosing a suitable domain of definition in $l^2(\mathbb{Z})$. Let

$$\mathscr{D}_A = \left\{ x \in l^2(\mathbb{Z}) \colon \sum_{k\in\mathbb{Z}} |d_{kk}x_k|^2 < +\infty \right\},$$

which is a dense subspace of $l^2(\mathbb{Z})$. Then A defines a symmetric operator $\phi[A] \colon \mathscr{D}_A \to l^2(\mathbb{Z})$,

$x \mapsto Ax$, by the matrix-vector multiplication (6.1.1). Notice that

$$\|(A-D)x\|_2 \le \|(GD)^{-1}(Dx)\|_2 + \|Rx\|_2 \le \|(GD)^{-1}\|_2\|Dx\|_2 + \|Rx\|_2 < \|Dx\|_2 + \|R\|_2\|x\|_2$$

for all $x \in \mathscr{D}_A$ Then by the Kato-Rellich theorem [93, Theorem 4.4 in Chapter 5], $\phi[A]$ is essentially self-adjoint due to the fact that $\phi[D]$, which is defined on $\mathscr{D}_A$, is essentially self-adjoint [4]. By the spectral decomposition of the closure of $\phi[A]$,

$$\overline{\phi[A]} = \int_{-\infty}^{+\infty} \lambda \,\mathrm{d}P_\lambda,$$

we define $\exp\big(\mathrm{i}t\overline{\phi[A]}\big)$ as [122, Theorem VIII.6]

$$\exp\big(\mathrm{i}t\overline{\phi[A]}\big) = \int_{-\infty}^{+\infty} \exp(\mathrm{i}t\lambda)\,\mathrm{d}P_\lambda, \qquad t \in \mathbb{R},$$

where P is a projection-valued measure on $l^2(\mathbb{Z})$. Since $\exp\big(\mathrm{i}t\overline{\phi[A]}\big)$ is unitary in $l^2(\mathbb{Z})$ and hence bounded, it has a unique matrix representation [4] with respect to the standard basis $\{e_n\}_{n\in\mathbb{Z}}$. This matrix representation is denoted by $\exp(\mathrm{i}tA)$.

The finite section method can be interpreted as extracting a diagonal block from a certain block diagonal approximation of the matrix. Hence a perturbation analysis of the matrix exponential as in [84, 151] is helpful for studying the truncation error in this method. Fortunately, the perturbation results mentioned in Section 4.1 carry over to doubly-infinite matrices. The solution of the linear differential equation

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \mathrm{i}Ax(t), \qquad x(t) \in l^2(\mathbb{Z})$$

is given by [36, Theorem 2.1.10]

$$x(t) = \exp(\mathrm{i}tA)x(0).$$

Similar to (4.1.8), using this connection between the linear differential equation and the exponential function, it can be shown [36, Theorem 3.2.1] that

$$\exp[\mathrm{i}t(A+\Delta A)]v - \exp(\mathrm{i}tA)v = \mathrm{i}\int_0^t \exp[\mathrm{i}(t-s)A]\Delta A \exp[\mathrm{i}s(A+\Delta A)]v\,\mathrm{d}s, \qquad \forall v \in l^2(\mathbb{Z}), \quad (6.1.2)$$

when both A and ΔA are self-adjoint, and in addition, ΔA is bounded. A simple consequence of (6.1.2) is that

$$\big\|\exp[\mathrm{i}t(A+\Delta A)]v - \exp(\mathrm{i}tA)v\big\|_2 \le \|\Delta A\|_2\|v\|_2,$$

indicating that the exponential of a skew-Hermitian matrix is always well-conditioned in terms of absolute error. Since $\big\|\exp(\mathrm{i}A)\big\|_2 = 1$, absolute accuracy is sufficient in our problem. We will see in Sections 6.2 and 6.3 that the perturbation result (6.1.2) plays an important role

when analyzing the error of the finite section method.

6.2 The finite section method for bounded matrices

To analyze the accuracy of the finite section method, we start by discussing a relatively simple case—A is a bounded Hermitian matrix. Throughout this section, we assume that the doubly-infinite matrix A is Hermitian and b-banded. In the following, we first recall the exponential decay property of $\exp(\mathrm{i}A)$ and then establish finite section methods based on this property.

6.2.1 The exponential decay property

It is well-known that when B is a finite banded matrix, the entries of $F(B)$ decay exponentially from the diagonal where F is an analytic function defined on a domain $\Omega \supset \Lambda(B)$. This property has been observed for the inverse of a certain banded matrix in [40, 41]. Then the decay property for general matrix functions has been developed [20], and even extends to general sparse matrices [21] and infinite matrices [23, 98]. In the following we briefly recall the results in [20, Section 2], on which our analyses are built on. We remark that the decay properties easily carry over to functions of bounded doubly-infinite matrices.

Definition 6.2.1. *We say that a matrix $A = [a_{ij}]$ has the* (K, ρ)-exponential decay property*, or* exponential decay property *in short, if there exist $K > 0$ and $\rho \in (0,1)$ such that*

$$\left|a_{ij}\right| \le K\rho^{-|i-j|}, \qquad \forall\, i,\, j. \tag{6.2.1}$$

The constant ρ is called the decay rate*. If for any $\rho \in (0,1)$ there exists a positive number K such that* (6.2.1) *holds for all i, j, we say that A decays* super-exponentially.

Evidently, all finite matrices trivially have the exponential decay property by choosing sufficiently large K. Hence this property is meaningful only when K can be chosen moderately small and ρ is not too close to one. However, for a doubly-infinite matrix A, the exponential decay property is nontrivial since it implies the boundedness of A in $l^2(\mathbb{Z})$. In fact, both $\|A\|_1 := \sup_j \sum_{i\in\mathbb{Z}} |a_{ij}|$ and $\|A\|_\infty := \sup_i \sum_{j\in\mathbb{Z}} |a_{ij}|$ are bounded by $K(1+\rho)(1-\rho)^{-1}$ and hence the Schur test [131] implies

$$\|A\|_2 \le \sqrt{\|A\|_1 \|A\|_\infty} \le K(1+\rho)(1-\rho)^{-1}.$$

Another closely related concept is *localization* [153]. Loosely speaking, a matrix is *localized* if it becomes sparse after dropping all small entries whose magnitudes are below a certain threshold. In this sense, the exponential decay property naturally indicates localization.

Proofs of the exponential decay properties of certain matrix functions are usually built on polynomial approximation (see, e.g., [20, 21, 41]), i.e., approximating $F(B)$ by a matrix

polynomial $p(B)$ where $p \in \mathbb{C}_k[x]$ is a polynomial of degree at most k. The propositions below will be used in our analyses.

Lemma 6.2.2 (Bernstein [20, 104])**.** *Let $\mathscr{E}_\chi$ ($\chi > 1$) be the* Bernstein ellipse *in the complex plane defined by*

$$\frac{\Re(z)^2}{\left(\chi+\chi^{-1}\right)^2} + \frac{\Im(z)^2}{\left(\chi-\chi^{-1}\right)^2} = \frac{1}{4}.$$

Then for any function F being analytic in the domain enclosed by $\mathscr{E}_\chi$ and continuous on $\mathscr{E}_\chi$, we have

$$\inf_{p\in\mathbb{C}_k[x]} \|F-p\|_\infty \leq \frac{2M(\chi)}{\chi^k(\chi-1)}, \qquad (k\in\mathbb{N})$$

where

$$M(\chi) = \max_{z\in\mathscr{E}_\chi} |F(z)|.$$

Proof. See [104, Theorem 7 in Chapter 5]. □

Theorem 6.2.3 (Benzi-Golub [20])**.** *Let B be a b-banded Hermitian matrix whose spectrum is contained in the interval $[-1,1]$. Then for any function F being analytic*[1] *inside the Bernstein ellipse $\mathscr{E}_\chi$ and continuous on $\mathscr{E}_\chi$, there exist constants $K > 0$ and $\rho \in (0,1)$ such that*

$$\left|[F(B)]_{ij}\right| \leq K\rho^{|i-j|}.$$

More precisely, these constants can be chosen as

$$K = \max\left\{\frac{2\chi M(\chi)}{\chi-1}, \|F(B)\|_2\right\} \quad \textit{and} \quad \rho = \chi^{-\frac{2}{b}}.$$

Proof. See [20, Theorem 2.2]. □

Although Theorem 6.2.3 is derived only for finite matrices in [20], the same proof is valid for analytic functions of a doubly-infinite Hermitian b-banded matrix B as long as $\Lambda(B) \subset [-1,1]$ holds. Now we consider a doubly-infinite b-banded Hermitian matrix A with spectrum $\Lambda(A) \subset [\lambda_0-\delta, \lambda_0+\delta]$.[2] In the following we show that $\exp(\mathrm{i}A)$ has the super-exponential decay property.

1. In [20], F is additionally required to be real analytic. But this extra assumption turns out to be unnecessary, see, e.g., [104, page 76].
2. Without loss of generality, we always assume that $\delta > 0$.

Corollary 6.2.4. *Suppose A is a doubly-infinite b-banded Hermitian matrix with $\Lambda(A) \subset [\lambda_0-\delta, \lambda_0+\delta]$. For any $\chi > 1$, let*

$$K = \frac{2\chi}{\chi-1}\exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big] \quad \textit{and} \quad \rho = \chi^{-\frac{2}{b}}. \tag{6.2.2}$$

Then

$$\Big|\big[\exp(\mathrm{i}A)\big]_{ij}\Big| \le K\rho^{|i-j|}, \qquad \forall\, i,\, j \in \mathbb{Z}. \tag{6.2.3}$$

Moreover, $\exp(\mathrm{i}A)$ *has the super-exponential decay property.*

Proof. For $\chi > 1$, we set

$$M(\chi) = \max_{z\in\mathcal{E}_\chi}\big|\exp(\mathrm{i}\delta z)\big| = \max_{\substack{x+\mathrm{i}y\in\mathcal{E}_\chi \\ x,y\in\mathbb{R}}}\big|\exp[\mathrm{i}\delta(x+\mathrm{i}y)]\big| = \max_{\substack{x+\mathrm{i}y\in\mathcal{E}_\chi \\ x,y\in\mathbb{R}}}\big|\exp[-\delta y]\big|$$
$$= \max_{y\in\left[-\frac{\chi-\chi^{-1}}{2},\frac{\chi-\chi^{-1}}{2}\right]}\big|\exp(-\delta y)\big| = \exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big].$$

Applying Theorem 6.2.3 to $F(z) = \exp(\mathrm{i}\delta z)$ with $B = (A-\lambda_0 I)/\delta$, which has spectrum contained in $[-1,1]$, yields

$$\Big|\big(\exp\big[\mathrm{i}(A-\lambda_0 I)\big]\big)_{ij}\Big| = \big|[F(B)]_{ij}\big| \le K\rho^{|i-j|}, \tag{6.2.4}$$

where $\rho = \chi^{-\frac{2}{b}}$ and

$$K = \max\Big\{\frac{2\chi M(\chi)}{\chi-1}, 1\Big\} = \frac{2\chi M(\chi)}{\chi-1} = \frac{2\chi}{\chi-1}\exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big] > 1.$$

The bound (6.2.3) now follows from (6.2.4) using the fact that

$$\big|\exp(\mathrm{i}A)\big| = \big|\exp(\mathrm{i}\lambda_0)\big|\cdot\big|\exp\big[\mathrm{i}(A-\lambda_0 I)\big]\big| = \big|\exp\big[\mathrm{i}(A-\lambda_0 I)\big]\big|.$$

For any $\rho \in (0,1)$ we choose $\chi = \rho^{-\frac{b}{2}}$ and $K = 2\chi\exp\big[\delta(\chi^2-1)/(2\chi)\big]/(\chi-1)$ according to (6.2.2) so that (6.2.3) holds for all $i,\, j \in \mathbb{Z}$. Therefore, by definition $\exp(\mathrm{i}A)$ has the super-exponential decay property. □

Since $F(z) = \exp(\mathrm{i}\delta z)$ is an entire function, in Corollary 6.2.4 we can choose any χ from the interval $(1,+\infty)$ to bound $\big|[\exp(\mathrm{i}A)]_{ij}\big|$. Sometimes an upper bound of $\big|[\exp(\mathrm{i}A)]_{ij}\big|$ for a given entry (i,j) is of interest. Thus it is desirable to find a χ that minimizes $K\rho^{|i-j|}$. Such a choice is made by taking $\theta = 1$ and $d = |i-j|$ in the following theorem.

Theorem 6.2.5. *Let b, d, δ, and θ be positive numbers. Then the function*

$$g(\chi) = \Big(\frac{2\chi}{\chi-1}\Big)^{\theta}\exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big]\chi^{-\frac{2d}{b}}, \qquad (\chi > 1)$$

has a unique minimum at $\chi = \chi_*$, where χ_* is the unique root of the cubic equation

$$\chi^3 - \Big(1 + \frac{4d}{b\delta}\Big)\chi^2 + \Big(1 + \frac{4d - 2b\theta}{b\delta}\Big)\chi - 1 = 0$$

in the interval $(1, +\infty)$. *Moreover, we have*

$$\lim_{d\to+\infty} \frac{\chi_*}{d} = \frac{4}{b\delta}.$$

Proof. We first notice that

$$\lim_{\chi\to 1+} g(\chi) = \lim_{\chi\to+\infty} g(\chi) = +\infty.$$

Therefore $g(\chi)$ has at least one minimum in the interval $(1, +\infty)$ as $g(\chi)$ is continuously differentiable. Any minimizer χ_* satisfies the condition

$$0 = \frac{\mathrm{d}g(\chi)}{\mathrm{d}\chi}\bigg|_{\chi=\chi_*} = g(\chi_*)\Big[-\frac{\theta}{\chi_*(\chi_*-1)} + \frac{\delta(\chi_*^2+1)}{2\chi_*^2} - \frac{2d}{b\chi_*}\Big].$$

Multiplying by $2\chi_*^2(\chi_*-1)/[g(\chi_*)\delta]$ yields $h(\chi_*) = 0$, where

$$h(\chi) = \chi^3 - \Big(1 + \frac{4d}{b\delta}\Big)\chi^2 + \Big(1 + \frac{4d - 2b\theta}{b\delta}\Big)\chi - 1.$$

Since $h(1) < 0$, $h(\chi)$ has either one root or three roots in the interval $(1, +\infty)$ according to the monotonicity of a real cubic function. If there are three roots in $(1, +\infty)$, the product of all these roots is greater than one. This contradicts Vieta's formulas which imply that the product of the roots of $h(\chi)$ is one. Therefore, $h(\chi)$ has only one root in $(1, +\infty)$. This root is also the unique minimizer of $g(\chi)$.

To obtain the asymptotic behavior of χ_*, we consider the function

$$\tilde{h}(\tilde{\chi}) = \frac{h(\chi)}{d^3} = \tilde{\chi}^3 - \Big(\frac{1}{d} + \frac{4}{b\delta}\Big)\tilde{\chi}^2 + \Big(\frac{1}{d^2} + \frac{4}{bd\delta} - \frac{2\theta}{d^2\delta}\Big)\tilde{\chi} - \frac{1}{d^3},$$

where $\tilde{\chi} = \chi/d$. Since χ_* is the largest real root of $h(\chi)$, $\tilde{\chi}_* = \chi_*/d$ is also the largest real root of $\tilde{h}(\tilde{\chi})$. When $d = +\infty$, we have $\tilde{\chi}_* = 4/(b\delta)$. Notice that $\tilde{\chi}_*$ is a continuous function of the coefficients of $\tilde{h}(\tilde{\chi})$ (see, e.g., [42, 163]). Therefore, we obtain

$$\lim_{d\to+\infty} \frac{\chi_*}{d} = \lim_{d\to+\infty} \tilde{\chi}_* = \frac{4}{b\delta}.$$ □

6.2.2 The finite section method

We have seen that $\exp(\mathrm{i}tA)$ has the (super-)exponential decay property when a doubly-infinite matrix A is banded Hermitian and bounded. We now make use of this property to establish the finite section method with guaranteed accuracy.

Suppose A is partitioned into the form

$$A = \begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{bmatrix},$$

where A_{22} corresponds to the computational window. The finite section method extracts the desired window from $\exp(\mathrm{i}A_{22})$. The matrix $\exp(\mathrm{i}A_{22})$ is the central diagonal block in $\exp\big(\mathrm{i}\,\mathrm{Diag}\{A_{11}, A_{22}, A_{33}\}\big)$, and the latter can be viewed as the exact exponential of a perturbed matrix. Hence the perturbation analysis (6.1.2) is helpful for studying the truncation error in the finite section method. The following theorem is based on this perturbation analysis.

Theorem 6.2.6. *Suppose that*

$$A = \begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{bmatrix}$$

is a doubly-infinite b-banded Hermitian matrix with $\Lambda(A) \subset [\lambda_0 - \delta, \lambda_0 + \delta]$, where A_{22} is the $(-w:w, -w:w)$ diagonal block of A. Let $\tilde{A} = \mathrm{Diag}\{A_{11}, A_{22}, A_{33}\}$ be a block diagonal approximation of A. Then for any $\chi > 1$, we have

$$\Big|\big[\exp(\mathrm{i}A) - \exp(\mathrm{i}\tilde{A})\big]_{ij}\Big| \le K\big(\rho^{|w-i|+|w-j|-\frac{b}{2}} + \rho^{|w+i|+|w+j|-\frac{b}{2}}\big), \qquad \forall\, i,\ j$$

where

$$K = \frac{b(b+2)}{4} \max\big\{\|A_{12}\|_2, \|A_{23}\|_2\big\}\Big(\frac{2\chi}{\chi-1}\Big)^2 \exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big] \quad \textit{and} \quad \rho = \chi^{-\frac{2}{b}}. \tag{6.2.5}$$

Proof. Let

$$A_0 = \begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & 0 \\ & 0 & A_{33} \end{bmatrix}, \qquad B = A_0 - A.$$

Then $\Lambda(A_0) \subset [\lambda_0 - \delta, \lambda_0 + \delta]$ because

$$\rho(A_0 - \lambda_0 I) = \|A_0 - \lambda_0 I\|_2 = \max\left\{\left\|\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} - \lambda_0 I\right\|_2, \|A_{33} - \lambda_0 I\|_2\right\} \le \|A - \lambda_0 I\|_2 = \rho(A - \lambda_0 I).$$

Similarly, it can be shown that $\Lambda(\tilde{A}) \subset [\lambda_0 - \delta, \lambda_0 + \delta]$. Using (6.1.2), we obtain

$$\exp(\mathrm{i}A_0)e_j - \exp(\mathrm{i}A)e_j = \mathrm{i}\int_0^1 \exp[\mathrm{i}(1-s)A]B\exp(\mathrm{i}sA_0)e_j\,\mathrm{d}s, \qquad \forall\, j \in \mathbb{Z}.$$

For each $s \in [0,1]$, we denote by $U(s) = \exp[\mathrm{i}(1-s)A]$ and $V(s) = \exp(\mathrm{i}sA_0)$. Then for $i,\ j \in \mathbb{Z}$,

we conclude from the nonzero structure of B that

$$\begin{aligned}
&\Big|\big[U(s)BV(s)\big]_{ij}\Big| \\
&= \left|\sum_{k=1}^{b/2}\sum_{\ell=-b/2+k}^{0} b_{w+k,w+\ell}U_{i,w+k}V_{w+\ell,j} + \sum_{k=-b/2+1}^{0}\sum_{\ell=1}^{b/2+k} b_{w+k,w+\ell}U_{i,w+k}V_{w+\ell,j}\right| \\
&\le \|A_{23}\|_2\left(\sum_{k=1}^{b/2}\sum_{\ell=-b/2+k}^{0}\big|U_{i,w+k}V_{w+\ell,j}\big| + \sum_{k=-b/2+1}^{0}\sum_{\ell=1}^{b/2+k}\big|U_{i,w+k}V_{w+\ell,j}\big|\right).
\end{aligned}$$

Using (6.2.2) and (6.2.3), we have

$$\begin{aligned}
\big|U_{i,w+k}\big| &\le \frac{2\chi}{\chi-1}\exp\Big[\frac{(1-s)\delta(\chi^2-1)}{2\chi}\Big]\rho^{|w+k-i|}, \\
\big|V_{w+\ell,j}\big| &\le \frac{2\chi}{\chi-1}\exp\Big[\frac{s\delta(\chi^2-1)}{2\chi}\Big]\rho^{|w+\ell-j|},
\end{aligned}$$

and thus

$$\big|U_{i,w+k}V_{w+\ell,j}\big| \le K_0\rho^{|w+k-i|+|w+\ell-j|} \le K_0\rho^{|w-i|+|w-j|-|k|-|\ell|} \le K_0\rho^{|w-i|+|w-j|-\frac{b}{2}},$$

where

$$K_0 = \Big(\frac{2\chi}{\chi-1}\Big)^2\exp\Big[\frac{\delta(\chi^2-1)}{2\chi}\Big].$$

Hence, we obtain

$$\Big|\big[U(s)BV(s)\big]_{ij}\Big| = \frac{b(b+2)}{4}\|A_{23}\|_2K_0\rho^{|w-i|+|w-j|-\frac{b}{2}}.$$

Integrating over the interval $[0,1]$ eventually yields

$$\Big|\big[\exp(\mathrm{i}A_0)-\exp(\mathrm{i}A)\big]_{ij}\Big| \le \frac{b(b+2)}{4}\|A_{23}\|_2K_0\rho^{|w-i|+|w-j|-\frac{b}{2}} \le K\rho^{|w-i|+|w-j|-\frac{b}{2}}.$$

Following the same analysis above, we obtain another estimate

$$\Big|\big[\exp(\mathrm{i}\tilde{A})-\exp(\mathrm{i}A_0)\big]_{ij}\Big| \le \frac{b(b+2)}{4}\|A_{12}\|_2K_0\rho^{|w+i|+|w+j|-\frac{b}{2}} \le K\rho^{|w+i|+|w+j|-\frac{b}{2}}.$$

Then the theorem is proved from

$$\big|\exp(\mathrm{i}\tilde{A})-\exp(\mathrm{i}A)\big| \le \big|\exp(\mathrm{i}A_0)-\exp(\mathrm{i}A)\big| + \big|\exp(\mathrm{i}\tilde{A})-\exp(\mathrm{i}A_0)\big|. \qquad \square$$

Now we are ready to derive the finite section method for computing the diagonal block $\big[\exp(\mathrm{i}A)\big]_{(-m:m,-m:m)}$. Based on Theorem 6.2.6, we can choose $w > m$ such that the perturbations introduced at the $\pm w$th rows have only negligible impact on the desired window. For a

given entry (i, j) in the desired window, we set $k = (i + j)/2$. Then

$$\begin{aligned}\left|\left[\exp(\mathrm{i}A) - \exp(\mathrm{i}\tilde{A})\right]_{ij}\right| &\leq K\big(\rho^{|w-i|+|w-j|-\frac{b}{2}} + \rho^{|w+i|+|w+j|-\frac{b}{2}}\big)\\ &= K\big(\rho^{2(w-k)-\frac{b}{2}} + \rho^{2(w+k)-\frac{b}{2}}\big).\end{aligned}$$

Since the function ρ^x is convex, we have

$$\rho^{x_2} + \rho^{x_3} \leq \rho^{x_1} + \rho^{x_4}$$

when $x_1 \leq x_2 \leq x_3 \leq x_4$ and $x_1 + x_4 = x_2 + x_3$. Therefore, taking $x_1 = 2(w-m)$, $x_2 = 2(w-k)$, $x_3 = 2(w+k)$, and $x_4 = 2(w+m)$ yields

$$\begin{aligned}\left|\left[\exp(\mathrm{i}A) - \exp(\mathrm{i}\tilde{A})\right]_{ij}\right| &\leq K\big(\rho^{2(w-k)-\frac{b}{2}} + \rho^{2(w+k)-\frac{b}{2}}\big)\\ &\leq K\big(\rho^{2(w-m)-\frac{b}{2}} + \rho^{2(w+m)-\frac{b}{2}}\big).\end{aligned}$$

It is then desirable to minimize the right-hand-side in the above inequality. Substituting $d = 2(w-m) - b/2$ and $\theta = 2$ into Theorem 6.2.5, we obtain that the parameter χ_* that minimizes $K\rho^{2(w-m)-\frac{b}{2}}$ is the unique root in the interval $(1, +\infty)$ of the cubic equation

$$\chi_*^3 - \Big(1 + \frac{4d}{b\delta}\Big)\chi_*^2 + \Big(1 + \frac{4(d-b)}{b\delta}\Big)\chi_* - 1 = 0. \tag{6.2.6}$$

Such a choice is already sufficient for practical purpose since

$$K\rho^{2(w-m)-\frac{b}{2}} + K\rho^{2(w+m)-\frac{b}{2}} \leq 2K\rho^{2(w-m)-\frac{b}{2}}$$

and usually $\rho^{2(w+m)-\frac{b}{2}} \ll \rho^{2(w-m)-\frac{b}{2}}$. Finally, we remark that the knowledge of the width of $\Lambda(A)$ is required in order to compute the constant K in (6.2.5). In practice a moderate overestimate of the width is sufficient since in Theorem 6.2.6 the closed interval $[\lambda_0 - \delta, \lambda_0 + \delta]$ is only required to contain $\Lambda(A)$. We demonstrate the finite section method in Algorithm 6.1. [3]

Evidently, the effectiveness of Algorithm 6.1 depends on the quality of the a priori estimate. We remark that sometimes the estimate based on Theorem 6.2.6 can severely overestimate the truncation error, mainly because the bound in Theorem 6.2.3 is pessimistic. An extreme case occurs when A is diagonal and has a wide spectrum. Theorem 6.2.3 still provides a very large constant K which grows exponentially with δ while the actual decay is arbitrarily fast. We will show another example in Section 6.3.1. Once the a priori estimate is too pessimistic, it might not be a good idea to identify the size of the computational window based on such a bound. A remedy for this issue will be proposed in the next section.

3. In Step 4 of Algorithm 6.1, we label the indices of E with $-m : m$ instead of $1 : (2m+1)$. We use this labeling convention for submatrices extracted from a doubly-infinite matrix, when there is no ambiguity.

Algorithm 6.1 (A priori) Finite section method for $\exp(\mathrm{i}A)$

Input: A is b-banded Hermitian and bounded, $m \in \mathbb{N}$, $\tau > 0$.

1: Estimate the extreme points of $\Lambda(A)$ and set

$$\delta \leftarrow \frac{\sup \Lambda(A) - \inf \Lambda(A)}{2}.$$

2: Find the smallest integer w such that

$$K\big(\rho^{2(w-m)-1} + \rho^{2(w+m)-1}\big) \le \tau \quad \text{and} \quad w \ge m,$$

where K and ρ are chosen optimally from (6.2.5) and (6.2.6) with $d = 2(w-m) - b/2$.

3: Compute $E \leftarrow \exp\big[\mathrm{i}A_{(-w:w,-w:w)}\big]$.

4: Output $E_{(-m:m,-m:m)}$.

6.3 The finite section method for unbounded matrices

In this section, we discuss how to derive the finite section method for unbounded self-adjoint matrices. The decay property of two classes of Wilkinson-type matrices are analyzed. This analysis is used to derive the finite section method for these matrices. Then we investigate some extensions to a certain class of diagonally dominant banded matrices.

6.3.1 Case study for Wilkinson-type W^- matrices

Our first example is the class of Wilkinson-type W^- matrices

$$W^-(\alpha) = \mathrm{Tridiag}\left\{\begin{array}{ccccccccccc} \cdots & & \alpha & & \cdots & & \alpha & \alpha & \cdots & & \alpha & & \cdots \\ \cdots & & n & & \cdots & & 1 & 0 & -1 & \cdots & -n & & \cdots \\ \cdots & & \alpha^* & & \cdots & & \alpha^* & \alpha^* & \cdots & & \alpha^* & & \cdots \end{array}\right\}.$$

Such matrices have applications in quantum mechanics. Instead of handling the doubly-infinite matrix $W^-(\alpha)$, we start with its central $(2n+1) \times (2n+1)$ diagonal block

$$W_n^-(\alpha) = \mathrm{Tridiag}\left\{\begin{array}{ccccccccc} \alpha & & \cdots & & \alpha & \alpha & \cdots & & \alpha \\ n & & \cdots & & 1 & 0 & -1 & \cdots & -n \\ \alpha^* & & \cdots & & \alpha^* & \alpha^* & \cdots & & \alpha^* \end{array}\right\}.$$

When $\alpha \neq 0$, $W_n^-(\alpha)$ is an irreducible tridiagonal matrix. Consequently all eigenvalues of $W_n^-(\alpha)$ are distinct; the corresponding eigenvectors are unique (up to scaling) [163]. The conclusion also holds when $\alpha = 0$ since $W_n^-(0)$ is diagonal and has distinct eigenvalues. We will show that the eigenvectors are highly localized once $|\alpha| \ll n$. The following lemma is a simplified version of [114, Lemma 4.1] tailored to $W_n^-(\alpha)$. The estimate provided here is slightly better than directly applying the conclusion of [114, Lemma 4.1], since the special structure of $W_n^-(\alpha)$ is taken into account.

Lemma 6.3.1. *Let $\lambda_{-n} \geq \lambda_{-n+1} \geq \cdots \geq \lambda_{n-1} \geq \lambda_n$ be eigenvalues of $W_n^-(\alpha)$, with normalized eigenvectors x_{-n}, x_{-n+1}, …, x_{n-1}, x_n (i.e., $\|x_j\|_2 = 1$). Then the entries of these eigenvectors satisfy*

$$|x_j(i)| \leq \prod_{k=0}^{|i-j|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} \tag{6.3.1}$$

for any integer $d_0 \geq 4|\alpha|$. [4]

Proof. The conclusion obviously holds when $\alpha = 0$ since $W_n^-(0)$ is diagonal. Thus we only consider the case $\alpha \neq 0$. We split $W_n^-(\alpha)$ into $W_n^-(\alpha) = W_n^-(0) + N_n(\alpha)$, where $N_n(\alpha)$ has zeros on its diagonal. Then by Weyl's theorem, we know that $\lambda_j \in \big(-j-2|\alpha|, -j+2|\alpha|\big)$ for all j. We rewrite $\big(W_n^-(\alpha) - \lambda_j I\big)x_j = 0$ as a set of equations:

$$\begin{aligned}
&(n-\lambda_j)x_j(-n) + \alpha x_j(-n+1) = 0,\\
&(n-1-\lambda_j)x_j(-n+1) + \alpha^* x_j(-n) + \alpha x_j(-n+2) = 0,\\
&\cdots\\
&(-k-\lambda_j)x_j(k) + \alpha^* x_j(k-1) + \alpha x_j(k+1) = 0,\\
&\cdots\\
&(-n+1-\lambda_j)x_j(n-1) + \alpha^* x_j(n-2) + \alpha x_j(n) = 0,\\
&(-n-\lambda_j)x_j(n) + \alpha^* x_j(n-1) = 0.
\end{aligned} \tag{6.3.2}$$

Since the eigenvectors are normalized, the conclusion trivially holds when $|i-j| < d_0$. Hereafter we assume that $|i-j| \geq d_0$.

We first consider the case $i \leq j - d_0$. In the following we show by induction that for any integer k satisfying $-n \leq k \leq j - d_0$ we have

$$|x_j(k)| \leq \frac{|\alpha|}{-k+j-3|\alpha|}|x_j(k+1)|. \tag{6.3.3}$$

The first equation in (6.3.2) yields

$$|x_j(-n)| = \frac{|\alpha|}{n-\lambda_j}|x_j(-n+1)| \leq \frac{|\alpha|}{n+j-2|\alpha|}|x_j(-n+1)| \leq \frac{|\alpha|}{n+j-3|\alpha|}|x_j(-n+1)|$$

because $n+j-3|\alpha| \geq d_0 - 3|\alpha| \geq |\alpha| > 0$. Then for $-n < k \leq j - d_0$, we consider the equation

$$(-k-\lambda_j)x_j(k) = -\alpha^* x_j(k-1) - \alpha x_j(k+1).$$

The induction hypothesis implies that $|x_j(k-1)| \leq |x_j(k)|$. Thus we obtain

$$|\alpha x_j(k+1)| \geq |(-k-\lambda_j)x_j(k)| - |\alpha x_j(k-1)| \geq (-k-\lambda_j-|\alpha|)|x_j(k)|.$$

4. To avoid using double subscripts in the notation, here we use $x_j(i)$ to represent the ith entry of x_j.

The corresponding coefficient $-k-\lambda_j-|\alpha|$ is positive because

$$-k-\lambda_j-|\alpha| \geq -k+j-3|\alpha| \geq d_0-3|\alpha| \geq |\alpha| > 0.$$

Hence we obtain

$$|x_j(k)| \leq \frac{|\alpha|}{-k+j-3|\alpha|}|x_j(k+1)|.$$

This finishes the proof of (6.3.3). Applying (6.3.3) repeatedly, we obtain

$$\begin{aligned}
|x_j(i)| &\leq \frac{|\alpha|}{-i+j-3|\alpha|}|x_j(i+1)| \\
&\leq \frac{|\alpha|}{-i+j-3|\alpha|}\cdot\frac{|\alpha|}{-(i+1)+j-3|\alpha|}|x_j(i+2)| \\
&\leq \cdots \\
&\leq |x_j(j-d_0+1)|\prod_{k=i}^{j-d_0}\frac{|\alpha|}{-k+j-3|\alpha|}.
\end{aligned}$$

Taking into account that $|x_j(j-d_0+1)| \leq 1$, we eventually arrive at

$$|x_j(i)| \leq \prod_{k=i}^{j-d_0}\frac{|\alpha|}{-k+j-3|\alpha|} = \prod_{k=0}^{j-i-d_0}\frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{|i-j|-d_0}\frac{|\alpha|}{k+d_0-3|\alpha|},$$

where the first equality follows from a relabelling of the indices ($k \leftarrow j-d_0-k$).

The case $i \geq j+d_0$ can be proved in the same way by starting from the last equation in (6.3.2). In the following we provide an alternative prove by reducing the case $i \geq j+d_0$ to the case $i \leq j-d_0$ which has already been proved. Let $X_n = [x_{-n},\ldots,x_n]$ and $\Lambda = \mathrm{Diag}\{\lambda_{-n},\ldots,\lambda_n\}$. By introducing

$$\Pi = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}$$

we derive from $W_n^-(\alpha)X_n = X_n\Lambda$ that

$$\big[-\Pi W_n^-(\alpha)\Pi\big]\big(\Pi X_n\Pi\big) = \big(\Pi X_n\Pi\big)\big(-\Pi\Lambda\Pi\big),$$

or simply $\big[-\Pi W_n^-(\alpha)\Pi\big]Y_n = Y_n\tilde{\Lambda}$, where $Y_n = \Pi X_n\Pi$, $\tilde{\Lambda} = \Pi(-\Lambda)\Pi$. That is, we transform the spectral decomposition of $W_n^-(\alpha)$ to the spectral decomposition of $W_n^-(-\alpha^*)$ since $-\Pi W_n^-(\alpha)\Pi = W_n^-(-\alpha^*)$. Notice that the diagonal entries of $\tilde{\Lambda}$ are in decreasing order. Then from the part of the conclusion that we have already proved, we conclude that

$$|x_j(i)| = |(X_n)_{ij}| = |(Y_n)_{-i,-j}| \leq \prod_{k=0}^{|i-j|-d_0}\frac{|\alpha|}{k+d_0-3|\alpha|}.$$

when $-i \le -j - d_0$, or equivalently, $i \ge j + d_0$. □

Lemma 6.3.1 demonstrates that the entries $x_j(i)$ decay super-exponentially with respect to $|i-j|$. A more general conclusion for block tridiagonal matrices has been developed in [114], which aims at deriving eigenvalue perturbation bounds, see also [54] and [153]. By choosing $d_0 = \lceil 5|\alpha| \rceil$, we obtain a simpler (but much looser) exponential decay bound

$$|x_j(i)| \le \min\{1, 2^{d_0 - |i-j|}\}. \tag{6.3.4}$$

A notable observation is that neither (6.3.1) nor (6.3.4) depends on the size of $W_n^-(\alpha)$, apart from the fact that these bounds are not useful when $n < 2|\alpha|$. Now from the spectral decomposition

$$W_n^-(\alpha) = \sum_{k=-n}^{n} \lambda_k x_k x_k^*,$$

we immediately obtain that

$$\exp\left[\mathrm{i}\beta W_n^-(\alpha)\right] = \sum_{k=-n}^{n} \exp(\mathrm{i}\beta\lambda_k) x_k x_k^*$$

and hence $\left|\exp\left[\mathrm{i}\beta W_n^-(\alpha)\right]\right| \le |X_n||X_n|^T$ where $X_n = [x_{-n}, \ldots, x_n]$ and $\beta \in \mathbb{R}$. Because the product of two (doubly-infinite) matrices with exponentially decaying off-diagonals also has the exponential decay property (see, e.g., [98]), we conclude that $\exp\left[\mathrm{i}\beta W_n^-(\alpha)\right]$ has the exponential decay property. Lemmas 6.3.2 and 6.3.3 below give quantitative estimates of the decay. We remark that although the exponential decay in eigenvectors implies the exponential decay in the matrix exponential, the opposite is not true. For example, the exponential of the tridiagonal matrix

$$\mathrm{i}\cdot\mathrm{Tridiag}\begin{Bmatrix} -1 & \cdots & -1 \\ 2 & \cdots & \cdots & 2 \\ -1 & \cdots & -1 \end{Bmatrix}$$

has the exponential decay property according to Theorem 6.2.3, while the eigenvectors are not localized.

Lemma 6.3.2. *Suppose two doubly-infinite matrices X and Y both have the exponential decay property, i.e.,*

$$|x_{ij}| \le K_X \rho_X^{|i-j|}, \quad |y_{ij}| \le K_Y \rho_Y^{|i-j|}, \qquad \forall i,\ j.$$

Then their product XY satisfies

$$\left|(XY)_{ij}\right| \le K_X K_Y \left(\frac{2}{1-\rho_0^2} + |i-j| - 1\right)\rho_0^{|i-j|}, \tag{6.3.5}$$

where $\rho_0 = \max\{\rho_X, \rho_Y\}$.

Proof. The entries in the product XY can be bounded by

$$\left|(XY)_{ij}\right| \le \sum_{k\in\mathbb{Z}} |x_{ik}y_{kj}| \le K_X K_Y \sum_{k\in\mathbb{Z}} \rho_0^{|i-k|+|k-j|}.$$

Without loss of generality, we consider the case $i \le j$. Notice that

$$|i-k|+|k-j| = \begin{cases} |i-j|, & \text{if } i \le k \le j, \\ |i-j| + 2\min\{|i-k|, |j-k|\}, & \text{if } k < i \text{ or } k > j. \end{cases}$$

Therefore, we obtain

$$\begin{aligned}
\sum_{k\in\mathbb{Z}} \rho_0^{|i-k|+|k-j|} &= \sum_{k=-\infty}^{i-1} \rho_0^{|i-j|+2|i-k|} + \sum_{k=i}^{j} \rho_0^{|i-j|} + \sum_{k=j+1}^{+\infty} \rho_0^{|i-j|+2|j-k|} \\
&= \Big(2\sum_{k=1}^{+\infty} \rho_0^{2k} + |i-j| + 1\Big)\rho_0^{|i-j|} \\
&= \Big(\frac{2\rho_0^2}{1-\rho_0^2} + |i-j| + 1\Big)\rho_0^{|i-j|} \\
&= \Big(\frac{2}{1-\rho_0^2} + |i-j| - 1\Big)\rho_0^{|i-j|}.
\end{aligned}$$

□

Lemma 6.3.3. *Suppose two doubly-infinite matrices X and Y both have the exponential decay property of the form*

$$|x_{ij}| \le \min\left\{1, \rho^{|i-j|-d_0}\right\}, \quad |y_{ij}| \le \min\left\{1, \rho^{|i-j|-d_0}\right\}.$$

Then the product XY can be bounded by

$$\left|(XY)_{ij}\right| \le \begin{cases} \Big(|i-j| - 2d_0 - 1 + \dfrac{2}{1-\rho}\Big)\rho^{|i-j|-2d_0}, & \text{if } |i-j| \ge 2d_0, \\ 2d_0 - |i-j| - 1 + \dfrac{2}{1-\rho}, & \text{if } |i-j| < 2d_0. \end{cases} \tag{6.3.6}$$

Proof. Without loss of generality, we only need to prove the conclusion for $i \le j$. Then for $i > j$, the conclusion is obtained using the fact that $(XY)_{ij} = (Y^T X^T)_{ji}$.

When $i \le j - 2d_0$, we have

$$\begin{aligned}
\left|(XY)_{ij}\right| &\le \sum_{k=-\infty}^{i+d_0} |y_{kj}| + \sum_{k=i+d_0+1}^{j-d_0-1} |x_{ik}y_{kj}| + \sum_{k=j-d_0}^{+\infty} |x_{ik}| \\
&\le \Big(|i-j| - 2d_0 - 1 + \frac{2}{1-\rho}\Big)\rho^{|i-j|-2d_0}.
\end{aligned}$$

When $j-2d_0 < i \le j$, we use

$$\begin{aligned}\big|(XY)_{ij}\big| &\le \sum_{k=-\infty}^{j-d_0-1} |y_{kj}| + \sum_{k=j-d_0}^{i+d_0} |x_{ik}y_{kj}| + \sum_{k=i+d_0+1}^{+\infty} |x_{ik}| \\ &\le 2d_0 - |i-j| + 1 + \frac{2\rho}{1-\rho} \\ &\le 2d_0 - |i-j| - 1 + \frac{2}{1-\rho}\end{aligned}$$

to obtain the conclusion. □

The estimates (6.3.5) and (6.3.6) can certainly be applied to finite matrices and yield decay bounds for $\exp\big[\mathrm{i}\beta W_n^-(\alpha)\big]$. To obtain an easily computable decay bound, we set

$$d_0 = \lceil 6|\alpha| \rceil, \qquad \rho = \frac{|\alpha|}{d_0 - 3|\alpha|},$$

and conclude from Lemma 6.3.3 that

$$\begin{aligned}\Big|\big(\exp[\mathrm{i}\beta W_n^-(\alpha)]\big)_{ij}\Big| &\le \Big(|i-j| - 2d_0 - 1 + \frac{2}{1-\rho}\Big)\rho^{|i-j|-2d_0} \\ &\le \big(|i-j| - 2d_0 + 2\big)\rho^{|i-j|-2d_0}\end{aligned}$$

when $|i-j| \ge 2d_0$, based on the fact that $\rho \le 1/3$. Then we consider the function

$$f(x) = (x+2)\Big(\frac{\mathrm{e}}{3}\Big)^x, \qquad (x \in \mathbb{R}).$$

The maximum of $f(x)$ is obtained by solving the equation $f'(x) = 0$ and yields

$$\max_{x\ge 0} f(x) = f\Big(\frac{3-2\ln 3}{\ln 3 - 1}\Big) = \frac{9}{\mathrm{e}^3(\ln 3 - 1)} < 5.$$

Then applying the inequality

$$\big(|i-j| - 2d_0 + 2\big)\Big(\frac{1}{3}\Big)^{|i-j|-2d_0} \le 5\exp\big(-|i-j| + 2d_0\big), \tag{6.3.7}$$

the decay bound (6.3.6) on $\exp\big[\mathrm{i}\beta W_n^-(\alpha)\big]$ simplifies to

$$\Big|\big(\exp[\mathrm{i}\beta W_n^-(\alpha)]\big)_{ij}\Big| \le 5\exp\big(2\lceil 6|\alpha|\rceil - |i-j|\big) \le 5\exp\big(12\lceil|\alpha|\rceil - |i-j|\big)$$

for $|i-j| \ge 2d_0$. Notice that $\exp\big(12\lceil|\alpha|\rceil - |i-j|\big) > 1$ when $|i-j| < 2d_0 < 12\lceil|\alpha|\rceil$. Taking into account that $\big\|\exp[\mathrm{i}\beta W_n^-(\alpha)]\big\|_2 = 1$, we combine the two cases into

$$\Big|\big(\exp[\mathrm{i}\beta W_n^-(\alpha)]\big)_{ij}\Big| \le \min\big\{1, 5\exp\big(12\lceil|\alpha|\rceil - |i-j|\big)\big\}. \tag{6.3.8}$$

An important observation is that both (6.3.6) and (6.3.8) provide estimates independent of n.

Finally, we remark that Theorem 6.2.3 can also be applied to $W_n^-(\alpha)$ for any given n. However, since $\delta = \Theta(n)$, the estimate deteriorates as n increases, see Figure 6.2.

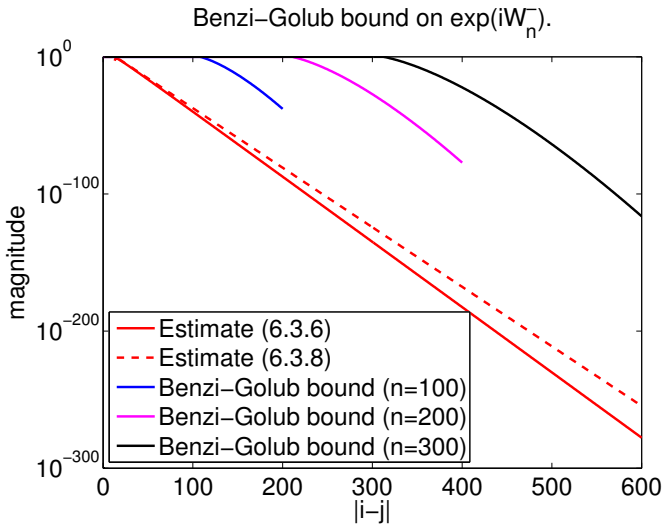


Figure 6.2 – The Benzi-Golub bound (6.2.3) with optimally chosen χ deteriorates as n increases. Our estimates (6.3.6) and (6.3.8) are also provided for reference.

6.3.2 Case study for Wilkinson-type W^+ matrices

Let us consider another Wilkinson-type matrix

$$W^+(\alpha) = \text{Tridiag}\left\{\begin{array}{ccccccccccc} \cdots & & \alpha & & \cdots & & \alpha & \alpha & \cdots & & \alpha & & \cdots \\ \cdots & & n & & \cdots & & 1 & 0 & 1 & & \cdots & & n & & \cdots \\ \cdots & & \alpha^* & & \cdots & & \alpha^* & \alpha^* & \cdots & & \alpha^* & & \cdots \end{array}\right\}$$

and its finite diagonal block

$$W_n^+(\alpha) = \text{Tridiag}\left\{\begin{array}{ccccccccccc} \alpha & & \cdots & & \alpha & & \alpha & \cdots & & \alpha \\ n & & \cdots & & 1 & & 0 & 1 & \cdots & & n \\ \alpha^* & & \cdots & & \alpha^* & & \alpha^* & \cdots & & \alpha^* \end{array}\right\}.$$

Such a matrix has also been considered in [114, 153]. The spectral decomposition of $W_n^+(\alpha)$ can be constructed from the spectral decomposition of two smaller matrices

$$U_n(\alpha) = \text{Tridiag}\left\{\begin{array}{ccccccccc} |\alpha| & & \cdots & & |\alpha| & & |\alpha| & & \\ n & & \cdots & & \cdots & & 1 & & 0 \\ |\alpha| & & \cdots & & |\alpha| & & 2|\alpha| & & \end{array}\right\}$$

and

$$V_n(\alpha) = \text{Tridiag}\left\{ \begin{array}{ccccc} |\alpha| & \cdots & |\alpha| \\ n & \cdots & \cdots & 1 \\ |\alpha| & \cdots & |\alpha| \end{array} \right\},$$

see [163] for details. Unlike the matrix $W_n^-(\alpha)$, the eigenvectors of $W_n^+(\alpha)$ do not have the exponential decay property (6.2.1) when $\alpha \neq 0$, no matter how we order the eigenvectors. However, it is still possible to exploit some localization, see the following lemma.

Lemma 6.3.4. *Let the eigenvalues of $W_n^+(\alpha)$ be in the order $\lambda_{-n}, \ldots, \lambda_n$, where*

$$\lambda_n \geq \lambda_{-n} \geq \lambda_{n-1} \geq \lambda_{-n+1} \geq \cdots \geq \lambda_1 \geq \lambda_{-1} \geq \lambda_0.$$

The corresponding normalized eigenvectors are x_{-n}, x_{-n+1}, …, x_{n-1}, x_n. Then the entries of these eigenvectors satisfy

$$\begin{aligned} |x_j(i)| &\leq \prod_{k=0}^{||i|-|j||-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}, \quad (|i| \geq |j|), \\ |x_j(i)| &\leq 2 \prod_{k=0}^{||i|-|j||-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}, \quad (|i| < |j|), \end{aligned} \tag{6.3.9}$$

for any integer $d_0 \geq 4|\alpha|$.

Proof. In the following we assume that $\alpha \neq 0$ since $\alpha = 0$ is a trivial case. By Weyl's theorem, we conclude that $\left|\lambda_j - |j|\right| \leq \left\|W_n^+(\alpha) - W_n^+(0)\right\|_2 \leq 2|\alpha|$, i.e., $\lambda_j \in \left(|j| - 2|\alpha|, |j| + 2|\alpha|\right)$. The corresponding eigenvector satisfies the following set of equations.

$$\begin{aligned} &(n-\lambda_j)x_j(-n) + \alpha x_j(-n+1) = 0 \\ &(n-1-\lambda_j)x_j(-n+1) + \alpha^* x_j(-n) + \alpha x_j(-n+2) = 0 \\ &\cdots \\ &(|k|-\lambda_j)x_j(k) + \alpha^* x_j(k-1) + \alpha x_j(k+1) = 0 \\ &\cdots \\ &(n-1-\lambda_j)x_j(n-1) + \alpha^* x_j(n-2) + \alpha x_j(n) = 0 \\ &(n-\lambda_j)x_j(n) + \alpha^* x_j(n-1) = 0 \end{aligned} \tag{6.3.10}$$

In the following, we discuss three cases—$i \geq |j|$, $i \geq -|j|$, and $|i| < |j|$.

We first analyze the case $i \geq |j|$, or more precisely, the nontrivial case $i \geq |j| + d_0$. Just like the proof of Lemma 6.3.1, we show by induction that

$$|x_j(k)| \leq \frac{|\alpha|}{k-|j|-3|\alpha|}|x_j(k-1)| \tag{6.3.11}$$

holds for any integer k satisfying $|j|+d_0 \le k \le n$. The last equation in (6.3.10) implies that

$$|x_j(n)| = \frac{|\alpha|}{n-\lambda_j}|x_j(n-1)| \le \frac{|\alpha|}{n-|j|-2|\alpha|}|x_j(n-1)| \le \frac{|\alpha|}{n-|j|-3|\alpha|}|x_j(n-1)|$$

since $n-|j|-3|\alpha| \ge d_0-3|\alpha| \ge |\alpha| > 0$. For $|j|+d_0 \le k < n$, we consider

$$(k-\lambda_j)x_j(k)+\alpha^* x_j(k-1)+\alpha x_j(k+1)=0.$$

A consequence of the induction hypothesis is that $|x_j(k+1)| \le |x_j(k)|$. Then we obtain

$$|\alpha x_j(k-1)| \ge |(k-\lambda_j)x_j(k)| - |\alpha x_j(k+1)| \ge (k-\lambda_j-|\alpha|)|x_j(k)|$$

and hence

$$|x_j(k)| \le \frac{|\alpha|}{k-\lambda_j-|\alpha|}|x_j(k-1)| \le \frac{|\alpha|}{k-|j|-3|\alpha|}|x_j(k-1)|.$$

The completes the proof of (6.3.11). Then applying (6.3.11) repeatedly and taking into account that $|x_j(|j|+d_0-1)| \le 1$ yields

$$|x_j(i)| \le \prod_{k=|j|+d_0}^{i} \frac{|\alpha|}{k-|j|-3|\alpha|} = \prod_{k=0}^{i-|j|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{||i|-|j||-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}.$$

Then the case $i \le -|j|$ (more precisely, $i \le -|j|-d_0$) is similar. We first prove by induction that

$$|x_j(k)| \le \frac{|\alpha|}{-k-|j|-3|\alpha|}|x_j(k+1)| \tag{6.3.12}$$

holds for any integer k satisfying $-n \le k \le -|j|-d_0$. We conclude from the first equation in (6.3.10) that

$$|x_j(-n)| = \frac{|\alpha|}{n-\lambda_j}|x_j(-n+1)| \le \frac{|\alpha|}{n-|j|-2|\alpha|}|x_j(-n+1)| \le \frac{|\alpha|}{n-|j|-3|\alpha|}|x_j(-n+1)|.$$

Then for $-n < k \le -|j|-d_0$, we consider

$$(-k-\lambda_j)x_j(k)+\alpha^* x_j(k-1)+\alpha x_j(k+1)=0.$$

Using $|x_j(k-1)| \le |x_j(k)|$, a weaker form of the induction hypothesis, we obtain

$$|\alpha x_j(k+1)| \ge |(-k-\lambda_j)x_j(k)| - |\alpha x_j(k-1)| \ge (-k-\lambda_j-|\alpha|)|x_j(k)|$$

and hence

$$|x_j(k)| \le \frac{|\alpha|}{-k-\lambda_j-|\alpha|}|x_j(k+1)| \le \frac{|\alpha|}{k-|j|-3|\alpha|}|x_j(k+1)|.$$

The inequality (6.3.12) is then proved. Applying (6.3.12) repeatedly, we obtain

$$|x_j(i)| \le \prod_{k=i}^{-|j|-d_0} \frac{|\alpha|}{-k-|j|-3|\alpha|} = \prod_{k=0}^{-i-|j|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{\big||i|-|j|\big|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}.$$

Finally, we consider the case $|i| < |j|$, or more precisely, the nontrivial case $|i| \le |j| - d_0$. Suppose that

$$i_0 = \arg\min_i |x_j(i)|, \quad \text{s.t. } -|j| + d_0 \le i \le |j| - d_0.$$

In the following we show by induction that

$$|x_j(k)| \le \frac{|\alpha|}{|j|-|k|-3|\alpha|}|x_j(k+1)|, \quad (i_0 < k \le |j| - d_0), \tag{6.3.13}$$

$$|x_j(k)| \le \frac{|\alpha|}{|j|-|k|-3|\alpha|}|x_j(k-1)|, \quad (-|j| + d_0 \le k < i_0). \tag{6.3.14}$$

From the equation

$$(|i_0+1| - \lambda_j)x_j(i_0+1) + \alpha^* x_j(i_0) + \alpha x_j(i_0+2) = 0$$

we obtain

$$|\alpha x_j(i_0+2)| \ge \big|(|i_0+1| - \lambda_j)x_j(i_0+1)\big| - |\alpha x_j(i_0)| \ge \big(\big||i_0+1| - \lambda_j\big| - |\alpha|\big)|x_j(i_0+1)|$$

and then

$$|x_j(i_0+1)| \le \frac{|\alpha|}{\big||i_0+1| - \lambda_j\big| - |\alpha|}|x_j(i_0+2)| \le \frac{|\alpha|}{|j|-|i_0+1|-3|\alpha|}|x_j(i_0+2)|.$$

For $i_0 + 1 < k \le |j| - d_0$, we use $|x_j(k-1)| \le |x_j(k)|$, which is a consequence of the induction hypothesis, to obtain

$$|\alpha x_j(k+1)| \ge \big|(|k| - \lambda_j)x_j(k)\big| - |\alpha x_j(k-1)| \ge \big(\big||k| - \lambda_j\big| - |\alpha|\big)|x_j(k)|$$

and thus

$$|x_j(k)| \le \frac{|\alpha|}{\big||k| - \lambda_j\big| - |\alpha|}|x_j(k+1)| \le \frac{|\alpha|}{|j|-|k|-3|\alpha|}|x_j(k+1)|.$$

This completes the proof of the inequality (6.3.13). The proof of (6.3.14) starts from analyzing the equation

$$(|i_0-1| - \lambda_j)x_j(i_0-1) + \alpha^* x_j(i_0-2) + \alpha x_j(i_0) = 0$$

and follows the same procedure. Then repeatedly applying (6.3.13) and (6.3.14) yields

$$|x_j(i)| \le \prod_{k=i}^{|j|-d_0} \frac{|\alpha|}{|j|-|k|-3|\alpha|} = \prod_{k=0}^{|j|-|i|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{\big||j|-|i|\big|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}$$

when $i_0 < i \le |j|-d_0$, and

$$|x_j(i)| \le \prod_{k=-|j|+d_0}^{i} \frac{|\alpha|}{|j|-|k|-3|\alpha|} = \prod_{k=0}^{|j|-|i|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{\big||j|-|i|\big|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|},$$

when $-|j|+d_0 \le i < i_0$, respectively. Therefore, we have proved that (6.3.9) holds for all i satisfying $-|j| < i < |j|$ except for $i = i_0$. It remains to show the case $i = i_0$. If $i_0 \neq 0$, we have

$$|x_j(i_0)| \le |x_j(-i_0)| \le \prod_{k=0}^{|j|-|-i_0|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} = \prod_{k=0}^{\big||i_0|-|j|\big|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}.$$

If $i_0 = 0$, we conclude from

$$-\lambda_j x_j(0) + \alpha^* x_j(-1) + \alpha x_j(1) = 0$$

that

$$\begin{aligned} |x_j(0)| &\le \frac{|\alpha|}{|\lambda_j|}\big(|x_j(1)| + |x_j(-1)|\big) \\ &\le \frac{|\alpha|}{|j|-2|\alpha|} \cdot 2 \prod_{k=0}^{|j|-1-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|} \\ &\le 2 \prod_{k=0}^{|j|-d_0} \frac{|\alpha|}{k+d_0-3|\alpha|}. \end{aligned}$$

The proof of Lemma 6.3.4 is completed. □

Let $X_n = [x_{-n}, \ldots, x_n]$. Then Lemma 6.3.4 implies that the decay rate in X_n is independent of n. We can simplify the decay bound to an exponential decay bound of the form

$$|x_j(i)| \le K \cdot \max\{\rho^{|i-j|}, \rho^{|i+j|}\} \le K\big(\rho^{|i-j|} + \rho^{|i+j|}\big),$$

e.g., $K = 2^{d_0+1}$ and $\rho = 1/2$ when choosing $d_0 = \lceil 5|\alpha| \rceil$. See Figure 6.3 for an illustration of the decay property of X_n. We now show that the same type of decay bound for $\exp\big[\mathrm{i}\beta W_n^+(\alpha)\big]$ can also be derived. Notice that X_n can be split as $X_n = Y + \Pi Z$, where both Y and Z have the exponential decay property $\max\{|y_{ij}|, |z_{ij}|\} \le K \cdot \rho^{|i-j|}$ and

$$\Pi = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}.$$

Then

$$\left|\exp\left[\mathrm{i}\beta W_n^+(\alpha)\right]\right| \le |X_n||X_n|^T \le \left(|Y||Y|^T + \Pi|Z||Z|^T\Pi\right) + \left(\Pi|Z||Y|^T + |Y||Z|\Pi\right),$$

where $|Y||Y|^T$, $|Z||Z|^T$, $|Z||Y|^T$, and $|Y||Z|^T$ all have the exponential decay property. By choosing

$$d_0 = \lceil 6|\alpha| \rceil, \qquad \rho = \frac{|\alpha|}{d_0 - 3|\alpha|},$$

and applying Lemma 6.3.3 to $X \leftarrow Y/2$ and $Y \leftarrow Y^T/2$, we obtain

$$\left|\left(|Y||Y|^T\right)_{ij}\right| \le 4(|i-j| - 2d_0 + 2)\rho^{|i-j|-2d_0}$$

when $|i-j| \ge 2d_0$. Applying (6.3.7) yields,

$$\left|(|Y||Y|^T)_{ij}\right| \le 20\exp(-|i-j| + 2d_0),$$

for $|i-j| \ge 2d_0$. The entries of $|Z||Z|^T$, $|Z||Y|^T$, and $|Y||Z|^T$ are bounded in the same manner, i.e.,

$$\begin{aligned}
&\left|(|Z||Z|^T)_{ij}\right| \le 20\exp(-|i-j| + 2d_0),\\
&\left|(|Z||Y|^T)_{ij}\right| \le 20\exp(-|i-j| + 2d_0),\\
&\left|(|Y||Z|^T)_{ij}\right| \le 20\exp(-|i-j| + 2d_0).
\end{aligned}$$

Comining these bounds and taking into account the fact that $\left\|\exp[\mathrm{i}\beta W_n^+(\alpha)]\right\|_2 = 1$, we obtain

$$\left|\left(\exp[\mathrm{i}\beta W_n^+(\alpha)]\right)_{ij}\right| \le \min\left\{1, 40\left[\exp\left(12\lceil|\alpha|\rceil - |i-j|\right) + \exp\left(12\lceil|\alpha|\rceil - |i+j|\right)\right]\right\}, \quad (6.3.15)$$

i.e., the entries of $\exp\left[\mathrm{i}\beta W_n^+(\alpha)\right]$ decay away from its diagonal as well as away from its anti-diagonal. We call this kind of decay property *bimodal exponential decay*. In contrast, we call the decay property (6.2.1) *unimodal exponential decay*. We will see in Section 6.3.3 that finite section methods based on bimodal decay can also be established, which naturally covers the unimodal diagonal decay property (6.2.1).

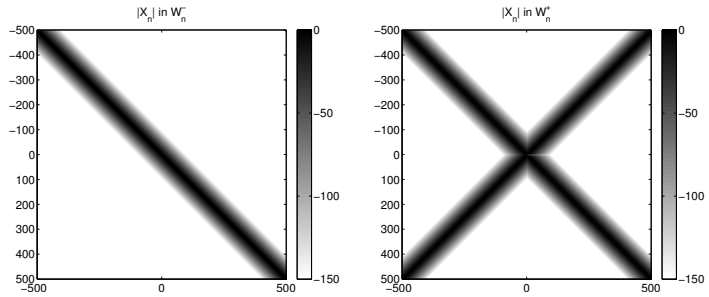


Figure 6.3 – Localized eigenvectors of $W_n^-(\alpha)$ and $W_n^+(\alpha)$ (for $n = 500$, $\alpha = 1$).

6.3.3 The finite section method

In Sections 6.3.1 and 6.3.2, we derived decay properties of two classes of finite Wilkinson-type matrices. Now we show that this kind of decay property is sufficient to guarantee the accuracy of the finite section method. Since the decay bounds are only available for finite matrices, we require a slightly different approach compared to the one of Section 6.2.2.

Consider two dynamical systems

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} = \mathrm{i}\begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{bmatrix}\begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} \tilde{E}_{11} & 0 & 0 \\ 0 & \tilde{E}_{22} & 0 \\ 0 & 0 & \tilde{E}_{33} \end{bmatrix} = \mathrm{i}\begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ 0 & 0 & A_{33} \end{bmatrix}\begin{bmatrix} \tilde{E}_{11} & 0 & 0 \\ 0 & \tilde{E}_{22} & 0 \\ 0 & 0 & \tilde{E}_{33} \end{bmatrix}$$

where E_{22} and $\tilde{E}_{22}$ are the central $(-w:w,-w:w)$ diagonal blocks. Now we assume that the tridiagonal Hermitian matrix A has the bimodal exponential decay property in the truncated exponentials:

$$\left|[\tilde{E}_{22}(s)]_{ij}\right| \le K\left(\rho^{|i-j|}+\rho^{|i+j|}\right),$$

where the constants K and ρ are independent of w and $s\in[0,1]$. We have already seen from the previous two subsections that this assumption holds for $A=\beta W^{\pm}(\alpha)$. From (6.1.2), we obtain

$$\begin{aligned}
&\left[E_{22}-\tilde{E}_{22}\right](1) \\
&= \mathrm{i}\int_0^1 \begin{bmatrix} 0 & I & 0 \end{bmatrix} \exp[\mathrm{i}(1-s)A] \begin{bmatrix} 0 & A_{12} & 0 \\ A_{21} & 0 & A_{23} \\ 0 & A_{32} & 0 \end{bmatrix} \begin{bmatrix} \tilde{E}_{11}(s) & 0 & 0 \\ 0 & \tilde{E}_{22}(s) & 0 \\ 0 & 0 & \tilde{E}_{33}(s) \end{bmatrix} \begin{bmatrix} 0 \\ I \\ 0 \end{bmatrix} \mathrm{d}s \\
&= \mathrm{i}\int_0^1 \left[E_{21}(1-s)A_{12}+E_{23}(1-s)A_{32}\right]\tilde{E}_{22}(s)\,\mathrm{d}s. \qquad (6.3.16)
\end{aligned}$$

Notice that $E_{21}(1-s)A_{12}$ has nonzero entries only in its first column and there exists an upper bound $\|E_{21}(1-s)A_{12}\|_2 \le \|A_{12}\|_2$. Using the bimodal exponential decay property of $\tilde{E}_{22}(s)$, we obtain that

$$\left\|[E_{21}(1-s)A_{12}\tilde{E}_{22}(s)]_{(:,j)}\right\|_2 \le \|A_{12}\|_2\cdot K\left(\rho^{|j+w|}+\rho^{|j-w|}\right), \qquad j=-w,\ldots,w,$$

i.e., only the first and last several columns of this matrix can have nonnegligible entries. A similar inequality holds for $E_{23}(1-s)A_{32}\tilde{E}_{22}(s)$. Therefore the $(-m:m)$-th columns in $E_{22}(1)$ and $\tilde{E}_{22}(1)$ agree with each other with accuracy at least $(\|A_{12}\|_2+\|A_{32}\|_2)\cdot K\left(\rho^{w-m}+\rho^{w+m}\right)$.

Algorithm 6.2 (A posteriori) Finite section method for $\exp(\mathrm{i}A)$

Input: A is tridiagonal and Hermitian, $m \in \mathbb{N}$, $\tau > 0$.
Additionally, $\exp\big[\mathrm{i}A_{(-n:n,-n:n)}\big]$ is known to have the bimodal exponential decay property for all n.

1: Let $k \leftarrow 0$, $w^{(0)} \leftarrow 2m$.
2: Compute $E \leftarrow \exp\big[\mathrm{i}A_{(-w^{(k)}:w^{(k)},-w^{(k)}:w^{(k)})}\big]$.
3: Let $T \leftarrow \big|A_{(-w^{(k)}-1,-w^{(k)})}\big| \cdot \big\|E_{(-w^{(k)},-m:m)}\big\|_1 + \big|A_{(w^{(k)}+1,w^{(k)})}\big| \cdot \big\|E_{(w^{(k)},-m:m)}\big\|_1$.
4: **if** $T < \tau$ **then**
5: Output $E_{(-m:m,-m:m)}$.
6: **else**
7: Let $w^{(k+1)} \leftarrow 2w^{(k)}$, $k \leftarrow k+1$.
8: **goto** Step 2.
9: **end if**

Certainly the $(2m+1) \times (2m+1)$ desired window is contained in this region. Therefore, by choosing a $(2w+1) \times (2w+1)$ computational window satisfying

$$\big(\|A_{12}\|_2 + \|A_{32}\|_2\big) \cdot K\big(\rho^{w-m} + \rho^{w+m}\big) \le \tau, \tag{6.3.17}$$

we ensure that $\big[\tilde{E}_{22}(1)\big]_{(-m:m,-m:m)}$ approximates the desired block in $E_{22}(1)$ with accuracy τ. Therefore, we can use (6.3.17) to find a suitable w a priori and obtain a finite section method similar to Algorithm 6.1.

Sometimes a priori estimates on the decay can be too pessimistic (e.g., if we apply the Benzi-Golub bound to $\exp\big[\mathrm{i}W_n^-(\alpha)\big]$). In some case we might even not have any concrete estimate for a moderate computational window despite the fact that we have the knowledge of asymptotic decay. Then algorithms such as Algorithm 6.1 become inappropriate. As a remedy for this issue, we propose a repeated doubling approach based on the a posteriori error estimate using (6.3.16), as shown in Algorithm 6.2. In this algorithm no a priori knowledge about the detailed decay rate is required. The computational window at most doubles the smallest one that fulfills the accuracy requirement. Similar techniques on stopping criteria can be found in [51, 94].

Finally, we return to the problem left in the previous subsections—the decay property of doubly-infinite matrices $\exp\big[\mathrm{i}\beta W^{\pm}(\alpha)\big]$. Theorem 6.3.5 states such a decay property. Interestingly, this property is derived as a consequence of the finite section method.

Theorem 6.3.5. *For any real number β, the doubly-infinite matrices $\exp[\mathrm{i}\beta W^{\pm}(\alpha)]$ have the bimodal exponential decay property. Moreover, we have estimates*

$$\begin{aligned}
\Big|\big(\exp[\mathrm{i}\beta W^{+}(\alpha)]\big)_{ij}\Big| &\le \min\Big\{1, 40\big[\exp\big(12\lceil|\alpha|\rceil - |i-j|\big) + \exp\big(12\lceil|\alpha|\rceil - |i+j|\big)\big]\Big\},\\
\Big|\big(\exp[\mathrm{i}\beta W^{-}(\alpha)]\big)_{ij}\Big| &\le \min\Big\{1, 5\exp\big(12\lceil|\alpha|\rceil - |i-j|\big)\Big\},
\end{aligned}$$

for all $i, j \in \mathbb{Z}$.

Proof. Let us consider $\exp[\mathrm{i}\beta W^+(\alpha)]$ first. We have already shown the decay property

$$\left|\left(\exp[\mathrm{i}\beta W_w^+(\alpha)]\right)_{ij}\right| \le \min\left\{1, K\left(\rho^{|i-j|}+\rho^{|i+j|}\right)\right\}$$

for any positive integer w, where $K = 40\exp\left(12\lceil|\alpha|\rceil\right)$ and $\rho = \mathrm{e}^{-1}$, see (6.3.15). To estimate the magnitude of the (i,j)-entry of $\exp[\mathrm{i}\beta W^+(\alpha)]$, we set $\tau = \varepsilon K\left(\rho^{|i-j|}+\rho^{|i+j|}\right)$ and $m = \max\{|i|,|j|\}$, where ε can be any positive number. Using the finite section method above, we can find a suitable $(2w+1)\times(2w+1)$ window such that

$$\begin{aligned}\left|\left(\exp[\mathrm{i}\beta W^+(\alpha)]\right)_{ij}\right| &\le \tau + \left|\left(\exp[\mathrm{i}\beta W_w^+(\alpha)]\right)_{ij}\right| \\ &\le \tau + K\left(\rho^{|i-j|}+\rho^{|i+j|}\right) \\ &= (1+\varepsilon)K\left(\rho^{|i-j|}+\rho^{|i+j|}\right).\end{aligned}$$

Since K and ρ are independent of w, letting $\varepsilon \to 0+$, we conclude that

$$\left|\left(\exp[\mathrm{i}\beta W^+(\alpha)]\right)_{ij}\right| \le K\left(\rho^{|i-j|}+\rho^{|i+j|}\right).$$

Taking into account the fact that $\left\|\exp[\mathrm{i}\beta W^+(\alpha)]\right\|_2 = 1$, we obtain

$$\left|\left(\exp[\mathrm{i}\beta W^+(\alpha)]\right)_{ij}\right| \le \min\left\{1, K\left(\rho^{|i-j|}+\rho^{|i+j|}\right)\right\}.$$

The proof for $\exp[\mathrm{i}\beta W^-(\alpha)]$ follows exactly the same procedure. We have shown that

$$\left|\left(\exp[\mathrm{i}\beta W_w^-(\alpha)]\right)_{ij}\right| \le \min\left\{1, \tilde{K}\rho^{|i-j|}\right\}$$

for any positive integer w, where $\tilde{K} = 5\exp\left(12\lceil|\alpha|\rceil\right)$ and $\rho = \mathrm{e}^{-1}$. Setting $m = \max\{|i|,|j|\}$, $\tau = \varepsilon\tilde{K}\rho^{|i-j|}$, and applying the finite section method above, we find a suitable $(2w+1)\times(2w+1)$ window such that

$$\left|\left(\exp[\mathrm{i}\beta W^-(\alpha)]\right)_{ij}\right| \le \tau + \left|\left(\exp[\mathrm{i}\beta W_w^-(\alpha)]\right)_{ij}\right| \le (1+\varepsilon)\tilde{K}\rho^{|i-j|}.$$

Letting $\varepsilon \to 0+$ and taking into account the fact that $\left\|\exp[\mathrm{i}\beta W^-(\alpha)]\right\|_2 = 1$, we obtain

$$\left|\left(\exp[\mathrm{i}\beta W^-(\alpha)]\right)_{ij}\right| \le \min\left\{1, \tilde{K}\rho^{|i-j|}\right\}.$$ □

6.3.4 Relation to aggressive early deflation

We have seen in Chapters 2 and 3 that aggressive early deflation is an advanced deflation strategy which helps accelerate the convergence of the QR algorithm. In this subsection, we use the idea of AED to obtain another derivation of the finite section method. The technique below combines the idea of *middle deflation* [29] and the explanation of AED in terms of localization [114].

We still consider the partitioning

$$A = \begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{bmatrix},$$

where A_{22} is the $(-w:w,-w:w)$ block of A. We further assume that A_{22} has localized eigenvectors in the sense that A_{22} has a spectral decomposition $A_{22} = X\Lambda_{22}X^*$ where Λ_{22} is diagonal and the unitary matrix X has bimodal exponential decay property

$$|x_{ij}| \le K\big(\rho^{|i-j|} + \rho^{|i+j|}\big),$$

with K and ρ independent of the size of A_{22}. For example, we have shown that Wilkinson-type matrices $W^{\pm}(\alpha)$ have such a property.

Consider a block diagonal unitary transformation

$$Q = \mathrm{Diag}\{I, X, I\}.$$

Let

$$A_0 = \begin{bmatrix} A_{11} & & \\ & A_{22} & \\ & & A_{33} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix},$$

$$B_0 = Q^* A_0 Q = \begin{bmatrix} A_{11} & & \\ & \Lambda_{22} & \\ & & A_{33} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix},$$

and

$$B = Q^* A Q = \begin{bmatrix} A_{11} & S_{12} & \\ S_{21} & \Lambda_{22} & S_{23} \\ & S_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix},$$

where each S_{ij} block contains one spike. By our assumption, the entries of X decay exponentially and hence the middle part of these spikes have negligible entries. Suppose w is sufficiently large so that more than half of the entries in these spikes are negligible. Then truncating tiny tails of these spikes yields another matrix

$$B_1 = \begin{bmatrix} A_{11} & \tilde{S}_{12} & 0 \\ \tilde{S}_{21} & \Lambda_{22} & \tilde{S}_{23} \\ 0 & \tilde{S}_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}.$$

We denote $A_1 = QB_1Q^*$. Let d_1 be the maximum length of the remaining spikes and $m_1 = w - d_1$. We further notice that the difference between the central $(2w+1)\times(2w+1)$ diagonal

blocks in $\exp(\mathrm{i}B_0)$ and $\exp(\mathrm{i}B_1)$ has the following structure:

$$\left[\exp(\mathrm{i}B_0)-\exp(\mathrm{i}B_1)\right]_{(-w:w,-w:w)} = \begin{array}{c} \\ d_1 \\ 2m_1+1 \\ d_1 \end{array} \begin{array}{c} \begin{array}{ccc} d_1 & 2m_1+1 & d_1 \end{array} \\ \begin{bmatrix} * & 0 & * \\ 0 & 0 & 0 \\ * & 0 & * \end{bmatrix} \end{array}.$$

Let

$$\Delta E = \left[\exp(\mathrm{i}A_0)-\exp(\mathrm{i}A_1)\right]_{(-w:w,-w:w)} = X\left[\exp(\mathrm{i}B_0)-\exp(\mathrm{i}B_1)\right]_{(-w:w,-w:w)}X^*.$$

Then the columns of ΔE is a linear combination of $\{Xe_{-w},\ldots,Xe_{-m_1-1},Xe_{m_1+1},\ldots,Xe_w\}$. Because of the decay property of X, there exist a positive integer $m \le m_1$ such that the $(-m:m)$-th rows of ΔE are negligible, provided that m_1 is not too small. The same observation is made on the columns of ΔE. According to (6.1.2), we have

$$\left\|\exp(\mathrm{i}A)-\exp(\mathrm{i}A_1)\right\|_2 = \left\|\exp(\mathrm{i}B)-\exp(\mathrm{i}B_1)\right\|_2 \le \|\Delta S_{21}\|_2 + \|\Delta S_{23}\|_2,$$

where $\Delta S_{ij} := S_{ij} - \tilde{S}_{ij}$ is the perturbation introduced by AED. Therefore, for $-m \le i,\ j \le m$, we have

$$\begin{aligned}\left|[\exp(\mathrm{i}A)-\exp(\mathrm{i}A_0)]_{ij}\right| &\le \left\|[\exp(\mathrm{i}A)-\exp(\mathrm{i}A_1)]\right\|_2 + \|\Delta E\|_2 \\ &\le \|\Delta S_{21}\|_2 + \|\Delta S_{23}\|_2 + \|\Delta E\|_2,\end{aligned}$$

which is also negligible.

In practice, the desired window size $2m+1$ is provided by the user, and the finite section method needs to search for a computational window size $2w+1$ using the analysis above to ensure the accuracy of the solution in the desired window. Although we do not provide a quantitative estimate for d, in practice such an inconvenience does not emerge when Algorithm 6.2 is applied. Because the decay rate in X is independent of w, the distance $d = w-m$ depends on K, ρ, and d_1, but not on m or w.

6.3.5 More general unbounded matrices

We have seen that the eigenvector decay bounds, as established in Lemmas 6.3.1, 6.3.2, and 6.3.4 play important roles in the derivation of the exponential decay property as well as finite section methods for Wilkinson-type matrices. In the following we extend our analyses to a more general class of unbounded matrices and establish finite section methods. We only consider the setting explained in Section 6.1. Additional requirements on the matrices will be discussed below.

To extend the technique in Lemma 6.3.1, estimates on the eigenvalues in terms of diagonal entries are required. For any matrix A, finite or infinite, we define the *dominance factors* at its

kth row as

$$\mu_k = \begin{cases} \dfrac{1}{|a_{kk}|} \displaystyle\sum_{j\neq k} |a_{kj}|, & \text{if } a_{kk} \neq 0, \\ +\infty, & \text{if } a_{kk} = 0. \end{cases}$$

The Gershgorin circle theorem [152] on a finite Hermitian matrix A states that

$$\Lambda(A) \subset \bigcup_k \big[a_{kk} - \mu_k |a_{kk}|, a_{kk} + \mu_k |a_{kk}|\big].$$

But even if A is diagonally dominant, in general we cannot further ensure that there exists an ordering of the eigenvalues λ_k of A satisfying

$$1 - \mu_k \leq \frac{\lambda_k}{a_{kk}} \leq 1 + \mu_k, \qquad \forall\, k, \tag{6.3.18}$$

when the Gershgorin disks are not separated. For instance,

$$A = \begin{bmatrix} 25 & 1 & 16 \\ 1 & 24 & 8 \\ 16 & 8 & 26 \end{bmatrix}$$

is such a counterexample to (6.3.18). To resolve this issue and establish a valid rowwise estimate similar to (6.3.18), we introduce the following concept.

Definition 6.3.6. *Let A be a strictly diagonally dominant matrix with dominance factors $\{\mu_k\}$. Then A is called* strong diagonally dominant *if there exists a set of numbers $\{\hat{\mu}_k\}$ satisfying $\mu_k \leq \hat{\mu}_k < 1$ $(\forall\, k)$ and*

$$\begin{cases} (a_{ii} - a_{jj})\big[(a_{ii} - |\hat{\mu}_i a_{ii}|) - (a_{jj} - |\hat{\mu}_j a_{jj}|)\big] \geq 0, \\ (a_{ii} - a_{jj})\big[(a_{ii} + |\hat{\mu}_i a_{ii}|) - (a_{jj} + |\hat{\mu}_j a_{jj}|)\big] \geq 0, \end{cases} \qquad \forall\, i,\, j. \tag{6.3.19}$$

The numbers $\hat{\mu}_k$'s are called strong dominance factors *of A. The set*

$$\big\{z \in \mathbb{C} \colon |z - a_{kk}| \leq |\hat{\mu}_k a_{kk}|\big\}$$

is called an extended Gershgorin disk *with respect to $\hat{\mu}_k$.*

The condition (6.3.19) has a geometrical interpretation—the leftmost/rightmost points of these extended Gershgorin disks follow the same order as their centers. This condition is used to limit the growth of off-diagonals compared to the diagonals A. With this new concept, we derive the following lemma, which provides a rowwise estimate. A similar rowwise estimate but with a different dominance assumption can be found in [17, Proposition 2].

Lemma 6.3.7. *Let A be an $N \times N$ diagonally dominant Hermitian matrix with $\hat{\mu}_k$ $(k = 1, \ldots, N)$ being its strong dominance factors. Then the ith smallest diagonal entry d_i and the ith smallest*

eigenvalue λ_i are related by

$$1-\hat{\mu}_i \le \frac{\lambda_i}{d_i} \le 1+\hat{\mu}_i. \tag{6.3.20}$$

Proof. Without loss of generality, we assume that the diagonal entries of A are in increasing order, i.e., $d_i = a_{ii}$ for all i. By the Cauchy interlacing theorem, λ_i never exceeds the largest eigenvalue of $A_{(1:i,1:i)}$. Let μ_i be the dominance factor at ith row of A. If $d_i < 0$, then by the Gershgorin circle theorem we have

$$\lambda_i \le \max_{1\le j\le i}(1-\mu_j)d_j \le \max_{1\le j\le i}(1-\hat{\mu}_j)d_j = (1-\hat{\mu}_i)d_i.$$

If $d_i > 0$, we notice that Gershgorin disks centered in the left half plane do not produce positive eigenvalues. Hence we have

$$\lambda_i \le \max_{\substack{j\le i\\ d_j>0}}(1+\mu_j)d_j \le \max_{\substack{j\le i\\ d_j>0}}(1+\hat{\mu}_j)d_j = (1+\hat{\mu}_i)d_i.$$

The two complementary estimates can be obtained by applying the same analysis to $-A$. □

Lemma 6.3.7 provides nice rowwise estimates for eigenvalues of strong diagonally dominant Hermitian matrices even when the Gershgorin disks overlap. Evidently, all finite diagonally dominant matrices are trivially strong diagonally dominant since we can choose $\hat{\mu}_k = (1+\max_k \mu_k)/2$, which is an upper bound independent of k. However, in many cases at least some $\hat{\mu}_k$ (e.g., which corresponds to an isolated Gershgorin disk) can be chosen not far larger than μ_k. In this case (6.3.20) becomes nearly as powerful as (6.3.18). With the help of this rowwise estimate, we now extend our analysis for Wilkinson-type matrices to more general cases. The following theorem, akin to Lemma 6.3.1, illustrates the decay in eigenvectors for nearly diagonally dominant matrices.

Theorem 6.3.8. *Suppose $A = \tilde{A} + R$ is an $N \times N$ Hermitian matrix where $\tilde{A}$ and R are both tridiagonal, and in addition, $\tilde{A}$ is diagonally dominant. μ_k and $\hat{\mu}_k$ ($k = 1, \ldots, N$) are $\tilde{A}$'s dominance factors and strong dominance factors, respectively. Let $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_N$ be the eigenvalues of A, with normalized eigenvectors $x_1, x_2, \ldots, x_N$. If the diagonal of $\tilde{A}$ is in increasing order, then the entries of x_j satisfy*

$$|x_j(i)| \le \begin{cases} \displaystyle\prod_{k=i}^{k_0} \frac{\mu_k|\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk}-\tilde{a}_{jj}| - \mu_k|\tilde{a}_{kk}| - \hat{\mu}_j|\tilde{a}_{jj}| - 3\|R\|_2}, & (j \ge i), \\ \displaystyle\prod_{k=k_0}^{i} \frac{\mu_k|\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk}-\tilde{a}_{jj}| - \mu_k|\tilde{a}_{kk}| - \hat{\mu}_j|\tilde{a}_{jj}| - 3\|R\|_2}, & (j < i), \end{cases} \tag{6.3.21}$$

where k_0 is chosen between i and j such that it maximizes $|i-k_0|$ and ensures

$$|\tilde{a}_{kk}-\tilde{a}_{jj}| > 4\|R\|_2 + 2\mu_k|\tilde{a}_{kk}| + \hat{\mu}_j|\tilde{a}_{jj}|$$

for all k between i and k_0.

Proof. By Lemma 6.3.7, the eigenvalues of $\tilde{A}$ satisfy $|\tilde{\lambda}_k - \tilde{a}_{kk}| \le \hat{\mu}_k |\tilde{a}_{kk}|$. Then Weyl's theorem implies that

$$|\lambda_k - \tilde{a}_{kk}| \le |\lambda_k - \tilde{\lambda}_k| + |\tilde{\lambda}_k - \tilde{a}_{kk}| \le \|R\|_2 + \hat{\mu}_k |\tilde{a}_{kk}|.$$

The rest of proof mimics Lemma 6.3.1. By defining $x_j(0) = x_j(N+1) = 0$, the equation $(A - \lambda_j I)x_j = 0$ is rewritten as

$$a_{k,k-1}x_j(k-1) + (a_{kk} - \lambda_j)x_j(k) + a_{k,k+1}x_j(k+1) = 0, \qquad (k = 1, \ldots, N).$$

We first consider the case $i \le j$. Let k_0 be the largest integer ensuring

$$|\tilde{a}_{kk} - \tilde{a}_{jj}| > 4\|R\|_2 + 2\mu_k |\tilde{a}_{kk}| + \hat{\mu}_j |\tilde{a}_{jj}|$$

for all $k \in [1, k_0]$. We assume the existence of such a k_0 because otherwise (6.3.21) becomes a trivial bound $|x_j(i)| \le 1$. We then show by induction that

$$|x_j(k)| \le \frac{\mu_k |\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk} - \tilde{a}_{jj}| - \mu_k |\tilde{a}_{kk}| - \hat{\mu}_j |\tilde{a}_{jj}| - 3\|R\|_2} |x_j(k+1)| \tag{6.3.22}$$

holds when $1 \le k \le k_0$. For $k = 1$, we have

$$\begin{aligned}
|x_j(1)| &= \frac{|a_{12}|}{|a_{11} - \lambda_j|} |x_j(2)| \\
&\le \frac{|a_{12} - \tilde{a}_{12}| + |\tilde{a}_{12}|}{|\tilde{a}_{11} - \tilde{a}_{jj}| - |a_{11} - \tilde{a}_{11}| - |\lambda_j - \tilde{a}_{jj}|} |x_j(2)| \\
&\le \frac{\mu_1 |\tilde{a}_{11}| + \|R\|_2}{|\tilde{a}_{11} - \tilde{a}_{jj}| - \hat{\mu}_1 |\tilde{a}_{11}| - 2\|R\|_2} |x_j(2)| \\
&\le \frac{\mu_1 |\tilde{a}_{11}| + \|R\|_2}{|\tilde{a}_{11} - \tilde{a}_{jj}| - \hat{\mu}_1 |\tilde{a}_{11}| - \hat{\mu}_j |\tilde{a}_{jj} - 3\|R\|_2} |x_j(2)|.
\end{aligned}$$

Then for $1 < k \le k_0$, the induction hypothesis implies $|x_j(k-1)| \le |x_j(k)|$ and then

$$\begin{aligned}
&|a_{k,k+1}x_j(k+1)| \\
&\ge \big(|a_{kk} - \lambda_j| - |a_{k,k-1}|\big)|x_j(k)| \\
&\ge \big(|\tilde{a}_{kk} - \tilde{a}_{jj}| - |\tilde{a}_{kk} - a_{kk}| - |\lambda_j - \tilde{a}_{jj}| - |\tilde{a}_{k,k-1} - a_{k,k-1}| - |\tilde{a}_{k,k-1}|\big)|x_j(k)| \\
&\ge \big(|\tilde{a}_{kk} - \tilde{a}_{jj}| - \mu_k |\tilde{a}_{kk}| - \hat{\mu}_j |\tilde{a}_{jj}| - 3\|R\|_2\big)|x_j(k)|
\end{aligned}$$

Therefore,

$$|x_j(k)| \le \frac{\mu_k |\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk} - \tilde{a}_{jj}| - \mu_k |\tilde{a}_{kk}| - \hat{\mu}_j |\tilde{a}_{jj}| - 3\|R\|_2} |x_j(k+1)|.$$

Repeatedly applying (6.3.22) yields

$$|x_j(i)| \le \prod_{k=i}^{k_0} \frac{\mu_k|\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk} - \tilde{a}_{jj}| - \mu_k|\tilde{a}_{kk}| - \hat{\mu}_j|\tilde{a}_{jj}| - 3\|R\|_2}.$$

When $i > j$, let k_0 be the smallest integer ensuring

$$|\tilde{a}_{kk} - \tilde{a}_{jj}| > 4\|R\|_2 + 2\mu_k|\tilde{a}_{kk}| + \hat{\mu}_j|\tilde{a}_{jj}|$$

for all $k \in [k_0, N]$. We introduce $X = [x_1, \dots, x_N]$, $\Lambda = \mathrm{Diag}\{\lambda_1, \dots, \lambda_N\}$, and

$$\Pi = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}$$

Then $(-\Pi A\Pi)(\Pi X\Pi) = (\Pi X\Pi)(-\Pi\Lambda\Pi)$ is the spectral decomposition of $-\Pi A\Pi$ and the diagonal entries of $-\Pi\Lambda\Pi$ are in increasing order. Therefore, we use the first inequality in (6.3.21) to conclude that

$$|x_j(i)| = |(\Pi X\Pi)_{N+1-i,N+1-j}| \le \prod_{k=k_0}^{i} \frac{\mu_k|\tilde{a}_{kk}| + \|R\|_2}{|\tilde{a}_{kk} - \tilde{a}_{jj}| - \mu_k|\tilde{a}_{kk}| - \hat{\mu}_j|\tilde{a}_{jj}| - 3\|R\|_2}.$$

This completes the proof of Theorem 6.3.8. □

***Remark* 6.3.9.** The bound in (6.3.21) cannot provide straightaway estimate without detailed knowledge of the matrix, mainly because we do not know how small the distance $|i - k_0|$ can be. There exist matrices (e.g., Laplacian matrices) such that (6.3.21) only provides trivial bounds $|x_j(i)| \le 1$. However, there are also matrices for which the decay property of eigenvectors can be well identified using (6.3.21). For example, for the Wilkinson-type matrix $W_n^-(\alpha)$ with $n > 2|\alpha| > 0$, we can introduce a perturbation [5]

$$R = W_n^-(\alpha) - W_n^-(0) + \epsilon \cdot e_0 e_0^T, \qquad (\epsilon > 0)$$

for a sufficiently small ϵ so that $W_n^-(\alpha) - R$ is diagonally dominant. By setting $\mu_k = 0$ and $\hat{\mu}_k = 2|\alpha|/|k + \epsilon|$ for $(-n \le k \le n)$, the estimate (6.3.21) is then simplified to

$$|x_j(i)| \le \prod_{k=0}^{|i-j|-d_0} \frac{2|\alpha|}{k + d_0 - 8|\alpha|}.$$

for $d_0 > 10|\alpha|$. This bound is worse than (6.3.1) since detailed information regarding the componentwise distribution in R is lost by crudely using $\|R\|_2 \le 2|\alpha|$. Nevertheless, this estimate is still asymptotically as good as (6.3.1).

5. We set $R(0,0) = \epsilon > 0$ to guarantee the strict diagonal dominance of $W_n^-(\alpha) - R$. But this is not crucial since the analysis in Lemma 6.3.7 will not be completely ruined by a zero row.

***Remark* 6.3.10.** If the diagonal of $|\tilde{A}|$ first decreases and then increases, just like the diagonal of $W_n^+(\alpha)$, the estimate (6.3.21) needs to be slightly adjusted accordingly. Roughly speaking, $|x_j(i)|$ is small if $A(p,p)$ and $A(q,q)$ are well-separated for all p close to i and q close to j. The theorem also naturally extends to banded matrices, based on block versions of Lemmas 6.3.1 and 6.3.7 [152, Chapter 6].

Despite that Theorem 6.3.8 is a more qualitative analysis rather than a sharp quantitative one, it is evident that certain types of (finite) diagonally dominant banded Hermitian matrices, possibly with small perturbations, have localized eigenvectors. More importantly, when A is extracted from an infinite matrix, the decay bound depends only on the location (i,j), but not on the size of A. Unfortunately, without detailed information of the decay, it would be difficult to derive a decay bound for $|\exp(\mathrm{i}A)| \le |X||X|^T$ as we have done in Lemmas 6.3.2 and 6.3.3. Here we only provide an intuitive explanation about the decay in $\exp(\mathrm{i}A)$. Let $A = X\Lambda X^*$ be the spectral decomposition of A, and Y be a b-banded approximation of X with accuracy $\|X - Y\|_2 \le \tau$. Then

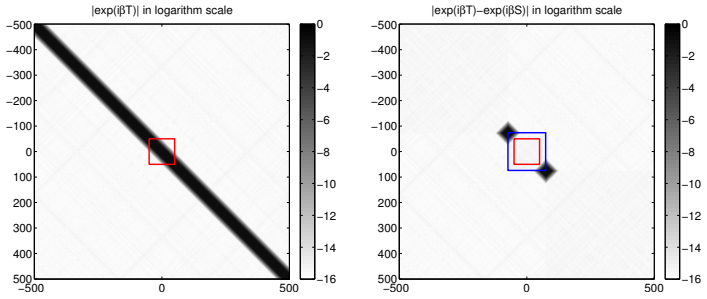
$$\begin{aligned}\|\exp(\mathrm{i}A) - Y\exp(\mathrm{i}\Lambda)Y^*\|_2 &= \|X\exp(\mathrm{i}\Lambda)X^* - Y\exp(\mathrm{i}\Lambda)Y^*\|_2\\ &\le \|X-Y\|_2\|\exp(\mathrm{i}\Lambda)\|_2\|X^*\|_2 + \|Y\|_2\|\exp(\mathrm{i}\Lambda)\|_2\|X^*-Y^*\|_2\\ &\le 2\tau(1+\tau).\end{aligned}$$

Therefore $\exp(\mathrm{i}A)$ can be well approximated by a $(2b)$-banded matrix.

As seen in Sections 6.3.3 and 6.3.4, to derive the finite section method for a doubly-infinite matrix, we only need the knowledge of decay properties in finite diagonal blocks. Hence when Theorem 6.3.8 produces nontrivial bounds for all sufficiently large diagonal blocks of a doubly-infinite matrix A, finite section methods can be applied to A. We classify such a kind of unbounded doubly-infinite matrices as follows.

1. A is Hermitian and banded.
2. A is the sum of three Hermitian matrices $A = D + G + R$, where D is diagonal and invertible, R is bounded, and $\|GD^{-1}\|_2 < 1$.
3. $D+G$ is strong diagonally dominant; in addition, the diagonal of $D+G$ changes monotonicity at most once.
4. For each extended Gershgorin disk, its $(2\|R\|_2)$-neighborhood intersects only finitely many other extended Gershgorin disks.

Loosely speaking, the third condition indicates that the diagonal of A is nearly sorted so that we can apply a banded version of Theorem 6.3.8 to obtain the decay property for sufficiently large finite diagonal blocks of $\exp(\mathrm{i}A)$; the last condition ensures that finite sections of A have reasonably well-separated eigenvalues so that the eigenvector matrix has a certain decay property. For example, any Wilkinson-type matrix, or more generally, any banded Hermitian matrix A whose off-diagonal part (i.e., by setting all diagonal entries of A to zero) is bounded and $|a_{ii} - a_{jj}| = \Theta\big(\big||i|-|j|\big|^t\big)$ for some $t > 0$, belongs to this class. In principle, both



(a) (b)

Figure 6.4 – (a) Decay property of $\exp(10\mathrm{i}\,T_{500})$. The 101×101 desired window is marked. (b) Error of the finite section method ($w=74$). Both the desired window and the computational window are marked.

Algorithm 6.1 and Algorithm 6.2 can be applied to unbounded self-adjoint matrices with slight modifications in the stopping criterion. We suggest that in general Algorithm 6.2 is preferred unless a reasonably accurate estimate of the decay is known in advance.

Finally, we make a remark on the decay rate. If (6.3.16) can be bounded by a bimodal exponential decay (e.g., it is the case when A has bounded off-diagonal entries and the bimodal decay in $\tilde{E}_{22}$ is exponential and independent of w), then the distance $d=w-m$ stays constant when the user requires a larger m. However, if the decay of (6.3.16) is slower than exponential, to keep the same accuracy requirement $w-m$ will grow as m increases. This is the major reason why exponential decay is of great interest in finite section methods.

6.4 Numerical experiments

In the following, we present numerical experiments for three examples to demonstrate the accuracy of finite section methods. We use reasonably large matrices to mimic infinite matrices. All experiments have been performed in MATLAB R2012a. The exponential function is computed via spectral decomposition (i.e., $\exp(\mathrm{i}A)=\exp(\mathrm{i}X\Lambda X^*)=X\exp(\mathrm{i}\Lambda)X^*$). It has been observed that sometimes even the componentwise accuracy of $\exp(\mathrm{i}A)$ is retained when the computed unitary matrix $\mathrm{fl}(X)$ has the exponential decay property.

Example 6.1. We first consider

$$T_n=\mathrm{Tridiag}\left\{\begin{matrix}-1 & \cdots & -1\\ 2 & \cdots & \cdots & 2\\ -1 & \cdots & -1\end{matrix}\right\}\in\mathbb{C}^{(2n+1)\times(2n+1)}$$

which are bounded with spectrum $\Lambda(T_n)\subset[0,4]$ for all $n\in\mathbb{N}$. By Theorem 6.2.3, for any constant β, $\exp(\mathrm{i}\beta T_n)$ has the exponential decay property. Suppose $n=500$, $m=50$, and $\beta=10$, i.e., the diagonal block $\left[\exp(\mathrm{i}\beta T_n)\right]_{(-50:50,-50:50)}$ is of interest. The desired (absolute) accuracy is $\tau=10^{-8}$. The magnitude of $\exp(\mathrm{i}\beta T_n)$ is shown in Figure 6.4(a).

Algorithm 6.1 requires $w \geq 74$ to fulfill the condition $K\big(\rho^{2(w-m)-1} + \rho^{2(w+m)-1}\big) \leq \tau$, with $\rho = \chi_*^{-1}$ chosen optimally from (6.2.6). As a comparison, the smallest possible computational window size to achieve accuracy 10^{-8} is $w_* = 69$. Algorithm 6.2 applied to this problem terminates after the first iterate, i.e., $w = 2m = 100$, which confirms the fact that $w < 2w_*$.

To visualize the error caused by truncation, let $S_{n,w}$ be a block diagonal approximation of T_n defined by

$$\begin{cases} S_{n,w}\big(\pm w, \pm(w+1)\big) = 0, \\ S_{n,w}\big(\pm(w+1), \pm w\big) = 0, \\ S_{n,w}(i,j) = T_n(i,j), \qquad \text{otherwise.} \end{cases}$$

It is comforting to see from Figure 6.4(b) that the error is localized around the corners of the computational window.

Example 6.2. Now we consider Wilkinson-type matrices $W^-(\alpha)$. In Figure 6.5, the exponential decay property of a 1001×1001 matrix (with $\alpha = 8$) is illustrated. It can be seen from Figure 6.5(g) that although the simplified bound (6.3.8) is asymptotically worse than the best bound on $|X||X|^T$ based on (6.3.1), the difference is insignificant for entries above 10^{-16}, since the choice $d = 6\lceil|\alpha|\rceil$ produces a modest leading factor K. Another important fact is that the decay rate is independent of the matrix size, see Figure 6.6 for an illustration.

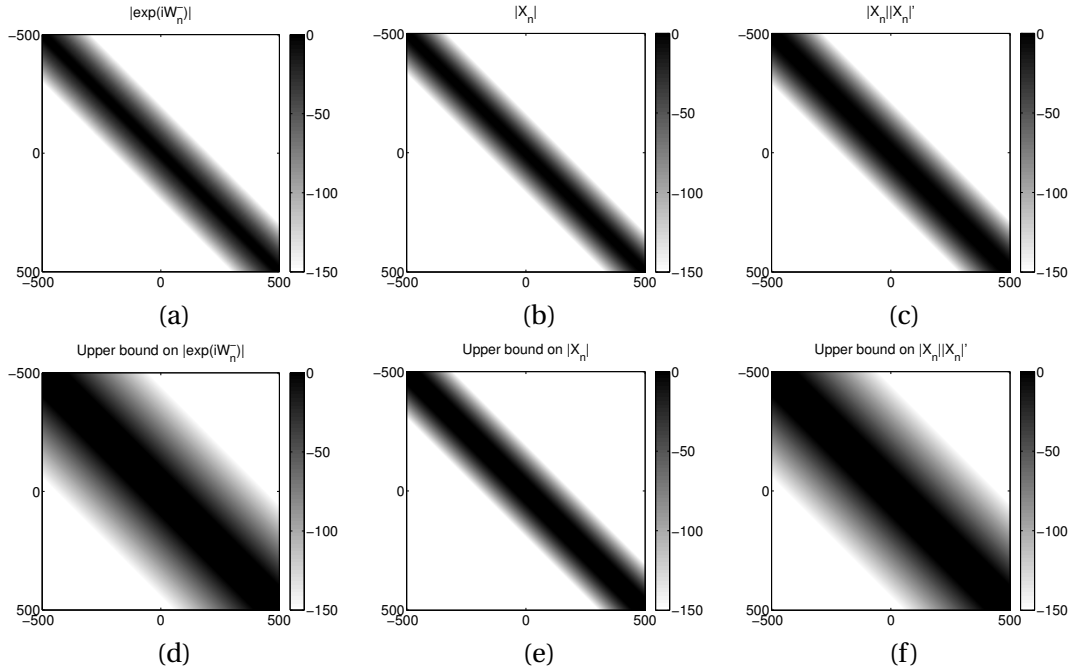
Table 6.1 contains the distance between the computational window and the desired window. The experiments were performed with different matrix sizes ($n = 100, 200, \ldots, 500$) and different central block size ($m = 10, 20, 30$). But we only present those values for different α and β, because d is independent of m and n. Our estimates are quite conservative, but still produce reasonably affordable computational window sizes. Another fact not shown in the table is that for fixed α and β, the desired $(2m+1) \times (2m+1)$ diagonal block extracted from matrices with different sizes agree quite well as expected.

Table 6.1 – The distance between the computational window and the desired section (with accuracy $\tau = 10^{-8}$). The number $d = w - m$ is derived from the a priori estimate, while $d_* = w_* - m$ is the smallest distance obtained by enumeration.

	$\alpha = 1$		$\alpha = 2$		$\alpha = 4$		$\alpha = 8$	
β	d_*	d	d_*	d	d_*	d	d_*	d
1	7	33	9	45	12	70	18	119
2	9	33	12	46	18	71	27	119
4	11	34	16	47	25	71	41	120
8	12	35	17	47	26	72	43	121

Example 6.3. Our last example is another doubly-infinite matrix

$$A = \text{Tridiag}\left\{ \begin{matrix} \cdots & n^{\frac{3}{4}} & \cdots & 1 & 1 & \cdots & n^{\frac{3}{4}} & \cdots \\ \cdots & n & \cdots & 1 & 0 & 1 & \cdots & n & \cdots \\ \cdots & n^{\frac{3}{4}} & \cdots & 1 & 1 & \cdots & n^{\frac{3}{4}} & \cdots \end{matrix} \right\}.$$



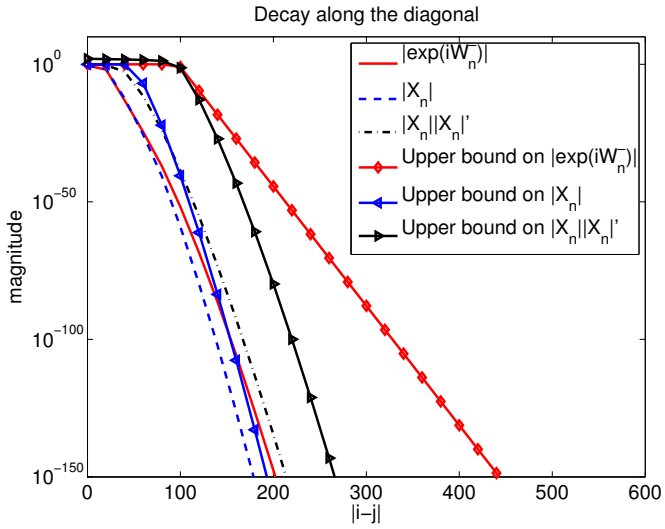


Figure 6.5 – The exponential decay property of $\exp\left[\mathrm{i}W_n^-(\alpha)\right]$, X_n, and $|X_n||X_n|^T$ with $n = 500$ and $\alpha = 8$. The upper bounds of $\exp\left[\mathrm{i}W_n^-(\alpha)\right]$ and $|X_n|$ are given by the estimates (6.3.8) and (6.3.1), respectively, with $d_0 = 6\lceil\alpha\rceil = 48$. The upper bound on $|X_n||X_n|^T$ is obtained from the upper bound on $|X_n|$ by explicit multiplication.

A variant of Theorem 6.3.8 indicates that $\exp(\mathrm{i}A)$ has a bimodal decay property, while the decay is slower than the exponential decay. Suppose we would like to extract the central 101×101 diagonal block (i.e., $m = 50$) with absolute accuracy $\tau = 10^{-8}$. Algorithm 6.2 applied to this problem terminates at $w = 4m = 200$. Plots of X, $|X||X|^T$, $\exp(\mathrm{i}A)$ as well as the error

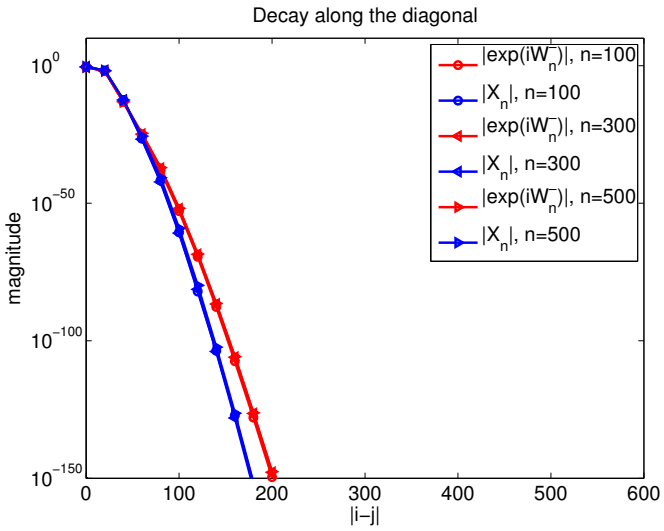


Figure 6.6 – The decay rate is independent of the matrix size ($W_n^-(\alpha)$ for $\alpha = 8$).

are shown in Figure 6.7. Although the a priori estimate based on decay in eigenvectors is too pessimistic, Algorithm 6.2 still handles this difficult example quite well.

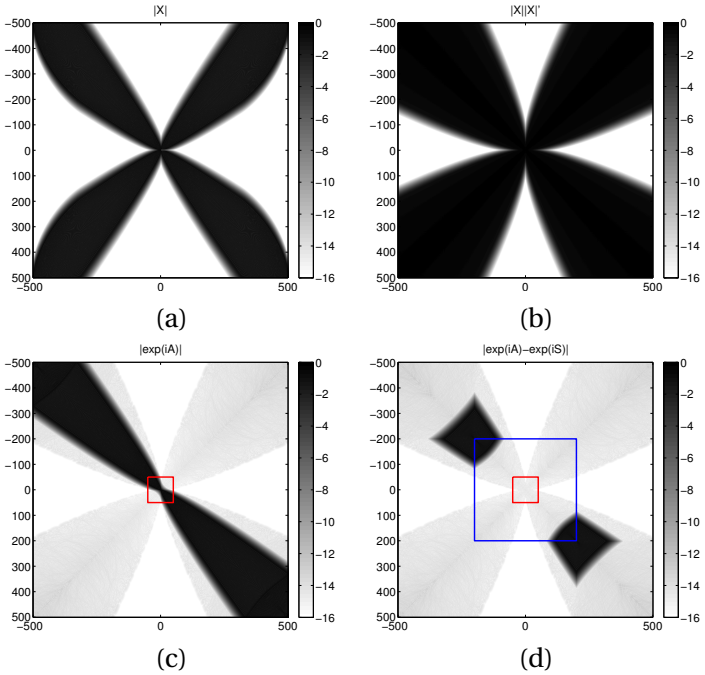


Figure 6.7 – (a)–(c) The decay in X, $|X||X|^T$, and $\exp(\mathrm{i}A)$, respectively, in Example 6.3. The desired window in $\exp(\mathrm{i}A)$ is marked. (d) Error of the finite section method. Both the desired window and the computational window are marked. Here S is the block diagonal matrix by dropping the $\pm w$th sub-diagonal entries ($w = 200$) of A.

But we remark that Algorithm 6.2 does not work if there is no decay at all in a reasonably

computable range. For example, for

$$B = \text{Tridiag}\left\{\begin{array}{ccccccccccc} \cdots & & n^{1.9} & & \cdots & 1 & 1 & \cdots & n^{1.9} & & \cdots \\ \cdots & & n^2 & & \cdots & 1 & 0 & 1 & \cdots & n^2 & & \cdots \\ \cdots & & n^{1.9} & & \cdots & 1 & 1 & \cdots & n^{1.9} & & \cdots \end{array}\right\},$$

which is also self-adjoint. There is no obvious decay in a modest finite section of $\exp(\mathrm{i}B)$, see Figure 6.8. Hence Algorithm 6.2 cannot compute the finite section with $m = 50$ for this matrix unless a computational window with $w = 1600$ is affordable.

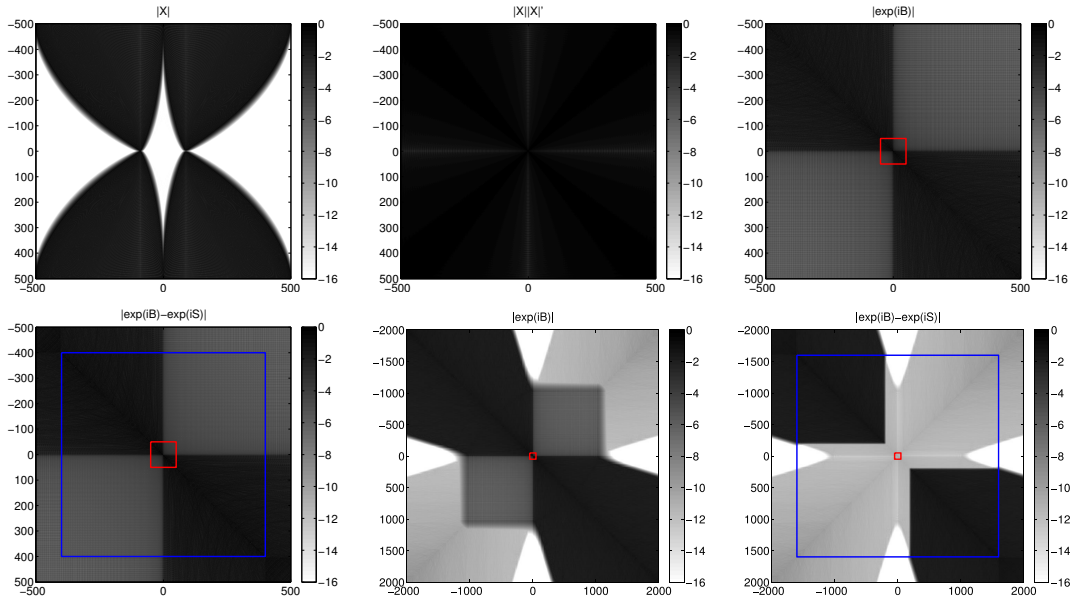


Figure 6.8 – There is no decay in modest finite sections of X, $|X||X|^T$, and $\exp(\mathrm{i}B)$ in Example 6.3. The computational window is required to be very large in order to find a good approximation. The desired window and the computational window are marked.

6.5 Summary

In this chapter, we have shown that certain decay properties can be used to establish finite section methods for extracting a finite diagonal block of $\exp(\mathrm{i}A)$. For bounded and banded Hermitian matrices, the exponential decay property of $\exp(\mathrm{i}A)$ can be derived by polynomial approximation; for unbounded matrices, we identify localized eigenvectors in its finite diagonal blocks to obtain the decay property. We also show that to attain a certain accuracy, the required distance between the desired window and the computational window stays constant when the decay is exponential. Numerical experiments demonstrate the robustness of the proposed finite section methods.

7 Conclusions

Two topics in dense and structured matrix computations are discussed in this thesis. In the following, we summarize the contributions of this thesis.

In Chapter 3, we have presented a new parallel implementation of the QR algorithm equipped with some modern techniques such as small-bulge multishift and aggressive early deflation (AED). An intermediate version of our library software PDHSEQR has been released in ScaLAPACK version 2.0. This work is largely based on the early work by Granat et al. [61]. We have made four major contributions to improve the efficiency of the algorithm. First, AED has been incorporated into the pipelined QR algorithm which is suitable for solving small- to medium-size problems. This leads to a multilevel AED approach which significantly accelerates the computation of the Schur decomposition in all levels of AED. Second, we have proposed a technique which largely reduces communication overhead by redistributing the matrix, and performing computations on a subset of processors. Third, we have proposed a refined strategy for balancing between the multishift QR sweeps and AED. Finally, a performance model has been established for the new parallel multishift QR algorithm. Although the performance model is only asymptotic, it provides estimates of the execution time and also suggests the choice of several tunable algorithmic parameters. Guidelines and autotuning tools concerning these tunable parameters are provided. With the help of these improvements, a computational bottleneck in the earlier version of PDHSEQR in [61] has been removed. Consequently, the new version of PDHSEQR is more efficient than the previous version. The improvements compared to PDLAHQR in ScaLAPACK version 1.8.0 is even more significant. Concerning the accuracy, we have also identified and fixed several anomalies in ScaLAPACK's PDLAHQR and the early version of PDHSEQR in [61] so that our new PDHSEQR is numerically more stable.

In Chapter 5, we have substantially extended the existing theory on the computation of $\exp(A)$ when A is essentially nonnegative. First, we have provided a novel a priori componentwise relative truncation error estimate for the truncated Taylor series coupled with scaling and squaring. This error estimate is used to derive the aggressively truncated Taylor series method which requires $\mathcal{O}(N^3 \log N)$ arithmetic operations. Compared to existing Taylor series

methods [167] which suffer from slow convergence and potentially require $\mathcal{O}(N^4)$ arithmetic operations, our aggressively truncated Taylor series method reduces the computational cost without sacrificing the accuracy. We have also established new a posteriori error estimates and used them to develop several variants of the aggressively truncated Taylor series method, including an interval algorithm without using interval arithmetic. Rounding error analyses have also been provided to illustrate the numerical stability of our proposed methods.

In Chapter 6, we have provided theoretical evidence for the validity of the finite section method for computing a finite diagonal block of $\exp(\mathrm{i}A)$ where A is a doubly-infinite banded Hermitian matrix. For the case A bounded, we have established an error estimate of the finite section method using the exponential decay property of $\exp(\mathrm{i}A)$. Such an error estimate naturally leads to a criterion for determining the computational window size in practical computation. The case when A is unbounded is much more difficult since existing techniques in approximation theory do not carry over. Therefore only several special classes of unbounded matrices, such as two classes of Wilkinson-type matrices, are studied. We have identified localized eigenvectors of finite diagonal blocks. Then an important technique used in estimating the error of the finite section method is to relate the error within the *finite* desired window to the decay property of a larger *finite* computational window, so that the difficulty of analyzing infinite matrices is avoided. We have also proposed an adaptive strategy for estimating the size of the computational window. This strategy works well even when a priori error estimates are too pessimistic or not easy to compute.

A Elementary Orthogonal Transformations

In the following we briefly recall two classes of orthogonal transformations—Householder reflections [79] and Givens rotations [55], which are extensively used in the QR algorithm discussed in Chapters 2 and 3. The results listed here can also be found in any standard textbook about numerical linear algebra (e.g., [58]).

A.1 Householder reflections

Let $w \in \mathbb{R}^N$ be a unit vector, i.e. $\|w\|_2 = 1$. Then the matrix $H_w = I - 2ww^T$ is called a *Householder reflection*. It can be easily verified that H_w is symmetric and orthogonal. In practice, applying a Householder reflection to a matrix A is accomplished through $H_w A \leftarrow A - 2w(w^T A)$ (or $AH_w \leftarrow A - 2(Aw)w^T$), which is much cheaper than explicitly performing a naive matrix-matrix multiplication. Therefore, applying a Householder reflection is a level 2 operation.

The main utility of Householder reflections in numerical linear algebra is to zero out all but the first entry in a given vector $x \in \mathbb{R}^N$, i.e., $H_w x = \alpha e_1$ for some $\alpha \in \mathbb{R}$. Notice that $2w(w^T x) = x - \alpha e_1$. The vector w is thus chosen as

$$w = \frac{x - \alpha e_1}{\|x - \alpha e_1\|_2}. \tag{A.1.1}$$

Since $|\alpha| = \|\alpha e_1\|_2 = \|H_w x\|_2 = \|x\|_2$, the only possible choices of α are $\alpha = \|x\|_2$ and $\alpha = -\|x\|_2$. To avoid numerical cancellation when computing $x - \alpha e_1$, we adopt

$$\alpha = \begin{cases} -\|x\|_2, & \text{if } x_1 \geq 0, \\ \|x\|_2, & \text{if } x_1 < 0, \end{cases}$$

and then use (A.1.1) to construct w.

Householder reflections are also used to construct an orthogonal matrix U such that Ue_1 is parallel to a prescribed nonzero vector x. We assume that U can be chosen as a Householder

reflection H_w with $H_w e_1 = \beta x$. Notice that $H_w x = \beta^{-1} e_1$. Thus this problem is reduced to the previous one—eliminate all but the first element in x.

The complex analogy is very similar—given a unit vector $w\mathbb{C}^N$, the matrix $H_w = I - 2ww^*$ is a unitary Hermitian matrix. To seek for α and w such that $H_w x = \alpha e_1$ for a given vector $x \in \mathbb{C}^N$, we choose

$$\alpha = \begin{cases} -\dfrac{x_1}{|x_1|}\|x\|_2, & \text{if } x_1 \neq 0, \\ \|x\|_2, & \text{if } x_1 = 0, \end{cases}$$

and still use (A.1.1) to construct w.

A.2 Givens rotations

A *Givens rotation* is an orthogonal matrix of the form

$$G(i,j,\theta) = \begin{array}{c} \\ i\text{th} \\ \\ j\text{th} \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & & i\text{th} & & j\text{th} \end{array} \\ \begin{bmatrix} I & & & & \\ & \cos\theta & & \sin\theta & \\ & & I & & \\ & -\sin\theta & & \cos\theta & \\ & & & & I \end{bmatrix} \end{array}, \qquad (\theta \in \mathbb{R}).$$

It is also known as a *Jacobi rotation* because it was used by Jacobi to solve symmetric eigenvalue problems [82]. A Givens rotation performs a linear combination of two rows (or columns) of a matrix. Therefore, applying a Givens rotation is a level 1 operation. To study the properties of Given rotations, it is sufficient to discuss the following 2×2 orthogonal matrix

$$G(\theta) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

where $c = \cos\theta$, $s = \sin\theta$.

Givens rotations provide another tool to perform orthogonal elimination. For example, for a given vector $[a,b]^T \in \mathbb{R}^2$, we seek for a Givens rotation $G(\theta)$ such that

$$G(\theta)^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}.$$

This is achieved by choosing $\cot\theta = -a/b$ when $b \neq 0$. In practice, the Givens rotation is

formed through

$$\begin{cases} \tau = -a/b,\ s = \dfrac{\tau}{\sqrt{1+\tau^2}},\ c = s\tau, & \text{if } |b| > |a|, \\ \tau = -b/a,\ c = \dfrac{\tau}{\sqrt{1+\tau^2}},\ s = c\tau, & \text{if } |b| \le |a|, \end{cases}$$

to avoid potential cancellations and overflow/underflow.

As mentioned in Chapter 2, swapping two eigenvalues in a real 2×2 upper triangular matrix can be performed by choosing suitable a Givens rotation. To solve the problem

$$G(\theta)^T \begin{bmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{bmatrix} G(\theta) = \begin{bmatrix} t_{22} & t_{12} \\ 0 & t_{11} \end{bmatrix},$$

we notice that $[t_{12}, t_{22}-t_{11}]^T$ and $[1,0]^T$ are the eigenvectors of $\left[\begin{smallmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} t_{22} & t_{12} \\ 0 & t_{11} \end{smallmatrix}\right]$, respectively, corresponding to the eigenvalue t_{22}. The Givens rotation satisfies

$$G(\theta)^T \begin{bmatrix} t_{12} \\ t_{22} - t_{11} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix}.$$

Thus, the problem of swapping eigenvalues is reduced to the problem of orthogonal elimination and is solved by choosing $\cot\theta = t_{12}/(t_{11} - t_{22})$.

A complex Givens rotation is a unitary matrix of the form

$$G = \begin{array}{c} \\ i\text{th} \\ \\ j\text{th} \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} \quad & i\text{th} & \quad & j\text{th} \end{array} \\ \begin{bmatrix} I & & & & \\ & c & & s & \\ & & I & & \\ & -s^* & & c & \\ & & & & I \end{bmatrix} \end{array}, \qquad c \in \mathbb{R}, \quad c^2 + |s|^2 = 1.$$

Complex Givens rotations are also used to eliminate elements in a given complex vector. For a nonzero vector $[a,b]^T \in \mathbb{C}^2$, the task

$$\begin{bmatrix} c & s \\ -s^* & c \end{bmatrix}^* \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix},$$

is achieved by choosing

$$c = \frac{|a|}{\sqrt{|a|^2 + |b|^2}}, \quad s = \frac{a}{|a|} \cdot \frac{b^*}{\sqrt{|a|^2 + |b|^2}}.$$

Consequently, the problem of swapping two consecutive eigenvalues in a (complex) Schur form is also solved.

B Basics in Rounding Error Analysis

This section recalls some basics in rounding error analysis for floating-point arithmetic. These basic results are used in Chapter 5. We refer to [71, 112, 164] for detailed discussions regarding floating-point arithmetic.

If there is no overflow or (gradual) underflow in the calculation, the rounding error is modeled by

$$\mathrm{fl}(\alpha \circ \beta) = (\alpha \circ \beta)(1+\epsilon), \qquad |\epsilon| \le \mathbf{u},$$

where "$\circ$" is $+$, $-$, $\times$, or $\div$, and $\mathbf{u}$ is the unit roundoff. In practice, the unit roundoff is $\mathbf{u} = 2^{-23} \approx 1.2 \times 10^{-7}$ in single precision arithmetic, and is $\mathbf{u} = 2^{-52} \approx 2.2 \times 10^{-16}$ in double precision arithmetic, on architectures following the IEEE-754 standard [80].

When rounding modes are switched to round *round towards* $-\infty$, the rounding error can be modeled as

$$\underline{\mathrm{fl}}(\alpha \circ \beta) = (\alpha \circ \beta)(1-\epsilon), \qquad 0 \le \epsilon \le \tilde{\mathbf{u}}.$$

Here the unit roundoff $\tilde{\mathbf{u}}$ is twice as large as the one under the standard rounding mode (i.e., *round towards nearest*) [112]. For example, $\tilde{\mathbf{u}}$ is 2^{-51} in double precision arithmetic under this biased rounding mode. Similarly, for the rounding mode *round towards* $+\infty$, we have

$$\overline{\mathrm{fl}}(\alpha \circ \beta) = (\alpha \circ \beta)(1+\epsilon), \qquad 0 \le \epsilon \le \tilde{\mathbf{u}}.$$

Both cases fit the rounding model

$$\mathrm{fl}(\alpha \circ \beta) = (\alpha \circ \beta)(1+\epsilon), \qquad |\epsilon| \le \tilde{\mathbf{u}}.$$

As a remark, the notation $\underline{\mathrm{fl}}(\cdot)$ and $\overline{\mathrm{fl}}(\cdot)$ used in this thesis does not have correspondence with any particular rounding mode. This notation merely implies that the computed quantity satisfies $\underline{\mathrm{fl}}(x) \le x$ and $\overline{\mathrm{fl}}(x) \ge x$. As an example, $\underline{\mathrm{fl}}(1-x)$ can be obtained via $-\mathrm{fl}(\mathrm{fl}(x)-1)$ under

the rounding mode *round towards* $+\infty$.

Let $X, Y \in \mathbb{R}^{N\times N}$. The following results are used in the rounding error analysis in Section 5.4. A direct consequence of the rounding model is that

$$|\mathrm{fl}(X+Y)-(X+Y)| \le \mathbf{u}|X+Y|.$$

It can be proved by induction that

$$\left|\mathrm{fl}\left(\sum_{k=1}^{N}\alpha_i\beta_i\right)-\sum_{k=1}^{N}\alpha_i\beta_i\right| \le \left[N\mathbf{u}+\mathcal{O}(\mathbf{u}^2)\right]\sum_{k=1}^{N}|\alpha_i\beta_i|.$$

Applying this result to individual entries of $\mathrm{fl}(XY)$ yields

$$|\mathrm{fl}(XY)-XY| \le \left[N\mathbf{u}+\mathcal{O}(\mathbf{u}^2)\right]|X||Y|.$$

C PDHSEQR User's Guide

C.1 Introduction

PDHSEQR is a parallel ScaLAPACK-style library for solving nonsymmetric eigenvalue problems. The library is written in Fortran 90 and targets distributed memory HPC systems. Using the small-bulge multishift QR algorithm with aggressive early deflation, it computes the real Schur decomposition $H = ZTZ^T$ of an upper Hessenberg matrix $H \in \mathbb{R}^{N\times N}$, such that Z is orthogonal and T is quasi-upper triangular. This document concerns the usage of PDHSEQR and is a supplement to the article [62]. For the description of the algorithm and implementation, we refer to [62] and the references therein (especially, [30, 31, 60, 61, 88]).

C.2 Installation

In the following, an installation guide is provided. It is assumed that the user is working in a Unix-like system.

C.2.1 Prerequisites

To build the library, the following software is required.

- A Fortran 90/95 compiler.
- The MPI library, e.g., OpenMPI or MPICH.
- An optimized BLAS library, e.g., ATLAS or OpenBLAS.
- The LAPACK library.
- The ScaLAPACK library (including BLACS and PBLAS).

C.2.2 How to compile the library

Download location.
The software version published by ACM TOMS can be downloaded from CALGO. [1] The latest version of the source code (with bug fixes) as well as the corresponding documents are available on the PDHSEQR homepage. [2]

Files in the tar-ball.
The following command unpacks the tar-ball and creates a directory `pdhseqr/`, which is the root directory of the library.

```
tar xzfv pdhseqr.tar.gz
```

Inside the root directory, there are several files and directories:

```
EXAMPLES/ MAKE_INC/ Makefile make.inc README SRC/ TESTING/ TOOLS/ TUNING/ ug.pdf
```

Below is an overview of these items.

- `EXAMPLES/` This directory contains two simple drivers.
- `MAKE_INC/` This directory contains several templates of `make.inc` for GNU, Intel, and PathScale compilers.
- `Makefile` The Makefile for building the library. This file does *not* need to be modified.
- `make.inc` *This is the only file which requires modifications when building the library.* It contains compiler settings and external libraries for the Makefile. The user is required to modify this file according to the target computational environment before compiling the library. Several templates of this file are provided in the directory `MAKE_INC`.
- `README` A shorter version of this document containing a quick installation guide.
- `SRC/` This directory contains source code for all computational routines of the library.
- `TESTING/` This directory contains testing examples.
- `TOOLS/` This directory contains several auxiliary routines (e.g., random number/matrix generators, input/output routines).
- `TUNING/` This directory contains auto-tuning scripts.
- `ug.pdf` The User's Guide of PDHSEQR.

Build the library.
Once `make.inc` is properly modified according to the computational environment, the library can be built by

```
make all
```

1. Collected algorithms of the ACM. See http://calgo.acm.org/.
2. PDHSEQR homepage: http://www8.cs.umu.se/~myshao/software/pdhseqr/.

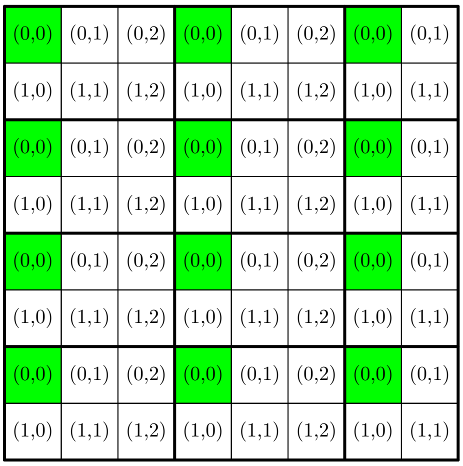


Figure C.1 – The 2D block-cyclic data layout across a 2×3 processor grid. For example, processor $(0,0)$ owns all highlighted blocks. Picture from [62].

in the root directory of PDHSEQR. This generates the library archive `libpdhseqr.a` in the root directory, two examples in `EXAMPLES/`, and test programs in `TESTING/`. Twelve quick tests in `TESTING/` are performed and (hopefully) the following result will be displayed on the screen:

```
% 7 out of 7 tests passed!
% 5 out of 5 tests passed!
```

This means that the Schur decomposition has been successfully computed for seven random matrices and five benchmark matrices, indicating that the compilation has been successful. We recommend that the script `runquick.sh` in `TESTING` is also run once to make sure that the parallel code works properly. You may need to modify the MPI execution command in this script according to your system (e.g., `mpirun`, `mpiexec`, etc.). If everything works out, 18 lines of information summarizing the 108 tests will be displayed and written to the file `summary.txt`. We also provide `runall.sh` with many large test cases in the same directory. (Running this set of tests may take *very long*!)

C.3 Using the package

C.3.1 ScaLAPACK data layout convention

In ScaLAPACK, the $p = p_r \cdot p_c$ processors are usually arranged into a $p_r \times p_c$ grid. Matrices are distributed over the rectangular processor grid in a *2D block-cyclic layout* with block size $M_b \times N_b$ (see Figure C.1 for an example). The information regarding the data layout is stored in an *array descriptor* so that the mapping between the entries of the global matrix and their corresponding locations in the memory hierarchy can be established. We adopt ScaLAPACK's data layout convention and require that the $N \times N$ input matrices H and Z have identical data layout with square data blocks (i.e., $M_b = N_b$). The processor grid, however, does not need to be square. A distributed matrix H is referenced by two arrays `H` (local matrix entries) and

DESCH (array descriptor). A typical setting of DESCH is listed below.

- DESCH(1): Type of the matrix. In our case, DESCH(1) = 1 since H is stored as a dense matrix.
- DESCH(2): The handle of the BLACS context.
- DESCH(3), DESCH(4): The size of H, i.e., DESCH(5) = DESCH(6) = N.
- DESCH(5), DESCH(6): Blocking factors M_b and N_b. We require that DESCH(5) = DESCH(6).
- DESCH(7), DESCH(8): The process row and column that contain h_{11}. Usually, DESCH(7) = DESCH(8) = 0.
- DESCH(9): Leading dimension of the local part of H on the current processor. This value needs to be at least one, even if the local part is empty.

C.3.2 Calling sequence

The main functionality of this package is to compute the real Schur decomposition of an upper Hessenberg matrix using the routine PDHSEQR. The ScaLAPACK routine PDGEHRD can be used to transform a general square matrix to Hessenberg form, see the test programs in TESTING/ for examples. The interface of PDHSEQR displayed below follows the convention of LAPACK/ScaLAPACK routines [6, 26].

```
      SUBROUTINE PDHSEQR( JOB, COMPZ, N, ILO, IHI, H, DESCH, WR, WI, Z,
     $                    DESCZ, WORK, LWORK, IWORK, LIWORK, INFO )
*
*     .. Scalar Arguments ..
      INTEGER            IHI, ILO, INFO, LWORK, LIWORK, N
      CHARACTER          COMPZ, JOB
*     ..
*     .. Array Arguments ..
      INTEGER            DESCH( * ) , DESCZ( * ), IWORK( * )
      DOUBLE PRECISION   H( * ), WI( N ), WORK( * ), WR( N ), Z( * )
```

For comparison, the (nearly identical) interface of the LAPACK routine DHSEQR.

```
      SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
     $                   LDZ, WORK, LWORK, INFO )
```

Also, the interface of the ScaLAPACK auxiliary routine PDLAHQR is similar.

```
      SUBROUTINE PDLAHQR( WANTT, WANTZ, N, ILO, IHI, A, DESCA, WR, WI,
     $                    ILOZ, IHIZ, Z, DESCZ, WORK, LWORK, IWORK,
     $                    ILWORK, INFO )
```

Therefore, it may not require much effort to switch existing code calling PDLAHQR to PDHSEQR.

An example for calling PDHSEQR is provided in the test program (TESTING/driver.f). We advice that PDHSEQR is called twice—the first call for performing a workspace query (by setting LWORK = −1 and the second call for actually performing the computation.

Below is a detailed list of the arguments.

- JOB: (global input) CHARACTER*1.
  JOB = 'E': Compute eigenvalues only;
  JOB = 'S': Compute eigenvalues and the Schur form T.
- COMPZ (global input) CHARACTER*1.
  COMPZ = 'N': Schur vectors (i.e., Z) are not computed;
  COMPZ = 'I': Z is initialized to the identity matrix and both H and Z are returned;
  COMPZ = 'V': Z must contain an orthogonal matrix Q on entry, and the product QZ is returned.
- N: (global input) INTEGER.
  The order of the Hessenberg matrix H (and Z).
- ILO, IHI: (global input) INTEGER.
  It is assumed that H is already upper triangular in rows and columns (1:ILO-1) and (IHI+1:N). They are normally set by a previous call to PDGEBAL, and then passed to PDGEHRD when the matrix output by PDGEBAL is reduced to Hessenberg form. Otherwise ILO = 1 and IHI = N should be used.
- H: (global input/output) DOUBLE PRECISION array of dimension (DESCH(9),*).
  DESCH: (global and local input) INTEGER array of dimension 9.
  H and DESCH define the distributed matrix H.
  On entry, H contains the upper Hessenberg matrix H.
  On exit, if JOB = 'S', H is quasi-upper triangular in rows and columns (ILO:IHI), with 1×1 and 2×2 blocks on the main diagonal. The 2×2 diagonal blocks (corresponding to complex conjugate pairs of eigenvalues) are returned in standard form, with $h_{ii} = h_{i+1,i+1}$ and $h_{i+1,i}h_{i,i+1} < 0$. If INFO = 0 and JOB = 'E', the contents of H are unspecified on exit.
- WR, WI: (global output) DOUBLE PRECISION array of dimension N.
  The eigenvalues of H(ILO:IHI,IHO:IHI) are stored in WR(ILO:IHI) and WI(ILO:IHI)—WR contains the real parts while WI contains the imaginary parts.
  If two eigenvalues are computed as a complex conjugate pair, they are stored in consecutive elements of WR and WI, say the i-th and (i+1)-th, with WI(i) > 0 and WI(i + 1) < 0.
  If JOB = 'S', the eigenvalues are stored in the same order as on the diagonal of the Schur form returned in H.
- Z: (global input/output) DOUBLE PRECISION array of dimension (DESCZ(9),*).
  DESCZ: (global and local input) INTEGER array of dimension 9.
  Z and DESCZ define the distributed matrix Z.
  If COMPZ = 'V', on entry Z must contain the current matrix Z of accumulated transformations from, e.g., PDGEHRD, and on exit Z has been updated.
  If COMPZ = 'N', Z is not referenced.
  If COMPZ = 'I', on entry Z does not need be set and on exit, if INFO = 0, Z contains the orthogonal matrix Z of the Schur vectors of H.

- WORK: (local workspace) DOUBLE PRECISION array of dimension LWORK.
  LWORK: (local input) INTEGER.
  In case LWORK = −1, a workspace query will be performed and on exit, WORK(1) is set to the required length of the double precision workspace. No computation is performed in this case.

- IWORK: (local workspace) INTEGER array of dimension LIWORK.
  LIWORK: (local input) INTEGER.
  In case LIWORK = −1, a workspace query will be performed and on exit, IWORK(1) is set to the required length of the integer workspace. No computation is performed in this case.

- INFO: (global output) INTEGER.
  If INFO = 0, PDHSEQR returns successfully.
  If INFO < 0, let $i = -$INFO, then the i-th argument had an illegal value.
  (See below for exceptions with $i = 7777$ or $i = 8888$.)
  If INFO > 0, then PDHSEQR failed to compute all of the eigenvalues. (This is a rare case.)
  Elements (1:ILO-1) and (INFO+1:N) of WR and WI contain the eigenvalues which have been successfully computed. Let U be the orthogonal matrix logically produced in the computation (regardless of COMPZ, i.e., no matter whether it is explicitly formulated or not). Then on exit,

$$\begin{cases} H_{in}U = U^T H_{out},\ Z = U, & \text{if INFO} > 0,\ \text{COMPZ} = 'I', \\ H_{in}U = U^T H_{out},\ Z_{out} = Z_{in}U, & \text{if INFO} > 0,\ \text{COMPZ} = 'V'. \end{cases}$$

  If INFO = 7777 or INFO = 8888, please send a bug report to the authors.

C.3.3 Example programs

We provide two simple examples in the directory EXAMPLES/. The program example1.f generates a 500×500 random matrix and computes its Schur decomposition, while example2.f reads the benchmark matrix OLM500 [3] in the Matrix Market format [9].

To compute eigenvalues of other matrices, the following lines of the example program need to be adjusted:

- Line 40: Matrix size and the block factor.

- Line 50–51: Make sure to provide sufficient memory.

- Line 222: Replace the matrix generator PDMATGEN2/PQRRMMM by your own matrix.

3. Downloaded from ftp://math.nist.gov/pub/MatrixMarket2/NEP/olmstead/olm500.mtx.gz.

C.4 Tuning the parameters*

The instructions below are intended for experienced users. Other users may want to skip reading this section.

In `SRC/pilaenvx.f` and `SRC/piparmq.f`, there are several machine-dependent parameters, see [62] for details. On contemporary architectures, we expect that most of the default values provided in the source code yield reasonable performance. However, for `PILAENVX(ISPEC = 12, 14, 23)` some fine tuning might be helpful. The package offers two scripts in the directory `TUNING/` that aim at tuning these three parameters.

Before starting the tuning procedure, you first need to choose a frequently used block factor (N_b) and modify the corresponding value in `tune1.in`, `tune2_1.in`, `tune2_2.in`, and `tune2_3.in`. You also need to adjust the MPI execution command according to your system in the scripts `tune1.sh` and `tune2.sh`.

The first script `tune1.sh` searches suitable settings for `PILAENVX(ISPEC = 12, 23)`. It performs the tests described in `tune1.in` on 1×1, 2×2, 4×4, 8×8 processor grids, and analyzes the collected data by the code `tune1.f`. This procedure usually takes 1–4 hours. When completed, it reports suggestions on the parameters in the file `suggestion1.txt`. You should then modify the constants `NMIN` (Line 193 in `SRC/piparmq.f`) and `NTHRESH` (Line 648 in `SRC/pilaenvx.f`) in accordance with these suggestions.

The second script `tune2.sh` searches suitable settings for `PILAENVX(ISPEC = 14)`. This set of tests should only be done after running `tune1.sh` and modifying the parameters in `SRC/pilaenvx.f`, `SRC/piparmq.f`, and `TUNING/piparmq.f` correspondingly. Then the library should be compiled again with the new settings:

```
make clean; make all; make tuning
```

Finally, the script `tune2.sh` is executed. This set of tests takes a long time (up to 1–2 days). If all tests are completed successfully, `tune2.f` computes and reports the suggested settings for the parameters in `suggestions2.txt`. You should then update `SRC/piparmq.f` (Line 197) and rebuild the library. This completes the tuning procedure.

If some of the tests are interrupted, due to an error, it may not be necessary to rerun the whole set of tests. You can also manually collect the execution times for each test into `summary2.txt` and apply `tune2.f` to determine the parameter suggestions.

It is possible to run both tuning scripts on other processor grids (besides the default 2×2, 4×4, 8×8). This, however, requires to not only adjust the scripts `tune*.sh` but also the programs `tune*.f` correspondingly.

C.5 Terms of usage

Use of the ACM Algorithm is subject to the ACM Software Copyright and License Agreement. [4] Furthermore, any use of the PDHSEQR library should be acknowledged by citing the corresponding paper [62]. Depending on the context, the citation of the papers [60, 61, 88] is also encouraged.

4. See http://www.acm.org/publications/policies/softwarecrnotice.

Bibliography

[1] E. Agullo, J. W. Demmel, J. J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180(1):012037, 2009.

[2] L. V. Ahlfors. *Complex Analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill, Inc., New York, NY, USA, 3rd edition, 1979.

[3] M. Ahues and F. Tisseur. A new deflation criterion for the QR algorithm. Technical report, 1997. LAPACK Working Note 122.

[4] N. I. Akhiezer and I. M. Glazman. *Theory of Linear Operators in Hilbert Space*. Dover Publications, New York, NY, USA, 1993.

[5] A. S. Alfa, J. Xue, and Q. Ye. Accurate computation of the smallest eigenvalue of a diagonally dominant M-matrix. *Math. Comp.*, 71(237):217–236, 2002.

[6] E. Anderson, Z. Bai, C. H. Bischof, L. S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3rd edition, 1999.

[7] M. Arioli, B. Codenotti, and C. Fassino. The Padé method for computing the matrix exponential. *Linear Algebra Appl.*, 240:111–130, 1996.

[8] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Kraemer, B. Lang, H. Lederer, and P. R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Comput.*, 37(12):783–794, 2011.

[9] Z. Bai, D. Day, J. J. Demmel, and J. J. Dongarra. A test matrix collection for non-Hermitian eigenvalue problems (release 1.0). Technical Report CS-97-355, Department of Computer Science, University of Tennessee, 1997. Also available online from http://math.nist.gov/MatrixMarket.

[10] Z. Bai and J. W. Demmel. On a block implementation of Hessenberg multishift QR iteration. *Int. J. High Speed Comput.*, 1(1):97–112, 1989. Also as LAPACK Working Note 8.

[11] Z. Bai and J. W. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, part I. In R. Sincovec, D. Keyes, M. Leuze, L. Petzold, and D. Reed, editors, *Parallel Processing for*

Scientific Computing, pages 391–398, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[12] Z. Bai and J. W. Demmel. On swapping diagonal blocks in real Schur form. *Linear Algebra Appl.*, 186:73–95, 1993.

[13] G. A. Baker, Jr. and P. Graves-Morris, editors. *Padé Approximants*, volume 59 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, New York, NY, USA, 2nd edition, 1996.

[14] R. Baker Kearfott and V. Kreinovich, editors. *Applications of Interval Computations*, volume 3 of *Applied Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[15] G. Ballard, J. W. Demmel, and I. Dumitriu. Minimizing communication for eigenproblems and the singular value decomposition. Technical Report UCB/EECS-2010-136, EECS Department, University of California, Berkeley, 2010. Also as LAPACK Working Note 237.

[16] G. Ballard, J. W. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.

[17] J. L. Barlow and J. W. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal.*, 27(3):762–791, 1990.

[18] R. H. Bartels and G. W. Stewart. Algorithm 432: Solution of the matrix equation $AX + XB = C$ [F4]. *Comm. ACM*, 15(9):820–826, 1972.

[19] M. Benzi and P. Boito. Decay properties for functions of matrices over C^*-algebras. Technical report, 2013. Available from http://arxiv.org/abs/1307.7119.

[20] M. Benzi and G. H. Golub. Bounds for the entries of matrix functions with applications to preconditioning. *BIT*, 39(3):417–438, 1999.

[21] M. Benzi and N. Razouk. Decay bounds and $O(n)$ algorithms for approximating functions of sparse matrices. *Electron. Trans. Numer. Anal.*, 28:16–39, 2007.

[22] A. Berman and R. J. Plemmons. *Nonnegative matrices in the Mathematical Sciences*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.

[23] P. J. Bickel and M. Lindner. Approximating the inverse of banded matrices by banded matrices with applications to probability and statistics. *Theory Probab. Appl.*, 56(1):1–20, 2012.

[24] P. Bientinesi, E. S. Quintana-Ortí, and R. A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application program interfaces. *ACM Trans. Math. Software*, 31(1):27–59, 2005.

[25] N. Biggs. *Algebraic Graph Theory*. Cambridge University Press, Cambridge, UK, 2nd edition, 1993.

[26] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. W. Demmel, I. Dhillon, J. J. Dongarra, S. J. Hammarling, G. Henry, A. Petitet, K. Stanley, D. W. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[27] P. Bochev and S. Markov. A self-validating numerical method for the matrix exponential. *Computing*, 43:59–72, 1989.

[28] A. Böttcher. C^*-algebra in numerical analysis. *Irish Math. Soc. Bulletin*, 45:57–133, 2000.

[29] K. Braman. Middle deflations in the QR algorithm, 2008. Talk at Householder Symposium XVII, Zeuthen, Germany.

[30] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[31] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[32] R. Byers. LAPACK 3.1 `xHSEQR`: Tuning and implementation notes on the small bulge multi-shift QR algorithm with aggressive early deflation. Technical report, 2007. LAPACK Working Note 187.

[33] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numer. Algorithms*, 10(2):379–399, 1995.

[34] B. Codenotti and C. Fassino. Error analysis of two algorithms for the computation of the matrix exponential. *Calcolo*, 29(1):1–31, 1992.

[35] C. Cohen-Tannoudji, B. Diu, and F. Laloe. *Quantum Mechanics*. Wiley-VCH, Berlin, Germany, 1992.

[36] R. F. Curtain and H. J. Zwart. *An Introduction to Infinite-Dimensional Linear Systems Theory*. Springer-Verlag, New York, NY, USA, 1995.

[37] K. Dackland and B. Kågström. An hierarchical approach for performance analysis of ScaLAPACK-based routines using the distributed linear algebra machine. In J. Waśniewski, J. J. Dongarra, K. Madsen, and D. Olesen, editors, *Applied Parallel Computing Industrial Computation and Optimization (PARA 1996)*, volume 1184 of *Lecture Notes in Comput. Sci.*, pages 186–195, Heidelberg, Germany, 1996. Springer-Verlag.

[38] P. I. Davies and N. J. Higham. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[39] N. Del Buono, L. Lopez, and R. Peluso. Computation of the exponential of large sparse skew-symmetric matrices. *SIAM J. Sci. Comput.*, 27(1):278–293, 2005.

[40] S. Demko. Inverse of band matrices and local convergence of spline projections. *SIAM J. Numer. Anal.*, 14:616–619, 1977.

[41] S. Demko, W. F. Moss, and P. W. Smith. Decay rates for inverses of band matrices. *Math. Comp.*, 43:491–499, 1984.

[42] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[43] L. Deng and J. Xue. Accurate computation of exponentials of triangular essentially non-negative matrices. *J. Fudan U. (Nat. Sci.)*, 50(1):78–86, 2011.

[44] J. Dixmier. *C∗-Algebras*. North-Holland Publishing Company, Amsterdam, the Netherlands, 1977.

[45] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.*, 27(1–2):215–227, 1989. Also as LAPACK Working Note 2.

[46] E. Estrada and D. J. Higham. Network properties revealed through matrix functions. *SIAM Rev.*, 52(4):696–714, 2010.

[47] M. R. Fahey. Algorithm 826: A parallel eigenvalue routine for complex Hessenberg matrices. *ACM Trans. Math. Software*, 29(3):326–336, 2003.

[48] C. Fassino. *Computation of Matrix Functions*. PhD thesis, University of Pisa, 1993.

[49] J. G. F. Francis. The QR transformation: A unitary analogue to the LR transformation—Part 1. *Comput. J.*, 4(3):265–271, 1961.

[50] J. G. F. Francis. The QR transformation—Part 2. *Comput. J.*, 4(4):332–345, 1962.

[51] A. Frommer and V. Simoncini. Stopping criteria for rational matrix functions of Hermitian and symmetric matrices. *SIAM J. Sci. Comput.*, 30(3):1387–1412, 2008.

[52] F. R. Gantmacher. *The Theory of Matrices, Volumes I and II*. AMS Chelsea Publishing, Providence, RI, USA, 1959.

[53] I. M. Gelfand. Normierte Ringe. *Rec. Math. [Mat. Sbornik] N. S.*, 9(51):3–24, 1941.

[54] D. Gill and E. Tadmor. An $O(N^2)$ method for computing the eigensystem of $N \times N$ symmetric tridiagonal matrices by the divide and conquer approach. *SIAM J. Sci. Stat. Comput.*, 11(1):161–173, 1990.

[55] W. Givens. Computation of plain unitary rotations transforming a general matrix to triangular form. *J. Soc. Indust. Appl. Math.*, 6(1):26–50, 1958.

[56] A. Goldsztejn. On the exponentiation of interval matrices. Technical report, 2009. Available from http://arxiv.org/abs/0908.3954.

[57] G. H. Golub and F. Uhlig. The QR algorithm: 50 years later its genesis by John Francis and Vera Kublanovskaya and subsequent developments. *IMA J. Numer. Anal.*, 29(3):467–485, 2009.

[58] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

[59] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, Boston, MA, USA, 2nd edition, 2003.

[60] R. Granat, B. Kågström, and D. Kressner. Parallel eigenvalue reordering in real Schur forms. *Concurrency and Computat.: Pract. Exper.*, 21(9):1225–1250, 2009. Also as LAPACK Working Note 192.

[61] R. Granat, B. Kågström, and D. Kressner. A novel parallel QR algorithm for hybrid distributed memory HPC systems. *SIAM J. Sci. Comput.*, 32(4):2345–2378, 2010. Also as LAPACK Working Note 216.

[62] R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. Technical Report UMINF-12.06, Department of Computing Science and HPC2N, Umeå University, 2012. Also as Technical Report MATHICSE Nr. 31.2013. Available from http://www8.cs.umu.se/research/uminf/.

[63] V. Grimm. Resolvent Krylov subspace approximation to operator functions. *BIT*, 52(3):639–659, 2012.

[64] L. Gurvits, R. Shorten, and O. Mason. On the stability of switched positive linear systems. *IEEE Trans. Automat. Control*, 52(6):1099–1103, 2007.

[65] R. Hagen, S. Roch, and B. Silbermann. *C^*-Algebras and Numerical Analysis*, volume 236 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker, Inc., New York, NY, USA, 2001.

[66] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer-Verlag, Berlin and Heidelberg, Germany, 2nd edition, 1996.

[67] G. Hargreaves. *Topics in Matrix Computations: Stability and Efficiency of Algorithms*. PhD thesis, University of Manchester, 2005.

[68] D. E. Heller and I. C. F. Ipsen. Systolic networks for orthogonal equivalence transformations and their applications. In *Proceedings of the Conference on Advanced Research in VLSI*, pages 113–122, MIT, Cambridge, MA, USA, 1982.

[69] G. Henry and R. A. van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: Myths and reality. *SIAM J. Sci. Comput.*, 17(4):870–883, 1996.

[70] G. Henry, D. S. Watkins, and J. J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. *SIAM J. Sci. Comput.*, 24(1):284–311, 2002. Also as LAPACK Working Note 121.

[71] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[72] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[73] N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[74] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Rev.*, 51(4):747–764, 2009.

[75] M. Hochbruck and A. Ostermann. Exponential integrators. *Acta Numer.*, 19:209–286, 2010.

[76] M. Hoemmen. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, University of California, Berkeley, 2010.

[77] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.

[78] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991.

[79] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comput. Mach.*, 5(4):339–342, 1958.

[80] IEEE Computer Society. IEEE standard for floating-point arithmetic, 2008. DOI: 10.1109/IEEESTD.2008.4610935.

[81] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distr. Com.*, 64(9):1017–1026, 2004.

[82] C. G. J. Jacobi. Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. *J. Reine Angew. Math.*, 30:51–94, 1846.

[83] M. E. C. Jordan. *Traité des substitutions et des équations algébriques*. Gauthier-Villars, Paris, France, 1870.

[84] B. Kågström. Bounds and perturbation bounds for the matrix exponential. *BIT*, 17(1):39–57, 1977.

[85] B. Kågström. Numerical computation of matrix functions. Technical Report UMINF-58.77, Department of Information Processing, Umeå University, 1977.

[86] B. Kågström. How to compute the Jordan normal form—the choice between similarity transformations and methods using the chain relations. Technical Report UMINF-91.81, Department of Information Processing, Umeå University, 1981.

[87] B. Kågström, D. Kressner, E. S. Quintana-Ortí, and G. Quintana-Ortí. Blocked algorithms for the reduction to Hessenberg-triangular form revisited. *BIT*, 48(3):563–584, 2008.

[88] B. Kågström, D. Kressner, and M. Shao. On aggressive early deflation in parallel variants of the QR algorithm. In K. Jónasson, editor, *Applied Parallel and Scientific Computing (PARA 2010)*, volume 7133 of *Lecture Notes in Comput. Sci.*, pages 1–10, Berlin and Heidelberg, Germany, 2012. Springer-Verlag.

[89] B. Kågström and A. Ruhe. Algorithm 563: JNF, an algorithm for the numerical computation of the Jordan normal form of a complex matrix [F2]. *ACM Trans. Math. Software*, 6(3):437–443, 1980.

[90] B. Kågström and A. Ruhe. An algorithm for the numerical computation of the Jordan normal form of a complex matrix. *ACM Trans. Math. Software*, 6(3):389–419, 1980.

[91] L. Karlsson and B. Kågström. Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures. *Parallel Comput.*, 37(12):771–782, 2011.

[92] L. Karlsson, D. Kressner, and B. Lang. Optimally packed chains of bulges in multishift QR algorithms. *ACM Trans. Math. Software*, 2013, to appear. Also as LAPACK Working Note 271.

[93] T. Kato. *Perturbation Theory for Linear Operators*. Springer-Verlag, Berlin, Germany, 1980.

[94] L. Knizhnerman and V. Simoncini. A new investigation of the extended Krylov subspace method for matrix function evaluations. *Numer. Linear Algebra Appl.*, 17(4):615–638, 2010.

[95] D. Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*, volume 46 of *Lecture Notes Comput. Sci. Eng.* Springer-Verlag, Heidelberg, Germany, 2005.

[96] D. Kressner. Block algorithms for reordering standard and generalized Schur forms. *ACM Trans. Math. Software*, 32(4):521–532, 2006. Also as LAPACK Working Note 171.

[97] D. Kressner. The effect of aggressive early deflation on the convergence of the QR algorithm. *SIAM J. Matrix Anal. Appl.*, 30(2):805–821, 2008.

[98] I. Krishtal, T. Strohmer, and T. Wertz. Localization of matrix factorizations. Technical report, 2013. Available from http://arxiv.org/abs/1305.1618.

[99] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Comp. Math. Math. Phys.*, 1(3):637–657, 1961.

[100] B. Lang. Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift, Bergische Universität Gesamthochschule Wuppertal, 1997.

[101] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software*, 5(3):308–323, 1979.

[102] M. Lindner. *Infinite Matrices and Their Finite Sections: An Introduction to the Limit Operator Method*. Frontiers in Mathematics. Birkhäuser, Basel, Switzerland, 2006.

[103] M. L. Liou. A novel method of evaluating transient response. *Proc. IEEE*, 57(5):20–23, 1966.

[104] G. G. Lorentz. *Approximation of Functions*. AMS Chelsea Publishing, Providence, RI, USA, 2nd edition, 2005.

[105] R. S. Martin, G. Peters, and J. H. Wilkinson. Handbook Series Linear Algebra: The QR algorithm for real Hessenberg matrices. *Numer. Math.*, 14(3):219–231, 1970.

[106] E. J. Mastascusa. A relation between Liou's method and the fourth-order Runge-Kutta method for evaluating transient response. *Proc. IEEE*, 57(5):803–804, 1969.

[107] B. J. Melloy and G. K. Bennett. Computing the exponential of an intensity matrix. *J. Comput. Appl. Math.*, 46(3):405–413, 1993.

[108] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.

[109] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[110] R. E. Moore, R. Baker Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[111] L. Moreau. Stability of continuous-time distributed consensus algorithms. In *43rd IEEE Conference on Decision and Control*, volume 4, pages 3998–4003, 2004.

[112] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, Boston, MA, USA, 2010.

[113] F. D. Murnaghan and A. Wintner. A canonical form for real matrices under orthogonal transformations. *Proc. Natl. Acad. Sci. USA*, 17(7):417–420, 1931.

[114] Y. Nakatsukasa. Eigenvalue perturbation bounds for Hermitian block tridiagonal matrices. *Appl. Numer. Math.*, 62:67–78, 2012.

[115] M. J. Parks. *A Study of Algorithms to Compute the Matrix Exponential*. PhD thesis, University of California at Berkeley, 1994.

[116] B. N. Parlett. A recurrence among the elements of functions of triangular matrices. *Linear Algebra Appl.*, 14:117–121, 1976.

[117] B. N. Parlett. *The Symmetric Eigenvalue Problem*, volume 20 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998. Corrected reprint of the 1980 original.

[118] B. N. Parlett and J. Le. Forward instability of tridiagonal QR. *SIAM J. Matrix Anal. Appl.*, 14(1):274–316, 1993.

[119] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[120] L. Prylli and B. Tourancheau. Fast runtime block cyclic data redistribution on multiprocessors. *J. Parallel Distr. Com.*, 45(1):63–72, 1997.

[121] G. Quintana-Ortí and R. A. van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Trans. Math. Software*, 32(2):180–194, 2006.

[122] M. Reed and B. Simon. *Functional Analysis*, volume I of *Methods of Modern Mathematical Physics*. Academic Press, London, UK, 1972.

[123] M. Reed and B. Simon. *Fourier Analysis, Self-Adjointness*, volume II of *Methods of Modern Mathematical Physics*. Academic Press, London, UK, 1975.

[124] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. R. Soc. Lond. A*, 210:307–357, 1911.

[125] A. Ruhe. Eigenvalues of a complex matrix by the QR-method. *BIT*, 6(4):350–358, 1966.

[126] H. Rutishauser. Une méthode pour la détermination des valeurs propres d'une matrices. *C. R. Acad. Sci.*, 240(1):214–235, 1955.

[127] H. Rutishauser. Solution of eigenvalue problems with the LR-transformation. *Nat. Bur. Stand. Appl. Math. Ser.*, 49:47–81, 1958.

[128] Y. Saad. *Numerical Methods for Large Eigenvalue Problems, Revised Edition*, volume 66 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

[129] T. Schreiber, P. Otto, and F. Hofmann. A new efficient parallelization strategy for the QR algorithm. *Parallel Comput.*, 20(1):63–75, 1994.

[130] I. Schur. Über die charakteristischen Wurzeln einer linearen Substitution mit einer Anwendung auf die Theorie der Integralgleichungen. *Math. Ann.*, 66:488–510, 1909.

[131] I. Schur. Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen. *J. Reine Angew. Math.*, 140:1–28, 1911.

[132] M. Shao. `PDLAQR1`: An improved version of the ScaLAPACK routine `PDLAHQR`. Technical Report UMINF-11.22, Department of Computing Science and HPC2N, Umeå University, 2011. Available from http://www8.cs.umu.se/research/uminf/.

[133] M. Shao. On the finite section method for computing exponentials of doubly-infinite skew-Hermitian matrices. Technical Report Nr. 26.2013, MATHICSE, EPF Lausanne, 2013. Available from http://mathicse.epfl.ch/page-68906-en.html.

[134] M. Shao, W. Gao, and J. Xue. Aggressively truncated Taylor series method for accurate computation of exponentials of essentially nonnegative matrices. Technical Report UMINF-12.04, Department of Computing Science and HPC2N, Umeå University, 2012. Available from http://www8.cs.umu.se/research/uminf/.

[135] P. N. Shivakumar and C. Ji. Upper and lower bounds for inverse elements of finite and infinite tridiagonal matrices. *Linear Algebra Appl.*, 247:297–316, 1996.

[136] R. B. Sidje. Inexact uniformization and GMRES methods for large Markov chains. *Numer. Linear Algebra Appl.*, 18:947–960, 2011.

[137] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines — EISPACK Guide*, volume 6 of *Lecture Notes in Comput. Sci.* Springer-Verlag, New York, NY, USA, 2nd edition, 1976.

[138] M. Stadelmann. Matrixfunktionen—Analyse und Implementierung. Master's thesis, ETH Zürich, 2009.

[139] C. J. Standish. Truncated Taylor series approximation to the state transition matrix of a continuous parameter finite Markov chain. *Linear Algebra Appl.*, 12(2):179–183, 1975.

[140] G. W. Stewart. A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. *SIAM J. Sci. Stat. Comput.*, 6(4):853–864, 1985.

[141] G. W. Stewart. A parallel implementation of the QR algorithm. *Parallel Comput.*, 5:187–196, 1987.

[142] G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614, 2001.

[143] G. W. Stewart. *Matrix Algorithms, Volume II: Eigensystems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[144] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, USA, 1994.

[145] A. Taylor and D. J. Higham. CONTEST: A controllable test matrix toolbox for MATLAB. *ACM Trans. Math. Software*, 35(4):26:1–26:17, 2009.

[146] S. Tomov, R. Nath, and J. J. Dongarra. Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Comput.*, 36(12):645–654, 2010.

[147] R. A. van de Geijn. *Implementing the QR Algorithm on an Array of Processors*. PhD thesis, University of Maryland, College Park, 1987.

[148] R. A. van de Geijn. Deferred shifting schemes for parallel QR methods. *SIAM J. Matrix Anal. Appl.*, 14(1):180–194, 1993.

[149] R. A. van de Geijn and D. G. Hudson. An efficient parallel implementation of the nonsymmetric QR algorithm. In *Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 697–700, Monterey, CA, 1989.

[150] R. A. van de Geijn and J. Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency and Computat.: Pract. Exper.*, 9(4):255–274, 1997. Also as LAPACK Working Note 96.

[151] C. F. Van Loan. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal.*, 14(6):971–981, 1977.

[152] R. S. Varga. *Geršgorin and His Circles*, volume 36 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, Germany, 2004.

[153] C. Vömel and B. N. Parlett. Detecting localization in an invariant subspace. *SIAM J. Sci. Comput.*, 33(6):3447–3467, 2011.

[154] G. Walz. Computing the matrix exponential and other matrix functions. *J. Comput. Appl. Math.*, 21(1):119–123, 1988.

[155] R. C. Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal.*, 14(4):600–610, 1977.

[156] D. S. Watkins. Shifting strategies for the parallel QR algorithm. *SIAM J. Sci. Comput.*, 15(4):953–958, 1994.

[157] D. S. Watkins. Forward stability and transmission of shifts in the QR algorithm. *SIAM J. Matrix Anal. Appl.*, 16(2):469–487, 1995.

[158] D. S. Watkins. The transmission of shifts and shift blurring in the QR algorithm. *Linear Algebra Appl.*, 241–243:877–896, 1996.

[159] D. S. Watkins. A case where balancing is harmful. *Electron. Trans. Numer. Anal.*, 23:1–4, 2006.

[160] D. S. Watkins. The QR algorithm revisited. *SIAM Rev.*, 50(1):133–145, 2008.

[161] D. S. Watkins and L. Elsner. Convergence of algorithms of decomposition type for the eigenvalue problem. *Linear Algebra Appl.*, 143:19–47, 1991.

[162] D. E. Whitney. More about similarities between Runge-Kutta methods for evaluating transient response. *Proc. IEEE*, 57(11):2053–2054, 1969.

[163] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, New York, NY, USA, 1965.

[164] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Dover Publications, New York, NY, USA, 1994.

[165] J. Xue and E. Jiang. Entrywise relative perturbation theory for nonsingular M-matrices and applications. *BIT*, 35(3):417–427, 1995.

[166] J. Xue and Q. Ye. Entrywise relative perturbation bounds for exponentials of essentially non-negative matrices. *Numer. Math.*, 110(3):393–403, 2008.

[167] J. Xue and Q. Ye. Computing exponentials of essentially non-negative matrices entrywise to high relative accuracy. *Math. Comp.*, 82:1577–1596, 2013.

[168] W. Zhu, J. Xue, and W. Gao. The sensitivity of the exponential of an essentially nonnegative matrix. *J. Comput. Math.*, 26(2):250–258, 2008.

Curriculum Vitae

Personal information

Name:	Shao, Meiyue
Date of birth:	1983/12/08
Place of birth:	Shanghai, China
Nationality:	Chinese
Language:	Chinese, English

Education

since 2012/05	École Polytechnique Fédérale de Lausanne, Switzerland Ph.D. student in Numerical Analysis Research topics: Parallel dense eigensolvers and matrix exponentials
2009/08–2012/04	Umeå University, Sweden Ph.Lic. in Computing Science Research topics: Parallel dense eigensolvers and matrix exponentials
2008/09–2009/02	Lawrence Berkeley National Laboratory, United States Visiting student Research topic: Factorization-based preconditioners
2006/09–2009/06	Fudan University, China M.A. in Computational Mathematics Research topics: Factorization-based preconditioners and numerical solution of integral equations
2002/09–2006/06	Fudan University, China B.S. in Computational Mathematics

Software Development

PDHSEQR	Parallel library software for the multishift QR algorithm with aggressive early deflation. (An intermediate version is available in ScaLAPACK v2.0.) Joint work with Robert Granat, Bo Kågström, and Daniel Kressner.
SuperLU	Sequential and parallel libraries to solve unsymmetric sparse linear systems using LU factorization. Joint work with Xiaoye S. Li et al.
FEMKS	Parallel library for solving Kohn-Sham equations using finite element methods. Joint work with Dier Zhang et al.

Awards

2002–2008	Scholarships by Fudan University
2005	Successful Participant in Mathematical Contest in Modeling
2004	Third Prize in China National Mathematical Contest in Modeling

Publications

[1] M. Shao. *Parallel Variants and Library Software for the QR Algorithm and the Computation of the Matrix Exponential of Essentially Nonnegative Matrices*, Licentiate Thesis, Umeå University, April 2012. (Supervised by Bo Kågström and Daniel Kressner)

[2] M. Shao. A supernodal approach to incomplete LU factorizations, Master's Thesis (in Chinese), Fudan University, June 2009. (Supervised by Weiguo Gao and Xiaoye S. Li)

[3] M. Shao. On the finite section method for computing exponentials of doubly-infinite skew-Hermitian matrices. Technical Report Nr. 26.2013, MATHICSE, EPF Lausanne, 2013. (Submitted to *Linear Algebra Appl.*)

[4] D. Kressner, M. Miloloža Pandur, and M. Shao. An indefinite variant of LOBPCG for definite matrix pencils. *Numer. Algorithms*, 2013 (to appear).

[5] Q. Xie and M. Shao. An elementary problem and techniques in linear algebra. (In Chinese) *Stud. in College Math.*, 16(4): 53–55, 2013.

[6] R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. Technical Report UMINF-12.06, Department of Computing Science and HPC2N, Umeå University, 2012. Also as Technical Report Nr. 31.2013, MATHICSE, EPF Lausanne, 2013. (Submitted to *ACM Trans. Math. Software*)

[7] M. Shao, W. Gao, and J. Xue. Aggressively truncated Taylor series method for accurate computation of exponentials of essentially nonnegative matrices. Technical Report UMINF-12.04, Department of Computing Science and HPC2N, Umeå University, 2012. (Submitted to *SIAM J. Matrix Anal. Appl.*)

[8] B. Kågström, D. Kressner, and M. Shao. On aggressive early deflation in parallel variants of the QR algorithm. In *Applied Parallel and Scientific Computing (PARA 2010)*, K. Jónasson (Ed.). *Lecture Notes in Comput. Sci.*, LNCS-7133, Springer-Verlag, Berlin, Heidelberg, Germany, 1–10, 2012.

[9] X. S. Li and M. Shao. A supernodal approach to incomplete LU factorizations with partial pivoting. *ACM Trans. Math. Software*, 37(4): 43:1–43:20, 2011.

[10] M. Shao. `PDLAQR1`: An improved version of the ScaLAPACK routine `PDLAHQR`. Technical Report UMINF-11.22, Department of Computing Science and HPC2N, Umeå University, 2011.

[11] M. Shao, W. Gao, and Y. Zhang. Robust projection method for Fredholm integral equation of the second kind. *Int. J. Comput. Math.*, 87(15): 3455–3466, 2010.

[12] X. S. Li, M. Shao, I. Yamazaki, and E. G. Ng. Factorization-based sparse solvers and preconditioners. *Journal of Physics: Conference Series* 180(2009) 012015, Institute of Physics Publishing, San Diego, CA, USA, 2009.