

COMPUTING MATRIX FUNCTIONS IN ARBITRARY PRECISION ARITHMETIC

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2019

Massimiliano Fasi
School of Mathematics

CONTENTS

LIST OF FIGURES	4
LIST OF TABLES	6
ABSTRACT	7
DECLARATION	8
COPYRIGHT STATEMENT	9
ACKNOWLEDGEMENTS	10
PUBLICATIONS	11
1 INTRODUCTION	12
Bibliography	16
2 BACKGROUND MATERIAL	21
2.1 Linear algebra	21
2.2 Floating point arithmetic	34
Bibliography	43
3 OPTIMALITY OF THE PATERSON-STOCKMEYER METHOD	47
3.1 Introduction	47
3.2 Evaluation of matrix polynomials	53
3.3 Rational matrix functions of order $[k/k]$	59
3.4 Diagonal Padé approximants to the matrix exponential	63
3.5 Conclusion	69
Bibliography	69
4 SOLUTION OF PRIMARY MATRIX EQUATIONS	73
4.1 Introduction	73
4.2 Background and notation	75
4.3 Classification of the solutions	79
4.4 A substitution algorithm	88
4.5 Numerical experiments	100
4.6 Conclusions	105
Bibliography	106
5 MULTIPRECISION ALGORITHMS FOR THE MATRIX EXPONENTIAL	108
5.1 Introduction	109
5.2 Padé approximation of matrix exponential	113
5.3 A multiprecision algorithm	120
5.4 Numerical experiments	132
5.5 Conclusions	139
Bibliography	140

6	MULTIPRECISION ALGORITHMS FOR THE MATRIX LOGARITHM	145
6.1	Introduction	146
6.2	Support for multiple precision arithmetic	150
6.3	Approximation of hypergeometric functions	151
6.4	Schur–Padé algorithm	154
6.5	Transformation-free algorithm	160
6.6	Numerical experiments	162
6.7	Conclusions	170
	Bibliography	171
7	WEIGHTED GEOMETRIC MEAN TIMES A VECTOR	176
7.1	Introduction	176
7.2	Notation and preliminaries	179
7.3	Quadrature methods	181
7.4	Krylov subspace methods	192
7.5	Computing $(A\#_t B)^{-1}v$	199
7.6	Numerical tests	200
7.7	Conclusions	211
	Bibliography	213
8	CONCLUSIONS	218
	Bibliography	219

LIST OF FIGURES

- Figure 3.1 Number of matrix multiplications required to evaluate a polynomial of degree k , for k between 1 and 50, by means of the scheme (3.3) with $s = \lfloor \sqrt{k} \rfloor$ and $s = \lceil \sqrt{k} \rceil$. The dotted and dashed lines mark the values of k that are integer multiples of $\lfloor \sqrt{k} \rfloor$ and $\lceil \sqrt{k} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate polynomials of optimal degree (in the sense of Definition 3.1) for the Paterson–Stockmeyer method. 54
- Figure 3.2 Number of matrix multiplications required to evaluate a rational function of order $\lfloor k/k \rfloor$, for k between 1 and 50, by means of the scheme (3.15), for $s = \lfloor \sqrt{2k} \rfloor$ and $s = \lceil \sqrt{2k} \rceil$. The dotted and dashed lines mark the values of k that are integer multiples of $\lfloor \sqrt{2k} \rfloor$ and $\lceil \sqrt{2k} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate rational matrix functions of optimal order (in the sense of Definition 3.1) for the evaluation scheme (3.15). 59
- Figure 3.3 Number of matrix multiplications required to evaluate $[k/k]$ Padé approximant to the matrix exponential, for k between 1 and 50, by means of the scheme (3.20), for $s = \lfloor \sqrt{k-1/2} \rfloor$ and $s = \lceil \sqrt{k-1/2} \rceil$. The dotted and dashed lines mark the values of k for which $\frac{k-1}{2}$ is an integer multiple of $\lfloor \sqrt{k-1/2} \rfloor$ and $\lceil \sqrt{k-1/2} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate the diagonal Padé approximants to the matrix exponential of optimal order (in the sense of Definition 3.1) for the evaluation scheme (3.20). 64
- Figure 4.1 Relative forward error of methods for the solution of rational matrix equations. 102
- Figure 4.2 Forward error of the Schur algorithm. 104
- Figure 5.1 Forward error of algorithms for the matrix exponential in double precision. 134
- Figure 5.2 Forward error of algorithms for the matrix exponential in arbitrary precision. 136
- Figure 6.1 Comparison of old and new bounds on the forward error of diagonal Padé approximants to the matrix logarithm. 154
- Figure 6.2 Forward error and computational cost of algorithms for the matrix logarithm in double precision. 165
- Figure 6.3 Forward error to unit roundoff ratio. 167
- Figure 6.4 Forward error of algorithms for the matrix logarithm in arbitrary precision. 168
- Figure 7.1 Parameters of convergence of Gaussian quadrature formulae for the inverse matrix square root. 204
- Figure 7.2 Convergence profiles for quadrature and Krylov methods for the geometric mean. 205

- Figure 7.3 Clusters in the adjacency matrices of the Wikipedia RfA signed network. [209](#)
- Figure 7.4 Execution time of algorithms for solving $(A\#_t B)x = v$. [211](#)

LIST OF TABLES

Table 4.1	Solutions to the equation $r(X) = A$ in Test 4.2.	104
Table 5.1	Timings of algorithms for the matrix exponential in quadruple precision.	138
Table 6.1	Execution time of algorithms for the matrix logarithm in high precision.	169
Table 7.1	Summary of algorithms for the matrix geometrix mean used in the experiments.	202
Table 7.2	Summary of matrices used in the experiments.	207
Table 7.3	Execution time of methods for the weighted geometric mean.	207

ABSTRACT

Functions of matrices arise in numerous applications, and their accurate and efficient evaluation is an important topic in numerical linear algebra. In this thesis, we explore methods to compute them reliably in arbitrary precision arithmetic: on the one hand, we develop some theoretical tools that are necessary to reduce the impact of the working precision on the algorithmic design stage; on the other, we present new numerical algorithms for the evaluation of primary matrix functions and the solution of matrix equations in arbitrary precision environments.

Many state-of-the-art algorithms for functions of matrices rely on polynomial or rational approximation, and reduce the computation of $f(A)$ to the evaluation of a polynomial or rational function at the matrix argument A . Most of the algorithms developed in this thesis are no exception, thus we begin our investigation by revisiting the Paterson–Stockmeyer method, an algorithm that minimizes the number of nonscalar multiplications required to evaluate a polynomial of a certain degree. We introduce the notion of optimal degree for an evaluation scheme, and derive formulae for the sequences of optimal degree for the schemes used in practice to evaluate truncated Taylor and diagonal Padé approximants.

If the rational function r approximates f , then it is reasonable to expect that a solution to the matrix equation $r(X) = A$ will approximate the functional inverse of f . In general, infinitely many matrices can satisfy this kind of equation, and we propose a classification of the solutions that is of practical interest from a computational standpoint. We develop a precision-oblivious numerical algorithm to compute all the solutions that are of interest in practice, which behaves in a forward stable fashion.

After establishing these general techniques, we concentrate on the matrix exponential and its functional inverse, the matrix logarithm. We present a new scaling and squaring approach for computing the matrix exponential in high precision, which combines a new strategy to choose the algorithmic parameters with a bound on the forward error of Padé approximants to the exponential. Then, we develop two algorithms, based on the inverse scaling and squaring method, for evaluating the matrix logarithm in arbitrary precision. The new algorithms rely on a new forward error bound for Padé approximants, which for highly nonnormal matrices can be considerably smaller than the classic bound of Kenney and Laub. Our experimental results show that in double precision arithmetic the new approaches are comparable with the state-of-the-art algorithm for computing the matrix logarithm, and experiments in higher precision support the conclusion that the new algorithms behave in a forward stable way, typically outperforming existing alternatives.

Finally, we consider a problem of the form $f(A)b$, and focus on methods for computing the action of the weighted geometric mean of two large and sparse positive definite matrices on a vector. We present two new approaches based on numerical quadrature, and compare them with several methods based on the Krylov subspace in terms of both accuracy and efficiency, and show which algorithms are better suited for a black-box approach. In addition, we show how these methods can be employed to solve a problem that arises in applications, namely the solution of linear systems whose coefficient matrix is a weighted geometric mean.

DECLARATION

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

COPYRIGHT STATEMENT

- i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420>), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.library.manchester.ac.uk/about/regulations>) and in The University’s Policy on Presentation of Theses.

ACKNOWLEDGEMENTS

I am very grateful to Michele Benzi and Stefan Güttel for reading the preliminary draft of this thesis and providing much useful feedback that improved the quality and coherence of the presentation.

It is my pleasure to express my gratitude to my supervisor, Nicholas J. Higham, for his guidance, his scientific contribution to my doctoral project, and his precious feedback on my work. I really appreciated the freedom to pursue, without pressure, any research topic I found interesting, and the leeway to collaborate with other researchers in Manchester as well as at other institutions.

My deep appreciation goes also to Bruno Iannazzo, for the many pleasant and productive hours spent working together during my many visits at the University of Perugia, for his abundant advice, and for all the effort he put into making our collaboration a success.

Part of this work was carried out while I was visiting other institutions. I thank Elias Jarlebring, Giampaolo Mele, and Emil Ringh at the KTH Royal Institute of Technology, Vanni Noferini at the University of Essex, Francesco Tudisco at the University of Strathclyde, and Andrii Dmytryshyn at the University of Örebro, for their great hospitality and for many fruitful discussions.

Many postgraduate students provided insightful comments on my work. I am particularly grateful to Bahar Arslan, Mario Berljafa, Michael Connolly, Steven Elsworth, Jiao Liu, Xiaobo Liu, Jacopo Marcheselli, Thomas McSweeney, Matteo Monti, Gian Maria Negri Porzio, Craig Newsum, Filippo Pagani, Emanuele Penocchio, Paul Rus-sel, Matteo Tamiozzo, and Mantė Žemaitytė.

Funding from the MathWorks, the University of Manchester, and the Royal Society is also gratefully acknowledged.

PUBLICATIONS

- Chapter 3 is based on the journal article: Massimiliano Fasi. **Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions.** *Linear Algebra Appl.*, 574:182–200, 2019.
- Chapter 4 is based on the journal article: Massimiliano Fasi and Bruno Iannazzo. **Computing primary solutions of equations involving primary matrix functions.** *Linear Algebra Appl.*, 560:17–42, 2019.
- Chapter 5 is based on the preprint: Massimiliano Fasi and Nicholas J. Higham. **An arbitrary precision scaling and squaring algorithm for the matrix exponential.** MIMS EPrint 2018.36, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2018. Submitted to *SIAM J. Matrix Anal. Appl.*
- Chapter 6 is based on the journal article: Massimiliano Fasi and Nicholas J. Higham. **Multiprecision algorithms for computing the matrix logarithm.** *SIAM J. Matrix Anal. Appl.*, 39(1):472–491, 2018.
- Chapter 7 is based on the journal article: Massimiliano Fasi and Bruno Iannazzo. **Computing the weighted geometric mean of two large-scale matrices and its inverse times a vector.** *SIAM J. Matrix Anal. Appl.*, 39(1):178–203, 2018.

1

INTRODUCTION

Linear algebra is among the oldest branches of mathematics. The earliest examples of simultaneous linear equations appear in two Babylonian tablets, VAT 8389 and 8391,¹ which contain exercises dealing with the computation of the rent of a field divided in two parts. The most comprehensive treatment regarding the solution of linear systems known to antiquity is in the eighth of *The Nine Chapters on the Mathematical Arts*, a collection of problems with solutions, where the coefficients of linear systems are arranged in rectangular arrays of numbers [12] and a method similar to Gaussian elimination is discussed [42].

In the western world, the theory of linear systems did not advance much until the end of the 17th century [34, Chap. 5], and Gauss was the first to introduce, albeit implicitly, matrices as a compact form for writing linear transformations [10, sect. V]. The term “matrix” was first used by Sylvester [43, p. 369], and a major contribution to the genesis of matrix analysis was given by Cayley, the first to realise that matrices are mathematical objects in their own right, and can be studied as single quantity rather than a set of coefficients.

In his seminal paper [5], Cayley defines matrix addition, multiplication, and inversion, introduces the zero and identity matrices, and records a number of observations. In his exposition, Cayley deals with rather small matrices. For instance, he proves what is now known as the Cayley–Hamilton theorem only for 2×2 matrices, deeming it unnecessary “to undertake the labour of a formal proof of the theorem in the general case of a matrix of any degree”, but assuring the reader that the result remains valid in the 3×3 case. Similarly, when discussing integer powers of a matrix he deduces a trigonometric formula for the n th power of a matrix of order 2, and after observing that the formula extends to negative or fractional values of n , he remarks

¹ The transcription from cuneiform and a translation in English are given by Høyrup [25, Chap. III].

that the case $n = 1/2$ defines the *square root* of matrix. He will later return to matrix functions, and devote a whole paper to the symbolic extraction of square roots of 3×3 matrices [6].

As for transcendental matrix functions, Laguerre is the first to discuss, en passant in a paragraph of a long letter to Hermite [36], the exponential of a matrix. In fact, the French mathematician confines himself to defining the exponential of a square matrix via the series expansion of e^z at 0 and mentioning that the scalar identity $e^{x+y} = e^x e^y$ does not generalise to matrices. In the following few years several authors proposed general formulae for generalising scalar functions to matrices, and we refer the interested reader to [23, sect. 1.10] for a detailed account.

Just over 10 years after Laguerre's definition, Peano [38] shows how the matrix exponential can be used to solve systems of homogeneous differential equations, demonstrating that matrix functions can have applications to the solution of theoretical problems. Frazer, Duncan, and Collar [9] are among the first to recognize the importance of matrix functions in practical applications, and in particular the central role of the exponential in the solution of systems of differential equations arising in engineering problems. A succinct account of classical applications of matrix functions can be found in [23, Chap. 2]. More recent applications include the analysis of complex networks [7, Chap. 6–8] and quantum chemistry [1].

While matrix functions are as old as matrix algebra itself, multiple precision arithmetic is a relatively young research area even in the much younger field of computer arithmetic. While the first theoretical model of an electro-mechanical machine capable of implementing floating-point [39] predates the first world war, and Konrad Zuse's Z3 computer, the first modern implementation, was realised during the second, software for computing in arbitrary precision was not available until the end of the seventies, when Brent released the first version of the MP library [3], [4], a Fortran package for multi-precision computation. The interest in such computing environments has been growing steadily since, and has exploded in recent years, thanks to the emergence of applications needing a range of precisions larger than that al-

lowed by the IEEE standard [30] and the considerable performance improvements in software for arbitrary precision computation.

Even though the research literature focusing on the efficient implementation of linear algebra subroutines for arbitrary precision arithmetic is growing quickly, the computation of matrix functions in low and high precision arithmetic has not yet attracted much attention. In this thesis, we satisfy the need for multiprecision algorithms for computing matrix functions, and propose techniques that are well suited for arbitrary precision environments. The main challenge we have to face is the fact that the working precision at which the algorithm is going to be run is known only at runtime and, as a consequence, should be treated as an input argument to the algorithm rather than a characteristic of the computational environment itself.

In fact, several algorithms for evaluating matrix functions in double precision can be adapted to arbitrary precision environments with little or no modifications. A notable example of these precision oblivious algorithms are substitution methods for the solution of matrix equations of the form $X^p = A$, where A is a square complex matrix and p is an integer greater than 1. These methods, such as the state-of-the-art algorithms for the square root [2], [19] and the p th root [41], [13], [29], are “direct”, in that they resemble forward and backward substitution for the solution of linear systems, and the logic of the algorithm need not change as long as routines to perform scalar sums, multiplications, divisions, and p th root extraction at the target precision are available.

Other algorithms can be adapted to multiprecision environments with just minor adjustments. A typical instance are iterative algorithms, which can be run in arbitrary precision by simply executing all elementary scalar operations in arbitrary precision and adjusting the internal tolerance that is used as stopping criterion. Iterative methods have been the object of an intense study for almost forty years, and new developments are still under way. Examples include Newton’s method, which has been studied for the computation of the polar decomposition [17, sect. 3.2-3.4], [32], the sign function [31], [32], [40], the square root [18], [21] [26], the p th root [15], [16], [27], and the Lambert W function [8], and Halley’s method, which has been developed

for the polar decomposition [37], the sector function [35], the p th root [15], and the Lambert W function [8]. These two methods belong to more general families of Padé iterations, that have been discussed in the literature for the polar decomposition [20], [33], the sector function [11], the sign function [14], [31], [33], and the p th root [11], [28].

Many state-of-the-art algorithms, however, follow a different approach, and their design relies heavily on the knowledge of the working precision at which the algorithm will be run. These algorithms reduce the computation of a function of a matrix to the evaluation of a polynomial or rational approximant—typically a Padé approximant—at a matrix argument. The order of the approximant is determined by using a set of constants that specify how small the 1-norm of powers of the matrix must be in order for the approximant to deliver full accuracy at a given precision. These precision-dependent constants are computed offline by combining an upper bound on the backward error of the approximants with a mix of symbolic and high precision computation. This technique was originally proposed by Higham [22], [24] for the computation of the matrix exponential in double precision, and provides very efficient algorithms for several matrix functions, at the price of a computationally expensive algorithm design stage.

In fact, the analysis of Al-Mohy and Higham can be repeated for any fixed precision, but the precision-dependent computation is too expensive to make it a viable strategy for arbitrary precision algorithms. For computing the exponential of a matrix, on-the-fly estimation of the backward error has been proposed as a technique to bound the truncation error of Taylor approximants at runtime, but this approach does not appear to generalize easily to other Padé approximants, and does not readily extend to other matrix functions.

In this thesis, we seek a more systematic exploration of the subject. Our contribution is twofold. On the one hand, we revisit general techniques for evaluating polynomials and rational functions of matrices and for solving polynomial and rational matrix equations, and give a precision-independent view of state-of-the-art techniques. On the other hand, we develop numerical algorithms for solving specific

problems pertaining to matrix functions, such as the evaluation of the exponential and logarithm of a matrix and the computation of the action of the weighted geometric mean of two Hermitian positive definite matrices on a vector.

Chapter 2 summarizes the main definitions and properties that are needed in the remainder of the thesis but are not given in the introductory sections of the following chapters. In order to reduce repetitions, for material that is introduced later on only a reference to the relevant section is provided. Since the thesis is in journal format, a few background topics are discussed in more than one place: in this case we provide a reference to the most complete review available.

Chapters 3, 4, 5, 6, and 7 are presented in a format suitable for publication and are based on the preprints and journal papers listed on page 11. For coauthored papers, we believe that the two authors contributed equally to the final manuscript and that it is not necessary to further discriminate their contribution.

BIBLIOGRAPHY

- [1] M. BENZI, P. BOITO, AND N. RAZOUK, *Decay properties of spectral projectors with applications to electronic structure*, SIAM Rev., 55 (2013), p. 3–64.
- [2] Å. BJÖRCK AND S. HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
- [3] R. P. BRENT, *Algorithm 524: MP, a Fortran multiple-precision arithmetic package [A1]*, ACM Trans. Math. Software, 4 (1978), p. 71–81.
- [4] ———, *A Fortran multiple-precision arithmetic package*, ACM Trans. Math. Software, 4 (1978), p. 57–70.
- [5] A. CAYLEY, *A memoir on the theory of matrices*, Philos. Trans. Roy. Soc. London, 148 (1858), p. 17–37.
- [6] ———, *On the extraction of the square root of a matrix of the third order*, Proc. Roy. Soc. Edinburgh, 7 (1872), p. 675–682.

- [7] E. ESTRADA, *The Structure of Complex Networks: Theory and Applications*, Oxford University Press, New York, 2011.
- [8] M. FASI, N. J. HIGHAM, AND B. IANNAZZO, *An algorithm for the matrix Lambert W function*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 669–685.
- [9] R. A. FRAZER, W. J. DUNCAN, AND A. R. COLLAR, *Elementary Matrices and Some Applications to Dynamics and Differential Equations*, Cambridge University Press, 1938. 1963 printing.
- [10] C. F. GAUSS, *Disquisitiones Arithmeticae*, Gerhard Fleischer, 1801.
- [11] O. GOMILKO, D. B. KARP, M. LIN, AND K. ZIĘTAK, *Regions of convergence of a Padé family of iterations for the matrix sector function and the matrix p th root*, J. Comput. Appl. Math, 236 (2012), p. 4410–4420.
- [12] J. F. GRUAR, *How ordinary elimination became Gaussian elimination*, Hist. Math., 38 (2011), p. 163–218.
- [13] F. GRECO AND B. IANNAZZO, *A binary powering Schur algorithm for computing primary matrix roots*, Numer. Algorithms, 55 (2010), pp. 59–78.
- [14] F. GRECO, B. IANNAZZO, AND F. POLONI, *The Paddé iterations for the matrix sign function and their reciprocals are optimal*, Linear Algebra Appl., 436 (2012), p. 472–477.
- [15] C.-H. GUO, *On Newton's method and Halley's method for the principal p th root of a matrix*, Linear Algebra Appl., 432 (2010), p. 1905–1922.
- [16] C.-H. GUO AND N. J. HIGHAM, *A Schur–Newton method for the matrix p th root and its inverse*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 788–804.
- [17] N. J. HIGHAM, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.
- [18] ———, *Newton's method for the matrix square root*, Math. Comp., 46 (1986), pp. 537–549.

- [19] —, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.
- [20] —, *The matrix sign decomposition and its relation to the polar decomposition*, Linear Algebra Appl., 212/213 (1994), pp. 3–20.
- [21] —, *Stable iterations for the matrix square root*, Numer. Algorithms, 15 (1997), pp. 227–242.
- [22] —, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 1179–1193.
- [23] —, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [24] —, *The scaling and squaring method for the matrix exponential revisited*, SIAM Rev., 51 (2009), pp. 747–764.
- [25] J. HØYRUP, *Lengths, widths, surfaces: a portrait of Old Babylonian algebra and its kin*, Springer-Verlag, New York, 1st ed., 2002.
- [26] B. IANNAZZO, *A note on computing the matrix square root*, Calcolo, 40 (2003), p. 273–283.
- [27] —, *On the Newton method for the matrix p th root*, SIAM J. Matrix Anal. Appl., 28 (2006), p. 503–523.
- [28] —, *A family of rational iterations and its application to the computation of the matrix p th root*, SIAM J. Matrix Anal. Appl., 30 (2009), p. 1445–1462.
- [29] B. IANNAZZO AND C. MANASSE, *A Schur logarithmic algorithm for fractional powers of matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 794–813.
- [30] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

- [31] C. KENNEY AND A. J. LAUB, *Rational iterative methods for the matrix sign function*, SIAM J. Matrix Anal. Appl., 12 (1991), p. 273–291.
- [32] ———, *On scaling Newton’s method for polar decomposition and the matrix sign function*, SIAM J. Matrix Anal. Appl., 13 (1992), p. 688–706.
- [33] A. KIEŁBASIŃSKI, P. ZIELIŃSKI, AND K. ZIĘTAK, *On iterative algorithms for the polar decomposition of a matrix and the matrix sign function*, Appl. Math. Comput., 270 (2015), p. 483–495.
- [34] I. KLEINER, *A History of Abstract Algebra*, Birkhäuser, Boston, MA, USA, 2007.
- [35] C. KOC AND B. BAKKALOGLU, *Halley’s method for the matrix sector function*, IEEE Trans. Automat. Control, 40 (1995), p. 944–949.
- [36] E. N. LAGUERRE, *Le calcul des systèmes linéaires, extrait d’une lettre adressé à M. Hermite*, in *Oeuvres de Laguerre, C. Hermite, H. Poincaré, and E. Rouché*, eds., vol. 1, Gauthier–Villars, Paris, 1898, pp. 221–267. The article is dated 1867 and is “Extrait du Journal de l’École Polytechnique, LXII^e Cahier”.
- [37] Y. NAKATSUKASA, Z. BAI, AND F. GYGI, *Optimizing Halley’s iteration for computing the matrix polar decomposition*, SIAM J. Matrix Anal. Appl., 31 (2010), p. 2700–2720.
- [38] G. PEANO, *Intégration par séries des équations différentielles linéaires*, Math. Annalen, 32 (1888), p. 450–456.
- [39] B. RANDELL, *From analytical engine to electronic digital computer: The contributions of Ludgate, Torres, and Bush*, IEEE Ann. Hist. Comput., 4 (1982), p. 327–341.
- [40] J. D. ROBERTS, *Linear model reduction and solution of the algebraic Riccati equation by use of the sign function*, Internat. J. Control, 32 (1980), p. 677–687.
- [41] M. I. SMITH, *A Schur algorithm for computing matrix p th roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989.
- [42] P. D. STRAFFIN, *Liu Hui and the first golden age of Chinese mathematics*, Math. Mag., 71 (1998), p. 163.

- [43] J. J. SYLVESTER, *XLVII. Additions to the articles in the September number of this journal, "On a new class of theorems," and on Pascal's theorem*, London, Edinburgh Dublin Philos. Mag. J. Sci., 37 (1850), p. 363–370.

2 | BACKGROUND MATERIAL

This chapter serves three main purposes. First, we establish the notation and terminology that will be adopted in the following chapters, and succinctly discuss some fundamental concepts. Secondly, we collect here, for ease of reference, most of the theoretical results that will be necessary later on. We do not claim that our review is complete, on the contrary, we stress that it should be used only as a reference, thus we provide pointers to the relevant literature for the interested reader. In the last section, we review briefly software packages and libraries that provide multiple precision capabilities, paying specific attention to the availability of linear algebra kernels and algorithms for the computation of matrix functions.

2.1 LINEAR ALGEBRA

Sets and functions. The empty set is denoted by \emptyset , the integers by \mathbb{Z} , the set of natural numbers by $\mathbb{N} = \{n \in \mathbb{Z} : n \geq 0\}$, the set of positive integers by \mathbb{N}_0 , and the set of consecutive integers between n_1 and n_2 by $\llbracket n_1, n_2 \rrbracket := \{n \in \mathbb{N} : n_1 \leq n \leq n_2\}$. The fields of real and complex numbers are denoted by \mathbb{R} and \mathbb{C} , respectively, and the orthogonal projections of a complex number z onto the real and imaginary axis are denoted by $\operatorname{Re}(z)$ and $\operatorname{Im}(z)$, respectively.

The notation $f: D \rightarrow C$ expresses the fact that the function f maps elements of the domain D to elements of the codomain C . We denote by $\mathbb{C}[z]$ the set of polynomials of the complex variable z with complex coefficients, and by $\mathbb{C}_\ell[z] \subset \mathbb{C}[z]$ the complex polynomials of degree at most ℓ . Let $p \in \mathbb{C}_k[z]$ and $q \in \mathbb{C}_m[z]$ be polynomials with nonzero leading coefficients. If p and q are coprime, that is, have no roots in common, we say that the quotient $r := p/q$ is a rational function of type $[k/m]$.

In order to map reals to integers, we use the *floor* function $\lfloor x \rfloor = \max \{n \in \mathbb{Z} : n \leq x\}$ and the *ceiling* function $\lceil x \rceil = \min \{n \in \mathbb{Z} : n \geq x\}$, defined for any $x \in \mathbb{R}$. With δ_{ij} we denote the Kroenecker delta, defined, for $i, j \in \mathbb{N}$, by

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Vector spaces and subspaces. Let \mathbb{F} be a field [1, (2.3) Def.]. A set V equipped with the two operations $+: V \times V \rightarrow V$ and $\cdot: \mathbb{F} \times V \rightarrow V$ is a *vector space* over \mathbb{F} if V is closed with respect to these two operations, that is if $u + v \in V$ and $\alpha \cdot v \in V$ for all $u, v \in V$ and $\alpha \in \mathbb{F}$, and the following properties are verified:

1. $u + v = v + u$ for all $u, v \in V$;
2. $(u + v) + w = u + (v + w)$ for all $u, v, w \in V$;
3. there exists $0 \in V$ such that $0 + v = v + 0 = v$;
4. there exists $\tilde{v} \in V$ such that $v + \tilde{v} = \tilde{v} + v = 0$ for all $v \in V$;
5. $\alpha \cdot (u + v) = \alpha \cdot u + \alpha \cdot v$ for all $u, v \in V$ and $\alpha \in \mathbb{F}$;
6. $(\alpha + \beta) \cdot v = \alpha \cdot v + \beta \cdot v$ for all $v \in V$ and $\alpha, \beta \in \mathbb{F}$;
7. $c \cdot (\beta \cdot v) = (c\beta) \cdot v$ for all $v \in V$ and $\alpha, \beta \in \mathbb{F}$;
8. $1 \cdot v = v$ for all $v \in V$, where $1 \in \mathbb{F}$ is the identity element of \cdot .

The set $W \subset V$ is a *subspace* of V if it is a vector space with respect to the operations $+$ and \cdot . It can be shown that a subspace is nonempty, as it always contains the zero vector.

Matrices and vectors. Let \mathbb{F} be a field. The vector space of $m \times n$ *matrices* over \mathbb{F} , that is, the set of m -by- n arrays of scalars from \mathbb{F} , is denoted by $\mathbb{F}^{m \times n}$. Matrices in $\mathbb{F}^{1 \times n}$ and $\mathbb{F}^{m \times 1}$ are called *row* and *column vectors*, respectively. In the former case, we often omit the second dimension and write \mathbb{F}^m to denote $\mathbb{F}^{m \times 1}$, to remark that \mathbb{F}^m can be identified with the m -dimensional vector space over \mathbb{F} .

We use capital Latin or Greek letters to denote matrices, and refer to their elements by the corresponding lowercase letter followed by one or more subscript indices that indicate the position of that element relatively to a row/column grid. If we want to refer to an element of an expression that produces a matrix, we enclose the expression in parenthesis and use the subscript notation.

Let $V = \{v_1, v_2, \dots, v_r\}$ be a set of m -dimensional vectors. The vector $y \in \mathbb{F}^m$ is a *linear combination* of the elements in V if it can be expressed in the form

$$y = \sum_{i=1}^r \alpha_i v_i,$$

for some $\alpha_1, \alpha_2, \dots, \alpha_r \in \mathbb{F}$. The vector space of all such linear combinations is called the *span* of V , which we denote by $\text{span } V$, and V is called a set of *generators* for $\text{span } V$. If a vector $v \in V$ belongs to $\text{span } V \setminus \{v\}$, then the vectors in V are *linearly dependent*, otherwise V is *linearly independent*. Note that a set of vectors containing the 0 vector is always linearly dependent. The dimension of the vector space $\text{span } V$ is the number of linearly independent vectors in V .

The sum of two matrices $A, B \in \mathbb{F}^{m \times n}$ is defined entry-wise, that is, $C := A + B$ is the m -by- n matrix with elements $c_{ij} = a_{ij} + b_{ij}$. Two matrices $A \in \mathbb{F}^{m \times n}$ and $B \in \mathbb{F}^{m' \times n'}$ are called *conformable* if $n = m'$, and the product of two conformable matrices $A \in \mathbb{F}^{m \times n}$ and $B \in \mathbb{F}^{n \times p}$ is the matrix $C := AB \in \mathbb{F}^{m \times p}$ with elements

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

This definition of the matrix product may seem rather arbitrary at first glance, but comes naturally when looking at matrices as linear transformations. In fact, by viewing the matrix $A \in \mathbb{F}^{m \times n}$ as the linear transformation from \mathbb{F}^n to \mathbb{F}^m that maps $x \in \mathbb{F}^n$ to the vector $Ax \in \mathbb{F}^m$, one can interpret the sum of two matrices as the sum of the corresponding linear transformations, and their product as their composition. We will not examine this equivalence further here, and will confine ourselves to a practical definition of range and null space of $A \in \mathbb{F}^{m \times n}$. The *range* of $A \in \mathbb{F}^{m \times n}$ is $\text{range } A = \{Ax : x \in \mathbb{F}^n\}$, and its null space is $\text{null } A = \{x \in \mathbb{F}^n : Ax = 0\}$.

A detailed discussion of matrices as linear operators can be found, for example, in [32, sect. 2.6], [3, Chap. 3], and [14, Chap. 2]. Alternative but equivalent definitions of the matrix multiplication above are discussed by Trefethen and Bau [35, Lect. 1] and Golub and Van Loan [15, sect. 1.1]. The *rank* of A , denoted by $\text{rank } A$, is the number of linearly independent rows (or, equivalently, columns) of A , and it is easy to see that $\text{range } A$ has dimension $\text{rank } A$. A matrix $A \in \mathbb{F}^{m \times n}$ is full-rank if it has the largest possible rank for matrices of that size, that is, if $\text{rank } A = \min\{m, n\}$.

In the following chapters, we focus on real and complex matrices, that is, we assume that our field \mathbb{F} is either \mathbb{R} or \mathbb{C} , and are mostly concerned with *square* matrices, for which $m = n$. The additive identity element of the vector space $\mathbb{C}^{n \times n}$ is the *zero matrix*, whose entries are all zero, and its multiplicative identity element is the *identity matrix* $I_n \in \mathbb{C}^{n \times n}$, whose elements are defined by $i_{ij} = \delta_{ij}$. In both cases, we omit the order n whenever it is clear from the context. Note that in general two square matrices do not commute. The inverse of a matrix $A \in \mathbb{C}^{n \times n}$, denoted by A^{-1} , is a matrix such that $AA^{-1} = A^{-1}A = I_n$. The matrix A need not have an inverse, but if it does then the inverse is unique. A matrix is called *nonsingular* if it has a matrix inverse, *singular* otherwise. Several conditions are equivalent to being nonsingular; for example, a matrix A is nonsingular if and only if $\text{null } A = \{0\}$, that is, if the only solution to the linear equation $Ax = 0$ is $x = 0$.

Let $A \in \mathbb{C}^{m \times n}$. We denote by A^T the *transpose* of A , that is, the n -by- m matrix whose element in position (i, j) is the element in position (j, i) of A , and by $A^* = \overline{A}^T$ its *conjugate transpose*, where \overline{A} denotes the matrix whose elements are the complex conjugate of the corresponding elements of A . It is easy to show that for two conformable complex matrices A and B , one has that $\overline{AB} = \overline{A} \overline{B}$, but $(AB)^T = B^T A^T$ and $(AB)^* = B^* A^*$.

Structured matrices. It is sometimes convenient to block together the elements of a matrix in order to work at the block level. A matrix whose elements are partitioned according to a block pattern is called a *block matrix*, and we typically assume that

rows and columns follow the same partitioning, which guarantees that the blocks along the diagonal are square.

A matrix $A \in \mathbb{F}^{m \times n}$ is diagonal if $a_{ij} = 0$ when $i \neq j$. As a diagonal matrix is fully determined by the elements along its diagonal, we denote the diagonal matrix with elements a_1, \dots, a_n by $\text{diag}(a_1, \dots, a_n)$. If $a_{ij} = 0$ when $i > j$ or $i < j$, the matrix A is upper-triangular or lower-triangular, respectively. Triangular matrices with $a_{ii} = 1$ for $i = 1, \dots, \min\{m, n\}$ are called unit triangular. A matrix is upper-Hessenberg (lower-Hessenberg) if $a_{ij} = 0$ for $i > j + 1$ ($i < j - 1$). Block diagonal and block triangular matrices can be defined analogously, by replacing elements with blocks in the definitions above. A block triangular matrix $A \in \mathbb{F}^{n \times n}$ is *quasi-triangular* if its diagonal blocks have size at most 2.

Transposition and conjugate transposition define several important classes of matrices. We say that a matrix $A \in \mathbb{C}^{n \times n}$ is *symmetric*, *skew-symmetric*, *Hermitian*, *skew-Hermitian*, or *normal*, if $A = A^T$, $A = -A^T$, $A = A^*$, $A = -A^*$, or $A^* = A^*A$, respectively. Note that symmetric and Hermitian matrices are normal.

Trace and Determinant. We use two functions to map complex square matrices to complex scalars, the trace and the determinant. The *trace* of a matrix $A \in \mathbb{C}^{n \times n}$, denoted by $\text{tr } A$ is the sum of its diagonal elements, that is,

$$\text{tr } A = \sum_{i=1}^n a_{ii}.$$

It is immediate to see that $\text{tr } A^T = \text{tr } A$ and $\text{tr } A^* = \overline{\text{tr } A}$. Moreover, for two matrices $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{n \times m}$, one has that

$$\text{tr}(AB) = \text{tr}(BA) = \sum_{i=1}^n \sum_{j=1}^m a_{ij} b_{ji}.$$

Finally, we stress that the trace is a linear operator, that is, if $n = m$ then

$$\text{tr}(\alpha A + \beta B) = \alpha \text{tr } A + \beta \text{tr } B,$$

for any $\alpha, \beta \in \mathbb{C}$.

The *determinant* of a matrix $A \in \mathbb{C}^{n \times n}$ can be defined in several ways. Here we opt for the recursive definition by means of the Laplace expansion, which can be found for example in [14, sect. 4.2]. We refer the reader to [3, sect. 10.B] for the combinatoric definition via permutations, to [32, sect. 4.2] for a functional characterization, and to [35, Ex. 21.1(c)] for a more practical definition. The equivalence of these four definitions is given, for example, by Strang [32, sect. 4.3]. Let $A \in \mathbb{C}^{n \times n}$, and let A_{ij} denote the $(n-1)$ -by- $(n-1)$ matrix obtained by removing the i th row and the j th column from A . For any $i, j \in \{1, 2, \dots, n\}$, the determinant of A is defined by

$$\det A = \sum_{k=1}^n (-1)^{i+k} \det A_{ik} = \sum_{k=1}^n (-1)^{j+k} \det A_{kj},$$

where the scalar $(-1)^{i+j} \det A_{ij}$ is called the *cofactor* of the element in position (i, j) of A . It is easy to see that $\det I = 1$, $\det A^T = \det A$, and $\det A^* = \overline{\det A}$. Moreover, it can be shown that $\det \alpha A = \alpha^n \det A$ for all $\alpha \in \mathbb{C}$, that a matrix $A \in \mathbb{C}^{n \times n}$ is singular if and only if $\det A = 0$, and that the determinant is multiplicative, that is, for any $A, B \in \mathbb{C}^{n \times n}$ one has that $\det(AB) = (\det A)(\det B)$, which readily gives that $\det A^{-1} = (\det A)^{-1}$,

Eigenvalues and eigenvectors. Let $A \in \mathbb{C}^{n \times n}$. If $\lambda \in \mathbb{C}$ and $x \in \mathbb{C}^n \setminus \{0\}$ verify the equation $Ax = \lambda x$, then λ is an *eigenvalue* of A , x is the corresponding *eigenvector*, and the pair (λ, x) is an *eigenpair* of A . The set of eigenvalues of A , denoted by $\sigma(A)$, is called the *spectrum* of A , and the magnitude of the largest eigenvalue, denoted by $\rho(A) = \{|\lambda| : \lambda \in \sigma(A)\}$ is its *spectral radius*. By rewriting the equation above as $(A - \lambda I)x = 0$, one can see that λ is an eigenvalue of A if and only if the matrix $A - \lambda I$ is singular or, equivalently, $\det(A - \lambda I) = 0$. In other words, the eigenvalues of A are the roots of the *characteristic polynomial* $\chi_A(z) = \det(A - zI) \in \mathbb{C}_n[z]$.

The Cayley–Hamilton theorem [20, Thm. 2.4.3.2] states that every complex square matrix satisfies its own characteristic polynomial, that is, $\chi_A(A) = 0$, but it need not be the polynomial of lowest degree with this property. The monic polynomial of least degree to be satisfied by a matrix A is called the *minimal polynomial* of A , and we

denote it by ϕ_A . It can be shown that χ_A and ϕ_A have the same roots (not necessarily with the same multiplicities), and that $\lambda \in \mathbb{C}$ is an eigenvalue of A if and only if it is a root of ϕ_A . The multiplicity of an eigenvalue λ as a root of the characteristic polynomial is its *algebraic multiplicity*. The *geometric multiplicity* of an eigenvalue λ is the number of linearly independent eigenvectors associated with λ , that is, the dimension of the subspace $\text{null}(A - \lambda I)$, usually called the *eigenspace* of λ .

Two matrices $A, B \in \mathbb{C}^{n \times n}$ are *similar* if there exists a nonsingular matrix $P \in \mathbb{C}^{n \times n}$ such that $A = PBP^{-1}$. In other words the linear operators A and B are the same up to a change of basis, and it can be shown that similar matrices have the same spectrum. A matrix A is *diagonalizable* if it is similar to a diagonal matrix D . In this case the decomposition $A = PDP^{-1}$ is called the *eigendecomposition* of A , and the diagonal elements of D are the eigenvalues of A .

Much more can be said about the eigenvalues of a matrix, and we refer the interested reader to [20, Chap. 1] for a detailed summary, and to [32, Chap. 5] for an elementary but rigorous discussion of eigenvalues, eigenvectors, and similarity transformations.

Vector and matrix norms. A function $\|\cdot\|: \mathbb{C}^{m \times n} \rightarrow \mathbb{R}$ is a *norm* on the vector space $\mathbb{C}^{m \times n}$ if for any $A, B \in \mathbb{C}^{m \times n}$ and $\alpha \in \mathbb{C}$, it satisfies:

1. $\|A\| \geq 0$, with $\|A\| = 0$ if and only if $A = 0$ (nonnegativity);
2. $\|\alpha A\| = |\alpha| \|A\|$ (homogeneity);
3. $\|A + B\| \leq \|A\| + \|B\|$ (triangular inequality).

We refer to a *vector norm* if the argument of the norm is a vector, and to a *matrix norm* if the argument is a matrix. Norms are uniformly continuous functions, and they are all equivalent up to a constant, in the sense that for any two norms $\|\cdot\|'$ and $\|\cdot\|''$ on the same vector space there exist two positive real constants $\alpha \leq \beta$ such that $\alpha \|A\|' \leq \|A\|'' \leq \beta \|A\|'$.

For the case $n = 1$, the class of p -norms is the most relevant. These norms are defined, for $p \in \mathbb{N}_0$, by

$$\|x\|_p = \left(\sum_{i=1}^m |x_i|^p \right)^{1/p}, \quad x \in \mathbb{C}^m,$$

and by passing to the limit in the definition above, one obtains

$$\|x\|_\infty = \max_{1 \leq i \leq m} |x_i|, \quad x \in \mathbb{C}^m.$$

Matrix norms can be defined in several ways. The *operator norm* (often called *subordinate* or *induced matrix norm*) on $\mathbb{C}^{m \times n}$ corresponding to the vector norm $\|\cdot\|$ is defined by

$$\|A\| = \max_{x \in \mathbb{C}^n \setminus \{0\}} \frac{\|Ax\|}{\|x\|} = \max_{x \in \mathbb{C}^n, \|x\|=1} \|Ax\|.$$

By definition one has that $\|I_n\| = 1$, and it can be shown that

$$\begin{aligned} \|A\|_1 &= \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|, \\ \|A\|_2 &= \rho(A^* A)^{1/2}, \\ \|A\|_\infty &= \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|. \end{aligned}$$

An example of matrix norm that is not induced by a vector norm is the Frobenius norm, defined for $A \in \mathbb{C}^{m \times n}$ by

$$\|A\|_F = \|\text{vec } A\|_2 = \left(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}.$$

It is easy to check that $\|A\|_F = \text{tr}(A^* A)^{1/2}$ and that $\|I_n\| = \sqrt{n}$, which shows that the Frobenius norm is not an operator norm. The norms $\|\cdot\|$, $\|\cdot\|'$, and $\|\cdot\|''$ are called *consistent* if $\|AB\| \leq \|A\|' \|B\|''$ for all $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{n \times p}$. A norm $\|\cdot\|$ on $\mathbb{C}^{m \times n}$ is *unitarily invariant* if $\|UAV\| = \|A\|$ for all unitary $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$. It is easy

to check that subordinate matrix norms, as well as the Frobenius norm, are consistent and unitarily invariant.

Norms can be used to characterize the spectral radius of square matrices, as it is easy to prove that, for any consistent norm $\|\cdot\|$ on $\mathbb{C}^{n \times n}$ and $A \in \mathbb{C}^{n \times n}$, one has that $\rho(A) \leq \|A\|$ and that $\lim_{k \rightarrow \infty} \sqrt[k]{\|A^k\|} = \rho(A)$. The *normwise condition number* of matrix inversion with respect to a norm $\|\cdot\|$ on $\mathbb{C}^{n \times n}$ is defined as $\kappa(A) = \|A\| \|A^{-1}\|$. We denote by $\kappa_F(A)$ the condition number with respect to the Frobenius norm, and by $\kappa_p(A)$ that with respect to the operator norm induced by the vector p -norm.

2.1.1 Matrix factorizations and decompositions

In this section we discuss several ways in which a matrix can be written as the product of two or three matrices with specific properties. We briefly recall the main theoretical results and, when of practical interest, some aspects related to their computation. For an in-depth discussion of the Jordan canonical form, we refer the reader to [20, Chap. 3]. An algorithmic perspective on these topics is given by Golub and Van Loan [15], whereas the accuracy and stability of each of the algorithms we describe here is discussed by Higham [18].

Jordan canonical form. Any matrix $A \in \mathbb{C}^{n \times n}$ with ν linearly independent eigenvectors is similar to a block diagonal matrix, $J = M^{-1}AM = \text{diag}(J_1, J_2, \dots, J_\nu)$ where $M \in \mathbb{C}^{n \times n}$ is nonsingular and each diagonal *Jordan block* is an upper triangular matrix of the form

$$J_i := \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix} \in \mathbb{C}^{m_i \times m_i}.$$

The matrix J is called the *Jordan canonical form* of A , and is unique up to the ordering of the blocks. The only eigenpair of the Jordan block J_i is (λ_i, e_1) ; the algebraic and

geometric multiplicity of λ_i as an eigenvalue of J_i are m_i and 1, respectively. An eigenvalue is *semisimple* if it appears only in Jordan blocks of size one, otherwise it is *defective*. A semisimple eigenvalue appearing in only one block is called *simple*. A matrix is defective if it has at least one defective eigenvalue or, equivalently, if the algebraic multiplicity of at least one of its eigenvalues exceeds its geometric multiplicity. If all eigenvalues are semisimple, then the Jordan canonical form of A is diagonal, and the matrix A is *diagonalizable*. Finally, A is *derogatory* if an eigenvalue appears in more than one Jordan blocks or, equivalently, if its minimal polynomial ϕ_A has degree strictly lower than its characteristic polynomial χ_A .

LU and Cholesky factorizations. We now discuss three ways of rewriting nonsingular matrices as the product of triangular and diagonal matrices. Since triangular systems are extremely easy to solve, these factorization are typically used in the solution of systems of linear equations.

For any nonsingular matrix $A \in \mathbb{C}^{n \times n}$, there exist a permutation matrix $P \in \mathbb{C}^{n \times n}$, a unit lower triangular matrix $L \in \mathbb{C}^{n \times n}$, and an upper triangular matrix $U \in \mathbb{C}^{n \times n}$ such that $PA = LU$. The matrices L and U are unique for a given P , but several strategies to choose the matrix P exist, and depending on the properties of A , the choice of P can influence the existence as well as the numerical stability of the LU decomposition.

For example, if all the principal leading minors of A are nonsingular, then the LU decomposition is guaranteed to exist with $P = I_n$, but this choice may perform poorly from a numerical point of view. The stability can be improved by relying on a *pivoting strategy*, that permutes the row of the matrix A while computing the factors L and U , and aims to reduce the growth of the backward error. A thorough analysis of the Gaussian elimination algorithm, which is typically used to compute the LU decomposition, is provided in [18, Chap. 9], along with an error analysis of complete, rook, and partial pivoting, the latter being the pivoting strategy most commonly used in practice. A discussion of the algorithmic and implementation aspects of Gaussian elimination can be found in [15, Chap. 3].

If the matrix $A \in \mathbb{C}^{n \times n}$ is symmetric positive definite, then there exists a unique lower triangular matrix $L \in \mathbb{C}^{n \times n}$ with positive diagonal entries such that $A = LL^*$. How to compute the Cholesky factor efficiently is discussed in [15, sect. 4.2], and the stability of the process is analyzed in detail in [18, Chap. 10].

QR factorization. If $A \in \mathbb{C}^{m \times n}$, there exist a unitary matrix $Q \in \mathbb{C}^{m \times m}$ and an upper triangular matrix $R \in \mathbb{C}^{m \times n}$ such that $A = QR$. This factorization is called the *QR factorization* of A , and if A has full column rank and $m < n$, by partitioning

$$Q = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}, \quad Q_1 \in \mathbb{C}^{m \times n}, \quad Q_2 \in \mathbb{C}^{m \times (m-n)},$$

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad R_1 \in \mathbb{C}^{n \times n}, \quad 0 \in \mathbb{C}^{(m-n) \times n},$$

we obtain the *thin QR factorization* $A = Q_1 R_1$.

This matrix factorization can be computed in many ways. The QR factorization via Householder reflections, Givens rotations, and via the Gram–Schmit procedure is discussed in details in [15, sect. 5.2]. In order to check whether the matrix A is (numerically) rank deficient, a pivoted version of these algorithms can be used to compute the *rank-revealing QR factorization*, that consists of a permutation matrix $P \in \mathbb{C}^{n \times n}$, a unitary matrix $Q \in \mathbb{C}^{m \times m}$, and an upper triangular matrix $R \in \mathbb{C}^{m \times n}$ such that $AP = QR$. It can be shown that the columns of A and those of R span the same space, thus the rank of A is the number of nonzero rows of R . The numerical behaviour of the algorithms to compute the QR factorization is discussed in [18, Chap. 19].

Schur decomposition. Any matrix $A \in \mathbb{C}^{n \times n}$ has a *Schur decomposition* $A = UTU^*$ where $U \in \mathbb{C}^{n \times n}$ is unitary and $T \in \mathbb{C}^{n \times n}$ is upper triangular. Since U is unitary, the matrix T is similar to A , thus the eigenvalues of A are the elements along the diagonal of T . It can be shown that T is diagonal if and only if A is normal. A real matrix $A \in \mathbb{R}^{n \times n}$ has also a *real Schur decomposition* QTQ^T where $Q \in \mathbb{R}^{n \times n}$ is orthogonal and T is upper quasi-triangular. Since Q is a similarity transformation, the matrices A and T have the same eigenvalues, and in particular any diagonal block of size 1

is a real eigenvalue of A , and the eigenvalues of a 2×2 block of T coincide with a complex conjugate pair of eigenvalues of A .

The Schur decomposition can be computed by means of the QR iteration, several variants of which are discussed in [15, Chap. 7], which covers most practical issues pertaining the efficient implementation of the algorithm, such as the use of the Hessenberg form of the matrix, implicit shifting, and deflation. More recently, a new interpretation of this algorithm as a core-chasing algorithm has been proposed [2, Chap. 3]. Note that in the latter reference the algorithm is called “Francis’s algorithm”, following the advice of Watkins [38]. The same author has at least two more classical references on this algorithm [36], [37].

SVD decomposition. If $A \in \mathbb{C}^{m \times n}$, there exists two unitary matrices $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ and a diagonal matrix $S \in \mathbb{R}^{m \times n}$, whose diagonal elements are non-negative and in nonincreasing order, such that $A = USV^*$. This is called the *SVD decomposition* of the matrix A . By partitioning, for $m > n$,

$$\begin{aligned} U &= \begin{bmatrix} U_1 & U_2 \end{bmatrix}, & U_1 &\in \mathbb{C}^{m \times n}, & U_2 &\in \mathbb{C}^{m \times (m-n)}, \\ S &= \begin{bmatrix} S_1 \\ 0 \end{bmatrix}, & S_1 &\in \mathbb{C}^{n \times n}, & 0 &\in \mathbb{C}^{(m-n) \times n}, \\ V &= V_1, \end{aligned}$$

and for $n < m$,

$$\begin{aligned} U &= U_1, \\ S &= \begin{bmatrix} S_1 & 0 \end{bmatrix}, & S_1 &\in \mathbb{C}^{m \times m}, & 0 &\in \mathbb{C}^{m \times (n-m)}, \\ V &= \begin{bmatrix} V_1 & V_2 \end{bmatrix}, & V_1 &\in \mathbb{C}^{m \times n}, & V_2 &\in \mathbb{C}^{(n-m) \times n}, \end{aligned}$$

we obtain the *thin SVD decomposition* $A = U_1 S_1 V_1^*$. The columns of U and V are called the left and right *singular vectors* of A , respectively, and the diagonal elements of S are its *singular values*. It is easy to show that the left and singular vectors are the

eigenvectors of the Hermitian positive semidefinite matrices AA^* and A^*A , respectively, and that the singular values are the positive square roots of the eigenvalues of AA^* (or A^*A , since $\sigma(A^*A) = \sigma(AA^*)$). Note that the largest singular value of A is its 2-norm.

The SVD decomposition provides much useful information about the linear transformation underlying the matrix A . For example, $\text{rank } A = \text{rank } S$, and since S is diagonal the rank of A is the number of its nonzero singular values. Moreover, if $r = \text{rank } A$, then the first r columns of U are a basis for the range of A , and the last $n - r$ columns of V are a basis for its null space. If the matrix A is not full rank, the thin SVD decomposition can be further reduced by dropping the left and right singular vectors corresponding to zero singular values.

This decomposition has countless numerical applications. It is the algorithm of choice for computing the numerical rank and 2-norm of a matrix, and provides the most accurate method for finding an orthonormal basis for the range and null space of a matrix. In applications, the SVD decomposition is the theoretical foundation for algorithms for computing the best low-rank approximations in the 2-norm and the Frobenius norm, and a subroutine in the algorithms for the solutions of archetypal numerical linear algebra problems such as least squares fitting and regularization.

Theoretical aspects of the SVD decomposition are covered in [20, sect. 2.6] and an extended list of properties can be found in [15, sect. 2.4]. Numerical algorithms for computing the SVD decomposition are discussed in [15, sect. 8.6].

2.1.2 Functions of matrices

Chapter 3 deals with one of the simplest examples of functions of matrices, namely *polynomials of matrices*. They are defined in section 3.1, where several algorithms for their efficient and accurate evaluation is presented. A more general definition of matrix function via the Jordan canonical form is provided in section 4.2, and the important concept of *primary matrix function* is defined in section 4.3.1, where primary solutions to matrix equations are also discussed. We show that these two concepts

are equivalent for functions defined as solutions to matrix equations, as it is the case, for example, for the p th root, the logarithm, and the Lambert W function of a matrix, implicitly defined by the equations $X^p = A$, $e^X = A$, and $Xe^X = A$, respectively.

Those few pages provide all the background that is needed to read this thesis, but do not cover at all many important aspects regarding functions of matrices. As further reading and to complement the information available there, we refer the interested reader to the monograph by Higham [19] and to the long paper by Evard and Uhlig [12].

2.2 FLOATING POINT ARITHMETIC

In this section we recall the floating point number system underlying the IEEE standard 754-1985 [21] and its revision 754-2008 [22]. The main references for the material covered here are [18, Chap. 2] and [28].

2.2.1 Floating point numbers

A family of floating point numbers $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ is a finite subset of the real line that is fully characterized by four parameters:

- the *radix* $b \in \mathbb{N} \setminus \{0, 1\}$;
- the *precision* $p \in \mathbb{N}$;
- the *extreme exponents* $e_{\min}, e_{\max} \in \mathbb{Z}$ such that $e_{\min} < e_{\max}$.

A real number $x := (s, m, e)$ belongs to the floating family $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ if it can be written as

$$x = (-1)^s \cdot m \cdot b^{e-p+1}, \tag{2.1}$$

where

- the *sign* s is 0 for $x \geq 0$ or 1 for $x < 0$;
- the *significand* m is a natural number not greater than $b^p - 1$;
- the *exponent* e is an integer such that $e_{\min} \leq e \leq e_{\max}$.

The function

$$f: \{0, 1\} \times \llbracket 0, b^p - 1 \rrbracket \times \llbracket e_{\min}, e_{\max} \rrbracket \rightarrow \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle \quad (2.2)$$

defined by (2.1) is not injective for a number (s, m, p) such that $m < b^{p-1}$ and $e > e_{\min}$, in which case $f(s, m, e) = f(s, m \cdot b, e - 1)$. The set of all possible representations for $x \in \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ that are admissible in $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ is denoted by

$$\zeta(x) := \{(s, m, e) \in \{0, 1\} \times \llbracket 0, b^p - 1 \rrbracket \times \llbracket e_{\min}, e_{\max} \rrbracket : x = f(s, m, e)\},$$

and is called the *cohort* of x . In order to ensure a unique representation for all numbers $x \in \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle \setminus \{0\}$, it is customary to *normalize* the system by assuming that if $x > b^{e_{\min}-p+1}$ then $b^{p-1} \leq m \leq b^p - 1$. In other words, a system is normalized if for all $x \in \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle \setminus \{0\}$ the triple with largest significand and smallest exponent is chosen as representative of the equivalence class $\zeta(x)$.

In such systems, the number $(s, m, e) \in \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle \setminus \{0\}$ is *normal* if $m > b^{p-1}$, and *subnormal* otherwise. The exponent of subnormal numbers is always e_{\min} , and in a normalized system any number $x = (s, m, e) \neq 0$ can be written in a unique way as

$$(-1)^s \cdot \left(d_0 + \frac{d_1}{b} + \cdots + \frac{d_{p-1}}{b^{p-1}} \right) \cdot b^e,$$

for some $d_0, d_1, \dots, d_{p-1} \in \llbracket 0, b - 1 \rrbracket$. Let $\tilde{m} := m \cdot b^{1-p}$ be the *normal significand* of $x = (s, m, e)$. If x is normal then $d_0 = b - 1$ and $1 \leq \tilde{m} < b$, whereas if x is subnormal then $d_0 < b - 1$ and $\tilde{m} < 1$. Finally, we pinpoint that 0 is the only number that does not have a unique representation in a normalized floating point system. In view of this peculiarity, it cannot be considered either normal or subnormal, and it belongs to a special class, together with two other special floating point data: infinity and NaN (Not a Number). Infinities are needed to express values whose magnitude

exceeds that of the largest positive and negative numbers that can be represented in $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$, that are $\pm(b - b^{1-p})b^{e_{\max}}$, whereas NaNs represent the result of invalid operations, such as taking the square root of a negative number, dividing 0 by 0, or multiplying an infinity by 0, and were introduced to ensure that the semantic of all floating point operations is well specified and the resulting floating point number system is closed.

In order to work with floating point numbers, it is necessary to have functions that can map the real numbers to a floating point family and vice versa. Since $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle \subset \mathbb{R}$, if $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ is normalized, then composing f in (2.2) with the identity function of \mathbb{R} restricted to the domain $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ gives an injective function that maps $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ to \mathbb{R} .

On the other hand, mapping real numbers to a certain family of floating point numbers proves harder, as in this case the function maps the infinite real line to a finite set. A function that performs this mapping is called a *rounding*, and the function $\text{fl}: \mathbb{R} \rightarrow \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ is a *rounding to nearest* if $\text{fl}(x)$ is an element in $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ nearest to $x \in \mathbb{R}$ in absolute value. A rounding to nearest is completely specified by this property and a rule to break ties; as tens of such rules exist, we refer the interested reader to [18, Chap. 2, Notes and References] for a general discussion, and to [28, sect. 2.2] and [22, sect. 4.3] for details about the rounding mode available in IEEE floating point arithmetic.

It can be shown [18, Thm. 2.2] that if $x \in [-(b - b^{1-p})b^{e_{\max}}, (b - b^{1-p})b^{e_{\max}}]$ then $\text{fl}(x) = x(1 + \delta)$ for some $\delta \in \mathbb{R}$ such that $|\delta| < u$, where

$$u := \frac{1}{2}b^{1-p}$$

is the *unit roundoff* of the floating point family $\mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$. An important consequence of this result is that u represents an accuracy threshold for the accuracy of floating point computations, thus it is customary to assume that for any $x, y \in \mathcal{F}\langle b, p, e_{\min}, e_{\max} \rangle$ and $\square = +, -, \times, \div$, one has that

$$\text{fl}(x \square y) = (x \square y)(1 + \delta), \quad |\delta| \leq u. \quad (2.3)$$

This assumption is referred to as the *standard model* for floating point arithmetic by Higham [18, Eq. (2.4)].

2.2.2 Fixed precision and arbitrary precision arithmetic

The four parameters b , p , e_{\min} , and e_{\max} specify only what numbers can be represented by a given normalized floating point number system, but in order to be of practical use, these four values need to be complemented by a floating point number format that specifies how these numbers can be stored as finite strings of digits. Most floating point formats of practical interest are *binary*, that is, work in radix 2 and represent floating point numbers by means of strings of binary digits, or *bits*.

By far the most common floating point number formats are the IEEE single and double precision binary floating point arithmetic specified by the ANSI/IEEE standard 754-1985 [21]. We note that the 2008 revision of the standard 754-2008 includes some radix-10 formats, but we do not consider them here as they currently are of limited practical interest. These two standards specify how to store efficiently in a portable way the numbers in the floating point families $\mathcal{F}\langle 2, 24, -126, 127 \rangle$ (*binary*₃₂, previously known as *single*), $\mathcal{F}\langle 2, 53, -1022, 1023 \rangle$ (*binary*₆₄, previously known as *double*), and $\mathcal{F}\langle 2, 113, -16382, 16383 \rangle$ (*binary*₁₂₈, previously known as *double extended*¹). These formats are not flexible, in that the user is not allowed to adjust the exponent range or the precision at which the computation is performed. This lack of flexibility, however, is counterbalanced by the performance offered by highly optimized hardware implementations of the logic circuits that operate on these numbers, and the possibility of writing efficient algorithms that are fine-tuned for a specific precision.

In many cases, on the other hand, one is willing to trade off some computational efficiency for the ability to compute with any number of digits. Floating point number formats that satisfy this need are said to be *arbitrary precision*, *multiprecision*, or *multiple precision*, and they typically allow the user not only to perform arbitrarily accurate computations, but also to recur to different levels of precision for different

¹ The term *quadruple* is often used, since *binary*₁₂₈ numbers can be stored using exactly four times as many bits as single precision ones.

portions of a same algorithm. The latter capability can be exploited to increase the performance of fixed precision as well as multiprecision code, by running in low precision subroutines dealing with quantities for which only an estimate is needed, and resorting to higher precision only in those key points where catastrophic cancellation might strike the computation. Arbitrary precision is usually not implemented in hardware, but made available through software libraries, which leads to lower efficiency if compared with fixed precision floating point frameworks.

The first software package to provide multiple precision capabilities was Brent's MP package [10], [11], a 618 line Fortran 66 library of arbitrary precision subroutines for performing a broad variety of computations with floating point numbers. In 1993 Bailey presented a more complete multiprecision system [4], which combined MPFUN, a package of multiprecision Fortran 77 subroutines, and TRANSMP, a program capable of translating source code equipped with special directives into Fortran 77 programs relying on MPFUN. A couple of years later Bailey released a Fortran 90 version of the package [5], and more recently a thread-safe version has been developed [6]. A C++ rewriting of MPFUN, called ARPREC [7], is also available.

A wide range of software supporting multiprecision floating point arithmetic is available, and a large number of programming languages supports arbitrary precision, either natively or via dedicated libraries. One of the most complete and well-maintained software packages for working with arbitrary precision arithmetic is the GNU MPFR library for multiple-precision floating point computation with correct rounding [13], based on GMP, the GNU Multiple Precision Arithmetic Library [16]. This library is written in C, but interfaces for most major programming languages are provided,² and many programming languages for scientific computing rely on it in order to provide multiprecision capabilities.

The programming language Julia [8] supports multiprecision floating point numbers by means of the built-in data type `BigFloat`, a wrapper to GNU MPFR, and the third-party data type `ArbFloat`, that relies on the C library `Arb` [23], also based on GNU MPFR. These data types allow for multiprecision computations with scalars, and provide a few basic linear algebra kernels. The `ArbFloat` package, in particular,

² A complete list can be found at <https://www.mpfr.org/#interfaces>.

has functions for computing the matrix square root and exponential in multiprecision. The programming language Python does not natively support multiprecision, but arbitrary precision data types are provided by the `mpmath` [24] and `SymPy` [27], [34] libraries, both of which depend on GNU MPFR in order to work with arbitrary precision floating point numbers. The `mpmath` library provides functions for evaluating in arbitrary precision a wide range of matrix functions, including real powers, exponential, logarithm, sine, and cosine. Finally, the `Boost.Multiprecision` library [9] for C++ relies on GNU MPFR to provide arbitrary precision floating point numbers in C++, but does not offer any linear algebra subroutines.

Most computer algebra systems support arbitrary precision computation. The open source systems Sage [31] and PARI/GP [30] support arbitrary precision floating point arithmetic, but do not implement any algorithms for the evaluation of matrix functions. The proprietary systems Maple [25] and Mathematica [26] offer functions that can evaluate in arbitrary precision real matrix powers, the matrix logarithm, the matrix exponential, and a function that computes $f(A)$ given a scalar function f and a square matrix A . The algorithms underlying the functions described above are not publicly available, to our knowledge, and in some cases they may involve symbolic arithmetic.

In the following chapters, we use the programming language MATLAB for all our numerical experiments. MATLAB does not support arbitrary precision floating point arithmetic natively, but the arbitrary precision floating point data types `mp` and `vpa` are provided by the Multiprecision Computing Toolbox [29] and the Symbolic Math Toolbox [33], respectively. Both toolboxes implement algorithms for the matrix square root, the exponential, the logarithm, and general matrix functions, and the Multiprecision Computing Toolbox also includes functions for computing the hyperbolic and trigonometric sine and cosine of a matrix in arbitrary precision.

The specifications of the floating point format underlying these two data types are not publicly available, but a direct experimentation can provide a few hints. For both toolboxes, the user can specify the required accuracy in terms of decimal digits, and the commands `digits(d)` and `mp.Digits(d)` set the working precision of `mp`

and `vpa` numbers, respectively, to d decimal digits. The experiments below are run in MATLAB 2018b, using version 4.5.3.12859 of the Advanpix Multiprecision Computing Toolbox and version 8.2 of the Symbolic Math Toolbox. First, in order to determine the radix-2 machine epsilon 2^{-p+1} of the two floating point formats, we use the function

```
function machine_epsilon = findeps(x)
    machine_epsilon = x;
    while x + (machine_epsilon/2) > x
        machine_epsilon = machine_epsilon / 2;
    end
end
```

which for IEEE single and double precision gives

```
>> findeps(single(1))
ans =
    single
    1.1920929e-07
>> findeps(double(1))
ans =
    2.220446049250313e-16
```

As expected, these two numbers are the single and double precision roundings of 2^{-23} and 2^{-52} , respectively. By setting the precision of the two toolboxes to $d = 34$, which should roughly be equivalent to IEEE quadruple precision, we obtain:

```
>> digits(34); findeps(vpa(1))
ans =
    2.869859254937225361251798186577748e-42
>> mp.Digits(34); findeps(mp(1))
ans =
    1.925929944387235853055977942584927e-34
```


where the former, corresponding to 2^{-112} is indeed the machine epsilon of IEEE quadruple precision.

This simple experiment, along with more extensive testing not reported here, seems to suggest that for a given d , the machine epsilon is $2^{\lceil \log_2(10^{-d}) \rceil}$ for the Multiprecision Computing Toolbox and $2^{\lceil \log_2(10^{-(d+8)}) \rceil + 1}$ for the Symbolic Math Toolbox. Our interpretation is that the latter performs the computation at much higher precision, without truncating the result, and then prints the number rounded to the first d significant decimal digits. This behaviour makes it hard to compare the results computed by the two toolboxes in a fair way.

An additional difficulty is given by the fact that it is not possible to compare mp and vpa numbers directly, since the former are cast into double before being converted into the latter, and trying to cast a vpa number into an mp object raises an error, as the following listing demonstrates.

```
>> d = 34; mp.Digits(d); digits(d - 8);
>> x = mp('5647653643/64736284736')
x =
    0.08724092934946151680248318907789599
>> vpa(x)
ans =
    0.087240929349461510122765162
>> vpa(x) - vpa(double(x))
ans =
    0.0
>> mp(vpa('5647653643/64736284736'))
Error using mp (line 1256)
Unsupported argument type
```

In view of these two complications, we chose to work with only one of the two toolboxes, the Multiprecision Computing Toolbox, not only because of its more consistent behaviour, but also because of its much faster execution time.

2.2.3 Error analysis

Errors striking numerical computations can have different sources. The input data may be inexact: noise can corrupt the measurement of physical quantities, rounding errors may arise when exact information is converted to finite precision so to be stored in a computer, and degradation can occur as a results of earlier computation. These issues are the realm of *uncertainty quantification*, but we will not explore them further in this thesis.

One of the main concerns of *numerical analysis*, on the other hand, are quantization errors, caused by the discretization of continuous quantities, and truncation errors, which arise when an infinite sum is truncated and replaced by a finite approximation that trades off accuracy for computational efficiency. For instance, if a function in a neighbourhood of a point is approximated by means of the first few terms of its Taylor expansion, the terms that are left out represent an archetypal example of truncation error. We consider the problem of bounding the truncation error of Padé approximations in Chapters 5 and 6.

Finally, the accumulation of rounding errors in the standard floating point model (2.3) is an inevitable consequence of the use of finite precision arithmetic, and is the subject of investigation of *error analysis*. The effect of rounding errors in linear algebra computations is considered in detail by Higham [17].

There are two ways of interpreting truncation and rounding errors. The aim of *forward error analysis* is to measure how far the solution returned by an algorithm is from the exact value the algorithm was meant to compute. Even if in practice the exact solution is typically not known, the forward error can be estimated by means of a reference solution computed in higher precision. *Backward error analysis* takes a different approach: the computed solution is seen as the exact solution to a perturbation of the original problem. As many such perturbations may exist, in general, one is usually interested in finding the smallest with respect to some metric. Depending on the problem at hand, backward errors can be hard to estimate experimentally, but are

an invaluable tool in understanding the stability of numerical algorithms, and have practical applications in the design of many algorithms for functions of matrices.

BIBLIOGRAPHY

- [1] M. ARTIN, *Algebra*, Prentice-Hall, Upper Saddle River, NJ, USA, 1991.
- [2] J. L. AURENTZ, T. MACH, L. ROBOL, R. VANDEBRIL, AND D. S. WATKINS, *Core-Chasing Algorithms for the Eigenvalue Problem*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2018.
- [3] S. J. AXLER, *Linear Algebra Done Right*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1997.
- [4] D. H. BAILEY, *Algorithm 719: multiprecision translation and execution of Fortran programs*, ACM Trans. Math. Software, 19 (1993), p. 288–319.
- [5] ———, *A Fortran 90-based multiprecision system*, ACM Trans. Math. Software, 21 (1995), p. 379–387.
- [6] D. H. BAILEY, *A thread-safe arbitrary precision computation package (full documentation)*, 2018.
- [7] D. H. BAILEY, H. YOZO, X. S. LI, AND B. THOMPSON, *ARPREC: An arbitrary precision computation package*, tech. report, Lawrence Berkeley National Laboratory, 2002.
- [8] J. BEZANSON, A. EDELMAN, S. KARPINSKI, AND V. B. SHAH, *Julia: A fresh approach to numerical computing*, SIAM Rev., 59 (2017), pp. 65–98.
- [9] BOOST C++ libraries. <http://www.boost.org>.
- [10] R. P. BRENT, *Algorithm 524: MP, a Fortran multiple-precision arithmetic package [A1]*, ACM Trans. Math. Software, 4 (1978), p. 71–81.
- [11] ———, *A Fortran multiple-precision arithmetic package*, ACM Trans. Math. Software, 4 (1978), p. 57–70.

- [12] J.-C. EVARD AND F. UHLIG, *On the matrix equation $f(X) = A$* , Linear Algebra Appl., 162-164 (1992), pp. 447–519.
- [13] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, AND P. ZIMMERMANN, *MPFR: A multiple-precision binary floating-point library with correct rounding*, ACM Trans. Math. Software, 33 (2007), pp. 13:1–13:15.
- [14] S. H. FRIEDBERG, A. J. INSEL, AND L. E. SPENCE, *Linear Algebra*, Featured Titles for Linear Algebra (Advanced) Series, Pearson Education, 4th ed., 2003.
- [15] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4th ed., 2013.
- [16] T. GRANLUND AND G. D. TEAM, *GNU MP 6.0 Multiple Precision Arithmetic Library*, Samurai Media Limited, United Kingdom, 2015.
- [17] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.
- [18] ———, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002.
- [19] ———, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [20] R. A. HORN AND C. R. JOHNSON, *Matrix Analysis*, Cambridge University Press, 1985.
- [21] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.
- [22] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008 (revision of IEEE Std 754-1985)*, Institute of Electrical and Electronics Engineers, New York, 2008.
- [23] F. JOHANSSON, *Arb: Efficient arbitrary-precision midpoint-radius interval arithmetic*, IEEE Trans. Comput., 66 (2017), p. 1281–1292.

- [24] F. JOHANSSON ET AL., *Mpmath: A Python library for arbitrary-precision floating-point arithmetic*. <http://mpmath.org>.
- [25] *Maple*. Waterloo Maple Inc., Waterloo, Ontario, Canada. <http://www.maplesoft.com>.
- [26] *Mathematica*. Wolfram Research, Inc., Champaign, IL, USA. <http://www.wolfram.com>.
- [27] A. MEURER, C. P. SMITH, M. PAPROCKI, O. ČERTIK, S. B. KIRPICHEV, M. ROCKLIN, A. KUMAR, S. IVANOV, J. K. MOORE, S. SINGH, T. RATHNAYAKE, S. VIG, B. E. GRANGER, R. P. MULLER, F. BONAZZI, H. GUPTA, S. VATS, F. JOHANSSON, F. PEDREGOSA, M. J. CURRY, A. R. TERREL, Š. ROUČKA, A. SABOO, I. FERNANDO, S. KULAL, R. CIMRMAN, AND A. SCOPATZ, *SymPy: Symbolic computing in Python*, PeerJ Computer Science, 3 (2017), p. e103.
- [28] J.-M. MULLER, N. BRISEBARRE, F. DE DINECHIN, C.-P. JEANNEROD, V. LEFÈVRE, G. MELQUIOND, N. REVOL, D. STEHLÉ, AND S. TORRES, *Handbook of Floating-Point Arithmetic*, Birkhäuser, Boston, MA, USA, 2010.
- [29] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. <http://www.advanpix.com>.
- [30] *PARI/GP*. <http://pari.math.u-bordeaux.fr>.
- [31] THE SAGE DEVELOPERS, *Sage Mathematics Software*. <http://www.sagemath.org>.
- [32] G. STRANG, *Linear Algebra and its Applications*, Harcourt Brace Jovanovich, San Diego, CA, 3rd ed., 2013.
- [33] *Symbolic Math Toolbox*. The MathWorks, Inc., Natick, MA, USA. <http://www.mathworks.co.uk/products/symbolic/>.
- [34] SYMPY DEVELOPMENT TEAM, *Sympy: Python library for symbolic mathematics*. <http://www.sympy.org>.
- [35] L. N. TREFETHEN AND D. BAU III, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

- [36] D. S. WATKINS, *Understanding the QR algorithm*, SIAM Rev., 24 (1982), p. 427–440.
- [37] D. S. WATKINS, *The QR algorithm revisited*, SIAM Rev., 50 (2008), p. 133–145.
- [38] D. S. WATKINS, *Francis's algorithm*, Amer. Math. Monthly, 118 (2011), p. 387.

3

OPTIMALITY OF THE PATERSON-STOCKMEYER METHOD FOR EVALUATING POLYNOMIALS AND RATIONAL FUNCTIONS OF MATRICES

Abstract. Many state-of-the-art algorithms reduce the computation of transcendental matrix functions to the evaluation of polynomial or rational approximants at a matrix argument. This task can be accomplished efficiently by resorting to the Paterson–Stockmeyer method, an evaluation scheme originally developed for matrix polynomials that extends quite naturally to rational functions. An important feature of this technique is that the number of matrix multiplications required to evaluate an approximant of order n grows slower than n itself, with the result that different approximants yield the same asymptotic computational cost. We analyze the number of matrix multiplications required by the Paterson–Stockmeyer method and by two widely used generalizations, one for evaluating diagonal Padé approximants of general functions and one specifically tailored to those of the exponential. In all the three cases, we identify the approximants of maximum order for any given computational cost.

Keywords: Paterson–Stockmeyer method, polynomial evaluation, polynomial of a matrix, matrix rational function, matrix function.

2010 MSC: 15A16, 13M10, 65F60.

3.1 INTRODUCTION

Several numerical methods for evaluating matrix functions, including the state-of-the-art algorithms for computing the exponential [1], [13], [14, Chap. 10], the loga-

rithm [2], [7], trigonometric [3] and hyperbolic functions, and their inverses [5], rely on rational approximation. The special case of polynomial approximants is of particular interest, as it usually yields simpler formulae and often leads to elementary proofs of theoretical results. In the literature, algorithms based on polynomial approximation have been proposed for computing the matrix exponential [6], [8], [9], [20], [21], the matrix logarithm [10], and trigonometric matrix functions [4], [19].

In order to compute $f(A)$, where $A \in \mathbb{C}^{n \times n}$ and f is a primary matrix functions in the sense of [14, Def. 1.2], these algorithms typically perform three main steps. First, a series of transformations is applied to A , in order to obtain a matrix B for which a suitable polynomial or rational approximant to f is guaranteed to deliver a prescribed level of accuracy. This approximant is then evaluated at the matrix B , and an approximation of $f(A)$ is obtained by exploiting properties of f in order to reverse the transformations initially applied to A .

Let us consider the polynomial

$$p(A) = \sum_{i=0}^k c_i A^i, \quad (3.1)$$

where $k \in \mathbb{N}$ and $c_0, c_1, \dots, c_k \in \mathbb{C}$. As a polynomial is nothing but a linear combination of powers of its argument, one can evaluate $p(A)$ by explicitly computing the first k powers of A , scaling them by the corresponding coefficients of p , and summing them up. If all the powers A^2, A^3, \dots, A^k are computed, this algorithm requires $k - 1$ matrix multiplications, k matrix scalings, k matrix sums, and one diagonal update of the form $A \leftarrow A + \alpha I$, for $\alpha \in \mathbb{C}$, which can be performed efficiently without explicitly forming the diagonal matrix αI . This technique requires at least $2n^2$ additional elements of storage, as it is necessary to keep track of the intermediate powers of A and of the accumulated partial sum.

A second evaluation scheme for (3.1) is the matrix version of Horner's method. This is the algorithm of choice for scalar polynomials, as it reduces the number of

multiplications to be performed without affecting that of scalar additions. In order to employ this scheme, we define the recursion

$$\begin{aligned} P_{k-1} &= c_k A + c_{k-1} I, \\ P_i &= P_{i+1} A + c_i I, \quad i = k-2, k-3, \dots, 0, \end{aligned} \tag{3.2}$$

and evaluate $p(A) = P_0$ by computing P_i for i from $k-1$ down to 0. For dense polynomials, this method requires $k-1$ matrix multiplications, but only one matrix scaling and k diagonal updates, and can be implemented in a memory efficient way that requires only a half of the additional storage needed by the algorithm that evaluates $p(A)$ by explicitly computing the powers of A .

In order to reduce the number of matrix multiplications needed to form $p(A)$, Paterson and Stockmeyer [17] proposed a less straightforward approach, which for $k \geq 4$ yields an operation count lower than that of the two techniques discussed thus far. By collecting powers of A in a suitable fashion, for $s \in \mathbb{N}_0 := \mathbb{N} \setminus \{0\}$ we obtain

$$p(A) = \sum_{i=0}^v (A^s)^i B_i^{[p]}(A), \quad v = \left\lfloor \frac{k}{s} \right\rfloor, \tag{3.3}$$

where

$$B_i^{[p]}(A) = \begin{cases} \sum_{j=0}^{s-1} c_{si+j} A^j, & i = 0, 1, \dots, v-1, \\ \sum_{j=0}^{|k|_s} c_{si+j} A^j, & i = v, \end{cases}$$

Here $|a|_b$ denotes, for two integers a and b , the remainder on integer division of a by b . In other words, if $|a|_b = \delta \in \mathbb{N}$, then $a = \gamma b + \delta$ for some $\gamma \in \mathbb{N}$. If $\delta = 0$, that is, if a is an integer multiple of b , we write $b \mid a$.

The scheme (3.3) requires $k - v + 1$ matrix scalings and additions, and $v + 1$ diagonal updates; computing A^2, A^3, \dots, A^s requires $s - 1$ matrix multiplications, and, at the price of storing these $s - 1$ additional matrices, no extra multiplication is needed

to compute $B_i^{[p]}(A)$, for $i = 0, \dots, \nu$. By evaluating (3.3) à la Horner, we obtain the recursion

$$\begin{aligned} \tilde{P}_{\nu-1} &= \begin{cases} c_k A^s + B_{\nu-1}^{[p]}(A), & s \mid k, \\ A^s B_{\nu-1}^{[p]}(A) + B_{\nu-1}^{[p]}(A), & s \nmid k, \end{cases} \\ \tilde{P}_i &= A^s \tilde{P}_{i+1} + B_i^{[p]}(A), \end{aligned} \quad i = \nu-2, \nu-3, \dots, 0, \quad (3.4)$$

and computing $p(A) = \tilde{P}_0$ requires $\nu - 1$ additional matrix multiplications if k is a multiple of s , and ν if it is not. Therefore, evaluating (3.1) by means of (3.3) requires

$$C_s^p(k) := s - 1 + \left\lfloor \frac{k}{s} \right\rfloor - [s \mid k] \quad (3.5)$$

matrix multiplications, where $[\cdot]$ denotes the Iverson bracket, defined, for a proposition \mathcal{P} , by

$$[\mathcal{P}] = \begin{cases} 1, & \text{if } \mathcal{P} \text{ is true,} \\ 0, & \text{if } \mathcal{P} \text{ is false.} \end{cases}$$

Taking the derivative of (3.5) with respect to s shows that the continuous relaxation of $C_s^p(k)$ is minimized by taking

$$s^* := \sqrt{k}. \quad (3.6)$$

As s must be an integer, we can choose either $s = \lfloor \sqrt{k} \rfloor$ or $s = \lceil \sqrt{k} \rceil$. These two choices, together with the evaluation scheme (3.3), give two variants of the Paterson–Stockmeyer method. (Note that this evaluation scheme is not defined for $k = 0$.) Hargreaves [12, Thm. 1.7.4] proved that, in fact, these two algorithms have the same cost for any $k \in \mathbb{N}$. In the next section, we provide a new proof of this result, in which we establish the notation and present techniques we will rely on later on.

It is important to pinpoint that this approach trades off memory for computational efficiency, since $s + 1$ additional matrices need to be stored, for a space complexity of $O(\sqrt{k}n^2)$. Van Loan [22] showed that, by computing $p(A)$ one column at a time, it

is possible to reduce the storage requirement of the algorithm to $3n^2$ additional elements, at the price of $(\alpha \log_2 s - 1)n^3$ additional flops, where α is a small constant that depends only on s . How to implement the original Paterson–Stockmeyer algorithm and this variant in a memory and communication efficient way has been recently discussed by Hoffman, Schwartz, and Toledo [15].

We note that the Paterson–Stockmeyer method is not the fastest known algorithm for evaluating polynomials of matrices: Paterson and Stockmeyer [17] discuss a technique that requires fewer matrix multiplications than the algorithm above, and an alternative approach for reducing the number of matrix multiplications to evaluate polynomials of matrices has recently been proposed by Sastre [18]. These algorithms evaluate several appropriately chosen polynomials of lower degree, whose coefficients are obtained from those of the original polynomial by means of various techniques. This preprocessing stage may introduce numerical instabilities, thus the new coefficients must be carefully chosen on a case-by-case basis, as done for example in [20] for the truncated Taylor approximants to the exponential of order 8, 15, 24, and 30.

Polynomials of the form (3.1) often arise when computing matrix functions by relying on Padé approximation. A rational function $r_{km} = p_{km}/q_{km}$, for $k, m \in \mathbb{N}$, is the $[k/m]$ Padé approximant to f at o if p_{km} and q_{km} are polynomials of degree k and m , respectively, $q_{km}(0) = 1$, and the first $k + m$ terms in the series expansion of $f(x) - r_{km}(x)$ at o are zero. In particular, we focus on truncated Taylor series, for which $m = 0$, and diagonal Padé approximants, for which $m = k$, since these are the two families of Padé approximants most commonly encountered in the literature. Subdiagonal Padé approximants are also considered [11], [16], but the partial fraction form is usually preferred for their evaluation.

The scheme (3.3) readily generalizes to the evaluation of rational matrix functions: after computing the first s powers of A , for some $s \in \mathbb{N}_0$, one can evaluate numerator and denominator separately, by means of (3.3), and then solve a multiple right-hand side linear system. An approximately optimal value for s can be determined by minimizing the continuous relaxation of the corresponding cost function.

Since the cost of matrix multiplications is asymptotically higher than that of matrix scalings and matrix sums, we follow the customary practice of measuring the efficiency of algorithms for evaluating polynomials of matrices by counting the numbers of matrix multiplications that need to be performed [14, Chap. 4]. The goal of this work is twofold. On the one hand, we study the optimality of the Paterson–Stockmeyer method amongst all methods of the form (3.3); on the other, we give several results that can aid in developing numerical algorithms for the computation of matrix functions. Now we summarize our contribution while outlining the structure of the following sections.

It has been observed [14, p. 74] that the Paterson–Stockmeyer method minimizes the number of matrix multiplications required to evaluate polynomials of degree between 2 and 16 by means of the scheme (3.3). In section 3.2.1 we show that this is in fact the case for polynomials of any degree.

When matrix functions are approximated by means of polynomials, it is customary not to consider all possible approximants, but only those that maximize the approximation degree for a given number of matrix multiplications. For example, since $C_s^p(11) \geq 5$ and $C_s^p(12) \geq 5$ for any $s \in \mathbb{N}_0$, there is little point in considering an approximant of degree 11 when that of degree 12 is likely to deliver a more accurate approximation at the same cost. The following definition allows us to make this notion precise and extend it to the case of rational approximants.

Definition 3.1 (Optimal orders of an evaluation scheme). Let $C(k)$, for $k \in \mathbb{N}$, be the number of matrix multiplications required by a scheme \mathcal{S} to evaluate an approximant of order k . Then $k' \in \mathbb{N}$ is an optimal order (or degree, if the approximant is a polynomial) for \mathcal{S} if there exists $\zeta \in \mathbb{N}$ such that

$$k' = \max\{k \in \mathbb{N} : C(k) = \zeta\}.$$

When designing algorithms for fixed precision arithmetic, one typically knows the order of the highest approximant that may be needed to achieve the required accuracy, k_{\max} say, and only the optimal orders smaller than k_{\max} are needed. These can be found by inspecting the values of $C(k)$ for $k \leq k_{\max}$, as was done in [14,

Table 4.1] and [6, Table 1] for polynomial approximants and in [14, Table 10.3] for the diagonal Padé approximants to the exponential. In arbitrary precision floating-point environments, however, depending on the working precision and the desired accuracy, an approximant of arbitrarily high order may be needed, and alternative techniques to efficiently find all optimal degrees become necessary.

In section 3.2.2, we derive a formula for the sequences of optimal degrees for the Paterson–Stockmeyer method for polynomial evaluation. In section 3.3, we obtain closed formulae for the optimal orders of the Paterson–Stockmeyer-like scheme for evaluating rational functions whose numerator and denominator have same degree, and in section 3.4 we consider the special case of the diagonal Padé approximants to the exponential.

Finally, in section 3.5 we summarize our findings and outline possible directions for future work.

3.2 EVALUATION OF MATRIX POLYNOMIALS

Figure 3.1 shows the value of the cost function (3.5) for the two canonical variants of the Paterson–Stockmeyer method, which differ only in the direction \sqrt{k} is rounded in order to obtain the parameter s in (3.3). It is well known that both choices yield the same computational cost for the evaluation of a polynomial of any degree, and in section 3.2.1 we show that this is the minimum value for $C_s^p(k)$ among all choices of $s \in \mathbb{N}^+$. The values marked with a red circle are discussed in section 3.2.2.

3.2.1 Optimality of the Paterson–Stockmeyer method

Most of the results that follow stem from a couple of simple observations. If $s = \lfloor \sqrt{k} \rfloor$, then by definition of the floor operator, we have that

$$s \leq \frac{k}{s} < \frac{(s+1)^2}{s} = s + 2 + \frac{1}{s}, \quad (3.7)$$

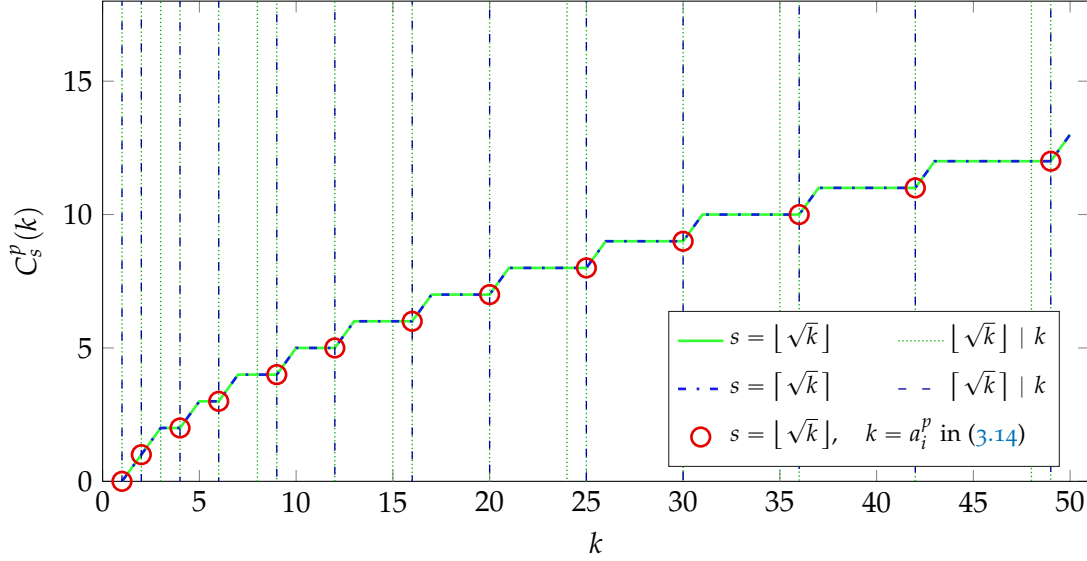


Figure 3.1: Number of matrix multiplications required to evaluate a polynomial of degree k , for k between 1 and 50, by means of the scheme (3.3) with $s = \lfloor \sqrt{k} \rfloor$ and $s = \lceil \sqrt{k} \rceil$. The dotted and dashed lines mark the values of k that are integer multiples of $\lfloor \sqrt{k} \rfloor$ and $\lceil \sqrt{k} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate polynomials of optimal degree (in the sense of Definition 3.1) for the Paterson–Stockmeyer method.

where the first inequality holds strictly if \sqrt{k} is not an integer. It follows that $\lfloor \frac{k}{s} \rfloor = s + t$, where t can only be 0, 1, or 2, and in fact it is convenient to split (3.7) into the three subcases

$$s + t \leq \frac{k}{s} < s + t + 1, \quad t = 0, 1, 2. \quad (3.8)$$

Combining (3.7) and (3.8) for $t = 2$ with the fact that k is an integer, reveals that $\lfloor \frac{k}{s} \rfloor = s + 2$ only if $s \mid k$, that is, only if $k = s(s + 2)$.

Theorem 3.1 (Hargreaves, [12, Thm. 1.7.4]). *Let $A \in \mathbb{C}^{n \times n}$ and let p be a polynomial of degree $k \in \mathbb{N}_0$. The two methods obtained by setting s in (3.3) to $s_f = \lfloor \sqrt{k} \rfloor$ and $s_c = \lceil \sqrt{k} \rceil$ require the same number of matrix multiplications to evaluate $p(A)$.*

Proof. We need to prove that $C_{s_f}^p(k) = C_{s_c}^p(k)$ for any $k \in \mathbb{N}_0$. If k is a perfect square, then $s_f = s_c$ and the result follows immediately. Otherwise, one has that $s := s_f = s_c - 1$, and thus that

$$\Delta(k) := C_{s_f}^p(k) - C_{s_c}^p(k) = \left\lfloor \frac{k}{s} \right\rfloor - [s \mid k] - 1 - \left\lfloor \frac{k}{s+1} \right\rfloor + [s+1 \mid k]. \quad (3.9)$$

If $s \mid k$ and $k \neq s^2$, then (3.7) implies that $\nu = \left\lfloor \frac{k}{s} \right\rfloor = \frac{k}{s}$ is either $s+1$ or $s+2$. If $\nu = s+1$, then $k = s(s+1)$ and $s+1 \mid k$, and substituting into (3.9) gives $\Delta(k) = 0$. If $\nu = s+2$, then

$$\frac{k}{s+1} = \frac{s(s+2)}{s+1} = s+1 - \frac{1}{s+1},$$

hence $\left\lfloor \frac{k}{s+1} \right\rfloor = s$ and $s+1 \nmid k$, and once again substituting into (3.9) shows that $\Delta(k) = 0$. When $s+1 \mid k$, multiplying (3.7) by $\frac{s}{s+1}$ gives

$$s-1 + \frac{1}{s+1} < \frac{k}{s+1} < s+1,$$

which leads once again to the case $k = s(s+1)$.

Finally, if $s \nmid k$ and $s+1 \nmid k$, then $\left\lfloor \frac{k}{s} \right\rfloor = s+t$, where t is either 0 or 1, and multiplying (3.8) by $\frac{s}{s+1}$ gives

$$s+t-1 - \frac{t-1}{s+1} \leq \frac{k}{s+1} < s+t - \frac{t}{s+1},$$

which implies that

$$\left\lfloor \frac{k}{s+1} \right\rfloor = s+t-1 = \left\lfloor \frac{k}{s} \right\rfloor - 1. \quad (3.10)$$

Substituting (3.10) into (3.9) concludes the proof. \square

In view of the result in Theorem 3.1, we can drop the subscript and adopt the notation $C^p(k)$ to indicate the number of matrix multiplications required by the Paterson–Stockmeyer method.

Next, we show that the Paterson–Stockmeyer method is the cheapest algorithm one can obtain from the evaluation scheme (3.3). Note that this result is not an obvious consequence of the optimality of s^* in (3.6), since the continuous relaxation of (3.5) does not take into account the discontinuities induced by the floor operator in $\left\lfloor \frac{k}{s} \right\rfloor$ and the non-continuous term $[s \mid k]$.

Proposition 3.2. *Let $A \in \mathbb{C}^{n \times n}$ and let p be a polynomial of degree $k \in \mathbb{N}_0$. The Paterson–Stockmeyer method minimizes the number of matrix multiplications required to evaluate $p(A)$ by means of the evaluation scheme (3.3).*

Proof. Let $s = \lfloor \sqrt{k} \rfloor$. In view of Theorem 3.1, it suffices to show that $C_{s+\ell}^p(k) \leq C_s^p(k)$, for all $\ell \in \mathbb{Z}$ such that $\ell > -s$. The proof is by exhaustion since, by (3.7), ν can take only the three values $s, s+1$, and $s+2$. For $t = 0, 1$, or 2 , we have that

$$C_s^p(k) = 2s + t - 1 - [s \mid k], \quad (3.11)$$

and since

$$\frac{k}{s+\ell} \geq \frac{s(s+t)}{s+\ell} = s - \ell + t + \eta_t^\ell, \quad \eta_t^\ell := \frac{\ell(\ell-t)}{s+\ell}, \quad (3.12)$$

we can conclude that

$$C_{s+\ell}^p(k) = s + \ell - 1 + \left\lfloor \frac{k}{s+\ell} \right\rfloor - [s+\ell \mid k] \geq 2s + t - 1 + \lfloor \eta_t^\ell \rfloor - [s+\ell \mid k].$$

For $t = 0$, η_0^ℓ is nonnegative, and $C_{s+\ell}^p(k)$ can be strictly smaller than $C_s^p(k)$ only if $s+\ell \mid k$ and $\lfloor \eta_0^\ell \rfloor = 0$ but $s \nmid k$. By taking the floor of (3.12), we see that the first condition is satisfied only if $k = (s+\ell)(s-\ell) = s^2 - \ell^2$ for some ℓ . However, k cannot be smaller than s^2 , thus the only admissible value for ℓ is 0, in which case $C_s^p(k) = C_{s+\ell}^p(k)$.

For $t = 1$, η_1^ℓ is nonnegative, and $C_{s+\ell}^p(k) < C_s^p(k)$ only if $k = (s+\ell)(s-\ell+1)$ and $s \nmid k$. Since k must be larger than $s(s+1)$, the only two admissible values for ℓ are 0 and 1, but in both cases we have that $k = s(s+1)$, and thus that $s \mid k$.

Finally, for $t = 2$ and $k = s(s+2)$, observe that $C_{s+\ell}^p(k) \geq C_s^p(k)$ unless $\lfloor \eta_2^\ell \rfloor = -1$ and $s+\ell \mid k$. The former condition is satisfied if and only if $\ell = 1$, but in this case $s+1 \nmid s(s+2)$, since

$$\frac{s(s+2)}{s+1} = s + \frac{s}{s+1}$$

cannot be integer for $s > 0$. \square

3.2.2 Optimal degrees for the Paterson–Stockmeyer method

We can characterize the degrees that are optimal for the Paterson–Stockmeyer method in the sense of Definition 3.1. In order to accomplish this task, we need to show that the cost function (3.5) is non-decreasing in k . Once again, this result is not obvious because of the terms $\left\lfloor \frac{k}{s} \right\rfloor$ and $[s \mid k]$ in (3.5).

Lemma 3.3. *The number of matrix multiplications required by the Paterson–Stockmeyer method to evaluate a matrix polynomial is non-decreasing in the degree of the polynomial.*

Proof. We want to show that, for $k \in \mathbb{N}_0$,

$$C^p(k) \leq C^p(k+1). \quad (3.13)$$

As floor and ceiling yield the same operation count, we can restrict ourselves to considering only $s = \lfloor \sqrt{k} \rfloor$ and $s' = \lfloor \sqrt{k+1} \rfloor$. If $s = s'$, then we only need to prove that $\left\lfloor \frac{k}{s} \right\rfloor \leq \left\lfloor \frac{k+1}{s} \right\rfloor$. By adding $\frac{1}{s}$ to all the terms in (3.8), we get that that, if $\left\lfloor \frac{k}{s} \right\rfloor = s + t$, then

$$s + t + \frac{1}{s} \leq \frac{k+1}{s} < s + t + 1 + \frac{1}{s'},$$

and thus that $\left\lfloor \frac{k+1}{s} \right\rfloor$ is either $s + t$ or $s + t + 1$, and cannot be smaller than $\left\lfloor \frac{k}{s} \right\rfloor$. Otherwise, we must have that $s' = s + 1$.

If $s \mid k$, then $k = s(s + t)$, for $t = 0, 1$, or 2 , and observing that

$$\frac{k+1}{s+1} = \frac{s^2 + st + 1}{s+1} = s + t - 1 + \frac{2-t}{s+1},$$

we can conclude that $\left\lfloor \frac{k+1}{s+1} \right\rfloor = s + t - 2$. Therefore, if $t = 0$ or 1 , then $s + 1 \nmid k + 1$ and the inequality (3.13) holds strictly, whereas if $t = 2$, then $k + 1 = (s + 1)^2$ and the equality is satisfied.

If $s + 1 \mid k + 1$ and $s \nmid k$, then $\lfloor \sqrt{k+1} \rfloor = s + 1$ and $\lfloor \sqrt{k} \rfloor = s$, which implies that $(s + 1)^2 \leq k + 1$ and $k + 1 < (s + 1)^2 + 1$, respectively. By dividing both inequalities by

$s + 1$, we get that $\frac{k+1}{s+1} = s + 1$, which readily implies that $k = (s + 1)^2 - 1 = s(s + 2)$.

Substituting these values into (3.13) shows that equality holds in this case.

Finally, when $s \nmid k$ and $s + 1 \nmid k + 1$, by multiplying all the terms in (3.8) by s , incrementing them by one, and dividing them by $s + 1$, one gets

$$s + t - 1 + \frac{s}{s + 1} \leq \frac{k + 1}{s + 1} < s + t + \frac{1 - t}{s + 1},$$

which implies that $\lfloor \frac{k+1}{s+1} \rfloor$ can be either $s + t - 1$ or $s + t$. Substituting into (3.13) shows that the former satisfies the equality and the latter the strict inequality. \square

Recall that an integer a is a quarter-square, a perfect square, or an oblong number, if there exists $b \in \mathbb{N}$ such that $a = \lfloor b^2/4 \rfloor$, $a = b^2$, or $a = b(b + 1)$, respectively.

Proposition 3.4. *The degree of a polynomial is optimal for the Paterson–Stockmeyer algorithm if and only if it is a positive quarter-square.*

Proof. By Lemma 3.3, a degree $k \in \mathbb{N}_0$ is optimal if and only if $C^p(k) < C^p(k + 1)$. Since positive quarter-squares are either positive perfect squares or positive oblong numbers, we need to prove only that $C^p(k) < C^p(k + 1)$ if and only if $k = s^2$ or $k = s(s + 1)$ for some $s \in \mathbb{N}_0$. We have that $\lfloor \sqrt{k} \rfloor = \lfloor \sqrt{k + 1} \rfloor = s$, and it is straightforward to verify that $C^p(s^2) = 2s - 2 < 2s - 1 = C^p(s^2 + 1)$ and $C^p(s(s + 1)) = 2s - 1 < 2s = C^p(s(s + 1) + 1)$, and thus that s^2 and $s(s + 1)$ are optimal degrees for all $s \in \mathbb{N}_0$.

Conversely, let $k \in \mathbb{N}_0$ be an optimal degree for the Paterson–Stockmeyer method, and let $s = \lfloor \sqrt{k} \rfloor$. Note that if k is not an integer multiple of s , then a polynomial with $s - (k \bmod s)$ more terms can be evaluated with the same number of matrix multiplications. Therefore, if k is optimal, then $s \mid k$ and, as a consequence of (3.7), k must be of the form $s(s + t)$, where $t = 0, 1$, or 2 . We already know that if $t = 0$ or $t = 1$, then k is optimal, and we need to show only that $k' := s(s + 2)$ is not. Since $k' + 1 = (s + 1)^2$, we have that $\sqrt{k' + 1} \mid k' + 1$, and thus that $C^p(k') = 2s = C^p(k' + 1)$, which shows that k' is not optimal. \square

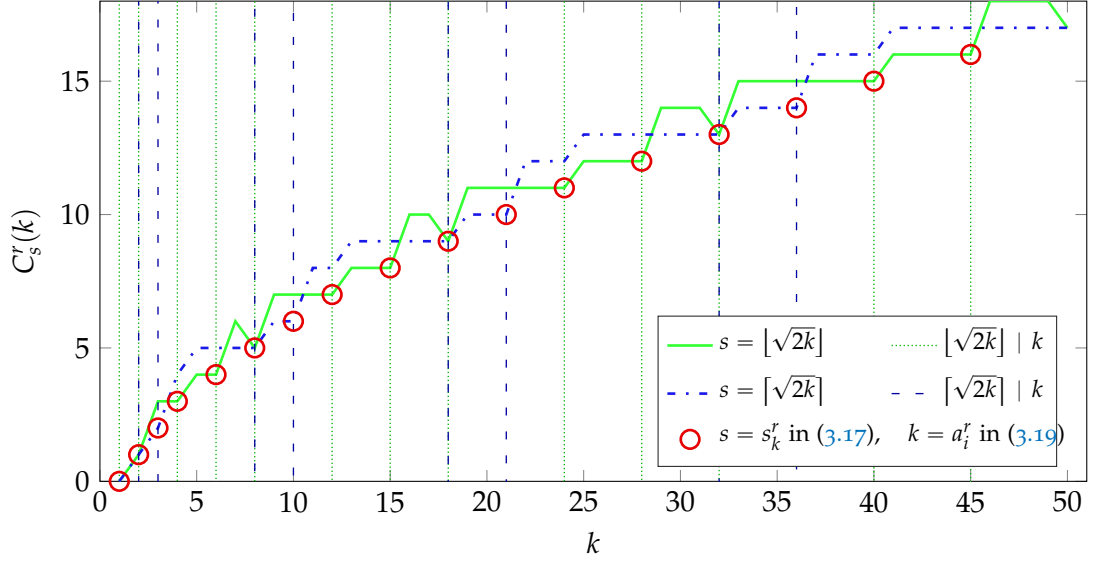


Figure 3.2: Number of matrix multiplications required to evaluate a rational function of order $[k/k]$, for k between 1 and 50, by means of the scheme (3.15), for $s = \lfloor \sqrt{2k} \rfloor$ and $s = \lceil \sqrt{2k} \rceil$. The dotted and dashed lines mark the values of k that are integer multiples of $\lfloor \sqrt{2k} \rfloor$ and $\lceil \sqrt{2k} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate rational matrix functions of optimal order (in the sense of Definition 3.1) for the evaluation scheme (3.15).

Therefore, the sequence of optimal degrees for the Paterson–Stockmeyer method is $(a_i^p)_{i \in \mathbb{N}}$, where

$$a_i^p = \left\lfloor \frac{(i+2)^2}{4} \right\rfloor. \quad (3.14)$$

By observing that $C^p(a_i^p) = i$, we can conclude that the polynomial of highest degree that can be evaluated with i matrix multiplications is that of degree a_i^p .

3.3 RATIONAL MATRIX FUNCTIONS OF ORDER $[k/k]$

A rational function is the quotient of two polynomials and, in the matrix case, it can be interpreted as the solution to a multiple right-hand side linear system whose coefficients and constant term are polynomials of the same matrix. Therefore, the value of a rational function at a matrix argument can be computed by relying on a suitable modification of the scheme (3.3) capable of minimizing the number of ma-

trix multiplications required to evaluate at once two polynomials at the same matrix argument.

Since in algorithms for computing matrix functions the evaluation of diagonal approximants is typically needed, in this section we focus on the evaluation of rational matrix functions of order $[k/k]$. Let us consider the task of evaluating $r(A) = q(A)^{-1}p(A)$, where both p and q are polynomials of degree $k \in \mathbb{N}_0$. We can rewrite numerator and denominator of this rational function as polynomials in A^s , which gives

$$p(A) = \sum_{i=0}^{\nu} B_i^{[p]}(A)(A^s)^i, \quad q(A) = \sum_{i=0}^{\nu} B_i^{[q]}(A)(A^s)^i, \quad \nu = \left\lfloor \frac{k}{s} \right\rfloor. \quad (3.15)$$

If this scheme is used and A^2, A^3, \dots, A^s are computed only once, then evaluating $r(A)$ requires the solution of one multiple right-hand side linear system and

$$C_s^r(k) := s - 1 + 2 \left\lfloor \frac{k}{s} \right\rfloor - 2[s \mid k] \quad (3.16)$$

matrix multiplications. The continuous relaxation of (3.16) is minimized by taking $s = \sqrt{2k}$, but, as Figure 3.2 shows, depending on k , either taking the floor or the ceiling of this quantity may yield the lowest flop count. Therefore, for $k \in \mathbb{N}_0$, we define

$$s_k^r := \arg \min \left\{ C_{\lfloor \sqrt{2k} \rfloor}^r(k), C_{\lceil \sqrt{2k} \rceil}^r(k) \right\}. \quad (3.17)$$

Figure 3.2 seems to suggest that if either rounding of $\sqrt{2k}$ divides k , then setting s to it in (3.15) will give $C_{s_k^r}^r(k)$. In the following we prove that, when that happens, s_k^r in fact minimizes the cost function $C_s^r(k)$ among all possible choices of s .

Lemma 3.5. *Let $A \in \mathbb{C}^{n \times n}$ and let p and q be polynomials of degree $k \in \mathbb{N}_0$. If $\lfloor \sqrt{2k} \rfloor \mid k$ or $\lceil \sqrt{2k} \rceil \mid k$, then setting s in (3.15) to $\lfloor \sqrt{2k} \rfloor$ or $\lceil \sqrt{2k} \rceil$, respectively, minimizes the number of matrix multiplications required to evaluate both $p(A)$ and $q(A)$ by the scheme (3.15).*

Proof. Let $\bar{s} = \lfloor \sqrt{2k} \rfloor$. By definition of the floor operator, $\bar{s}^2 \leq 2k < (\bar{s} + 1)^2$, and thus

$$\frac{\bar{s}}{2} \leq \frac{k}{\bar{s}} < \frac{\bar{s}}{2} + 1 + \frac{1}{2\bar{s}}.$$

Since $\bar{s} \mid k$, we have that $\frac{k}{\bar{s}} = \frac{\bar{s}+t}{2}$, where $t = 0$ or 2 if \bar{s} is even and $t = 1$ if \bar{s} is odd, and thus that $C_{\bar{s}}^r(k) = 2\bar{s} + t - 3$. In order to determine the number of multiplications required when setting $s \neq \bar{s}$ in (3.15), note that for $\ell \in \mathbb{N}$ such that $\ell > -\bar{s}$, we have

$$\frac{k}{\bar{s} + \ell} = \frac{\bar{s}(\bar{s} + t)}{2(\bar{s} + \ell)} = \frac{1}{2} \left(\bar{s} - \ell + t + \eta_t^\ell \right), \quad \eta_t^\ell := \frac{\ell^2 - t\ell}{\bar{s} + \ell}. \quad (3.18)$$

If $\bar{s} + \ell \mid k$, then $\eta_t^\ell \geq -\frac{1}{\bar{s} + \ell} > -1$, thus $\left\lfloor \frac{k}{\bar{s} + \ell} \right\rfloor \geq \frac{\bar{s} - \ell + t}{2}$ and $C_{\bar{s} + \ell}^r(k) \geq 2\bar{s} + t - 3 = C_{\bar{s}}^r(k)$. On the other hand, if $\bar{s} + \ell \nmid k$, then $\left\lfloor \frac{k}{\bar{s} + \ell} \right\rfloor \geq \frac{\bar{s} - \ell - t - 1}{2}$, and $C_{\bar{s} + \ell}^r(k) \geq 2\bar{s} + t - 2 > C_{\bar{s}}^r(k)$.

The proof for $\bar{s} = \lceil \sqrt{2k} \rceil$ is rather similar. From $(\bar{s} - 1)^2 < 2k \leq \bar{s}^2$ we have that

$$\frac{\bar{s}}{2} - 1 + \frac{1}{2\bar{s}} < \frac{k}{\bar{s}} \leq \frac{\bar{s}}{2},$$

and since $\bar{s} \mid k$, that $\frac{k}{\bar{s}} = \frac{\bar{s}-t}{2}$, for $t = 0$ or $t = 1$. For $\ell > -\bar{s}$, one has that $\frac{k}{\bar{s} + \ell} = \frac{1}{2}(\bar{s} - \ell - t + \eta_{-t}^\ell)$, and we can argue as above that if $\bar{s} + \ell \mid k$ then $C_{\bar{s} + \ell}^r(k) \geq 2\bar{s} - t - 3 = C_{\bar{s}}^r(k)$, while if $\bar{s} + \ell \nmid k$, then $C_{\bar{s} + \ell}^r(k) \geq 2\bar{s} - t - 2 > C_{\bar{s}}^r(k)$. \square

In order to characterize the optimal degrees for the scheme (3.15), we need to define the cost function $C^r(k) = \min_{1 \leq s \leq k} \{C_s^r(k)\}$, which represents the number of matrix multiplications needed to evaluate a diagonal rational function by means of (3.15) over all reasonable choices of s . In analogy with quarter-squares, we say that $a \in \mathbb{N}$ is an eight-square if there exists $b \in \mathbb{N}$ such that $a = \lfloor b^2/8 \rfloor$.

Proposition 3.6. *The degree of numerator and denominator of a rational function is optimal for the evaluation scheme (3.15) if and only if it is a positive eight-square.*

Proof. Let $r = p/q$, where p and q are polynomials of degree $k \in \mathbb{N}_0$. Note that when $s \nmid k$, then adding $s - (k \bmod s)$ more terms to p and q does not increase the number of matrix multiplications required by the scheme (3.15), thus we only need to consider cases where k is an integer multiple of s .

Let us begin by showing that if k is a positive eight-square then it is optimal. Note that $k = x(2x + t)$, for some $x \in \mathbb{N}_0$, if $k \equiv t \pmod{4}$ and $t = 0, 1$, or 2 , and that $k = (2x + 1)(x + 1)$ for some $x \in \mathbb{N}$, if $k \equiv 3 \pmod{4}$. We consider the four cases separately. In the following, we always assume that $\ell \in \mathbb{Z}$ is such that $\ell > -s$ and that $j \in \mathbb{N}$.

If $k = 2x^2$, then $s = \sqrt{2k} = 2x$, and since $s \mid k$, by Lemma 3.5 the minimum number of matrix multiplications required to evaluate $r(A)$ is $C_s^r(k) = 2s - 3$. Since

$$\frac{k+j}{s+\ell} = \frac{1}{2} \left(s - \ell + \eta_j^\ell \right), \quad \eta_j^\ell := \frac{\ell^2 + 2j}{s + \ell},$$

and $\eta_j^\ell > 0$, we have that $s + \ell \mid k + j$ only if $\eta_j^\ell \geq 1$, which implies that $C_{s+\ell}^r(k+j) \geq 2s - 2 > C_s^r(k)$.

If $k = x(2x + 1)$, then k is an integer multiple of $s = \lceil \sqrt{2k} \rceil = 2x + 1$, thus $C_s^r(k) = 2s - 4$ and

$$\frac{k+j}{s+\ell} = \frac{1}{2} \left(s - \ell - 1 + \eta_j^\ell \right), \quad \eta_j^\ell := \frac{\ell^2 + \ell + 2j}{s + \ell},$$

Being strictly positive, η_j^ℓ must be at least 1 for $s + \ell$ to divide $k + j$, which implies that $C_{s+\ell}^r(k+j) \geq 2s - 3 > C_s^r(k)$.

If $k = 2x(x + 1)$, then $s = \lfloor \sqrt{2k} \rfloor = 2x$, and $C_s^r(k) = 2s - 1$. On the other hand,

$$\frac{k+j}{s+\ell} = \frac{1}{2} \left(s - \ell + 2 + \eta_j^\ell \right), \quad \eta_j^\ell := \frac{\ell^2 - 2\ell + 2j}{s + \ell},$$

where as before $\eta_j^\ell > 0$. In order for $s + \ell$ to divide $k + j$, we have that η_j^ℓ must be at least 1, which in turn gives that $C_{s+\ell}^r(k+j) = 2s > C_s^r(k)$.

Finally, if $k = (2x + 1)(x + 1)$, then $s = \lceil \sqrt{2k} \rceil = 2x + 1$ and $C_s^r(k) = 2s - 2$. Moreover

$$\frac{k+j}{s+\ell} = \frac{1}{2} \left(s - \ell + 1 + \eta_j^\ell \right), \quad \eta_j^\ell := \frac{\ell^2 - \ell + 2j}{s + \ell},$$

where $\eta_j^\ell > 0$. As before, since $s + \ell \mid k + j$ only if $\eta_j^\ell \geq 1$, we have that $C_{s+\ell}^r(k+j) = 2s - 1 > C_s^r(k)$.

We have established that all eight-squares are optimal degrees for the evaluation scheme (3.15). In order to prove that all optimal degrees are eight-squares, it suffices to note that for all $n \in \mathbb{N}$ there exists an eight-square k such that $C^r(k) = n$. By Definition 3.1, optimal orders must be unique, therefore all optimal degrees must be eight-squares. \square

In view of this result, the sequence of optimal orders for the scheme (3.15) with $s = s_k^r$ in (3.17) is $(a_i^r)_{i \in \mathbb{N}}$, where

$$a_i^r = \left\lfloor \frac{(i+3)^2}{8} \right\rfloor. \quad (3.19)$$

Moreover, since $C^r(a_i^r) = i$, the rational function of highest order that can be evaluated with i matrix multiplications is that of order $[a_i^r/a_i^r]$.

3.4 DIAGONAL PADÉ APPROXIMANTS TO THE MATRIX EXPONENTIAL

Let $r = p/q$ be the $[k/k]$ diagonal Padé approximant to the exponential. The evaluation of these rational matrix functions deserves special attention, as the identity $q(x) = p(-x)$ allows for a much faster evaluation of r at a matrix argument. Let $\mu_k^e = \lfloor k/2 \rfloor$ and $\mu_k^o = \lfloor (k-1)/2 \rfloor$. By separating the $\mu_k^e + 1$ powers of A of even degree from the $\mu_k^o + 1$ powers of odd degree, we can write

$$\begin{aligned} p(A) &= \sum_{i=0}^k c_i A^i = \sum_{i=0}^{\mu_k^e} c_{2i} A^{2i} + A \sum_{i=0}^{\mu_k^o} c_{2i+1} A^{2i} =: U_e(A^2) + AU_o(A^2), \\ q(A) &= p(-A) = U_e(A^2) - AU_o(A^2), \end{aligned}$$

which shows that once $U_e(A^2)$ and $AU_o(A^2)$ are available, evaluating $p(A)$ and $q(A)$ requires no additional matrix multiplication.

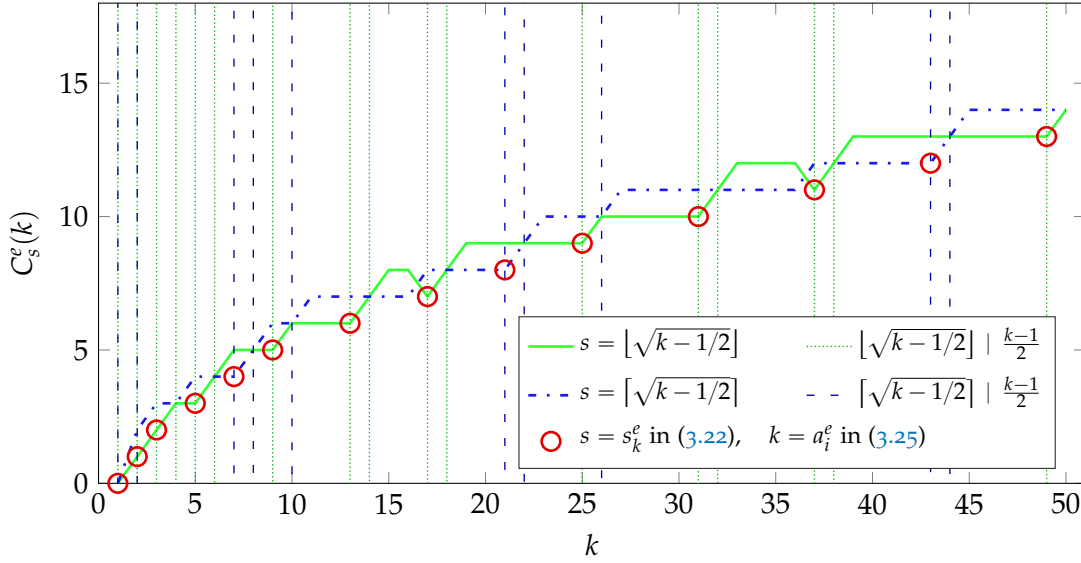


Figure 3.3: Number of matrix multiplications required to evaluate $[k/k]$ Padé approximant to the matrix exponential, for k between 1 and 50, by means of the scheme (3.20), for $s = \lfloor \sqrt{k-1/2} \rfloor$ and $s = \lceil \sqrt{k-1/2} \rceil$. The dotted and dashed lines mark the values of k for which $\frac{k-1}{2}$ is an integer multiple of $\lfloor \sqrt{k-1/2} \rfloor$ and $\lceil \sqrt{k-1/2} \rceil$, respectively; the circles mark the number of matrix multiplications required to evaluate the diagonal Padé approximants to the matrix exponential of optimal order (in the sense of Definition 3.1) for the evaluation scheme (3.20).

As $U_e(A^2)$ and $U_o(A^2)$ are polynomials in A^2 , they can be evaluated by means of the scheme

$$U_e(A^2) = \sum_{i=0}^{v_e} B_i^{[U_e]}(A^2) (A^{2s})^i, \quad U_o(A^2) = \sum_{i=0}^{v_o} B_i^{[U_o]}(A^2) (A^{2s})^i, \quad (3.20)$$

where $v_e = \lfloor \mu_k^e/s \rfloor$ and $v_o = \lfloor \mu_k^o/s \rfloor$, and the powers of A^2 are computed only once. Computing A^2, A^4, \dots, A^{2s} requires s matrix multiplications, evaluating the polynomials $U_e(A^2)$ and $U_o(A^2)$ require $\left\lfloor \frac{\mu_k^e}{s} \right\rfloor - [s \mid \mu_k^e]$ and $\left\lfloor \frac{\mu_k^o}{s} \right\rfloor - [s \mid \mu_k^o]$, respectively, and one additional matrix multiplication is needed to compute $AU_o(A^2)$. Therefore evaluating $r(A)$ requires one matrix inversion and

$$C_s^e(k) := s + 1 + \left\lfloor \frac{\mu_k^e}{s} \right\rfloor + \left\lfloor \frac{\mu_k^o}{s} \right\rfloor - [s \mid \mu_k^e] - [s \mid \mu_k^o] \quad (3.21)$$

matrix multiplications. The continuous relaxation of (3.21) is approximately minimized by taking $s = \sqrt{k - \frac{1}{2}}$ but, as Figure 3.3 shows, the two roundings do not typically yield the same computational cost.

Therefore, as in (3.17) we define

$$s_k^e := \arg \min \left\{ C_{\lfloor \sqrt{k-1/2} \rfloor}^e(k), C_{\lfloor \sqrt{k-1/2} \rfloor}^e(k) \right\}. \quad (3.22)$$

Lemma 3.7. *Let $A \in \mathbb{C}^{n \times n}$, let $k \in \mathbb{N}_0$ be odd, let p and q be the numerator and denominator of the $[k/k]$ Padé approximant to the exponential, respectively, and let $s_f = \lfloor \sqrt{k-1/2} \rfloor$ and $s_c = \lfloor \sqrt{k-1/2} \rfloor$. If $s_f \mid \frac{k-1}{2}$ or $s_c \mid \frac{k-1}{2}$, then setting s to s_f or s_c , respectively, minimizes the number of matrix multiplications required to evaluate both $q(A)$ and $p(A)$ by means of the scheme (3.20).*

Proof. If k is odd, then $\mu_k^e = \mu_k^o = \frac{k-1}{2}$. For s_f , we have

$$\frac{k-1}{2s_f} = \frac{s_f + t}{2}, \quad (3.23)$$

where $t = 0$ or 2 , if s_f is even, and $t = 1$, if s_f is odd, and it is easy to see that $C_{s_f}^e(k) = 2s_f + t - 1$. From (3.23), we have that $k-1 = s_f(s_f + t)$, thus for $\ell > -s_f$

$$C_{s_f+\ell}^e(k) \geq \begin{cases} s_f + \ell + 2 \lfloor \theta_t^\ell \rfloor - 1, & s_f + \ell \mid \frac{k-1}{2}, \\ s_f + \ell + 2 \lfloor \theta_t^\ell \rfloor + 1, & s_f + \ell \nmid \frac{k-1}{2}, \end{cases}$$

where

$$\theta_t^\ell := \frac{s_f - \ell + t + \eta_t^\ell}{2}, \quad \eta_t^\ell := \frac{\ell^2 - t\ell}{s_f + \ell}.$$

If $s_f + \ell \mid \frac{k-1}{2}$, then $C_{s_f+\ell}^e(k) \geq C_{s_f}^e(k)$ if and only if $\lfloor \theta_t^\ell \rfloor \geq \frac{s_f - \ell + t}{2}$. Note that, for $\alpha, \beta \in \mathbb{R}^+$, we have that $\lfloor \alpha \rfloor < \beta$ if and only if $\alpha < \lfloor \beta \rfloor$, and since $s_f + t$ is even, $s_f - \ell + t$ has the same parity as ℓ . Therefore, we only need to show that there exists no $\ell > -s_f$ such that

$$\theta_t^\ell < \left\lfloor \frac{s_f - \ell + t}{2} \right\rfloor = \begin{cases} \frac{s_f - \ell + t}{2}, & \ell \text{ is even,} \\ \frac{s_f - \ell + t + 1}{2}, & \ell \text{ is odd.} \end{cases}$$

These two conditions are equivalent to η_t^ℓ being strictly smaller than 0 and 1, respectively. However, since $s_f + \ell \mid \frac{k-1}{2}$, the quantity η_t^ℓ must be an integer and have the same parity as ℓ , and we need to ensure only that there are no values of ℓ such that $\eta_t^\ell \leq -2$ or $\eta_t^\ell \leq -1$. It is easy to check that for t between 0 and 2, $\eta_t^\ell \leq -2$ is equivalent to $\ell^2 + (2-t) + 2s_f \leq 0$, which has no even solutions, whereas $\eta_t^\ell \leq -1$ is equivalent to $\ell^2 + (1-t) + s_f \leq 0$, which has no odd solutions.

If $s_f + \ell \nmid \frac{k-1}{2}$, then by the same argument we conclude that we need to prove that there exists no $\ell > -s_f$ such that

$$\theta_t^\ell < \left\lceil \frac{s_f - \ell + t - 2}{2} \right\rceil = \begin{cases} \frac{s_f - \ell + t - 2}{2}, & \ell \text{ is even,} \\ \frac{s_f - \ell + t - 1}{2}, & \ell \text{ is odd.} \end{cases}$$

These two conditions lead to the inequalities $\eta_t^\ell < -2$ and $\eta_t^\ell < -1$, which have no solution for t between 0 and 2, as discussed above.

The proof for s_c is similar. In this case, we have that $s_c \mid \frac{k-1}{2}$ if and only if

$$\frac{k-1}{2s_c} = \frac{s_c-1}{2},$$

and thus that $C_{s_c}^e(k) = 2s_c - 2$. It is easy to show that, for $\ell > -s_c$,

$$C_{s_c+\ell}^e(k) \geq \begin{cases} s_c + \ell + 2 \left\lfloor \theta^\ell \right\rfloor - 1, & s_c + \ell \mid \frac{k-1}{2}, \\ s_c + \ell + 2 \left\lfloor \theta^\ell \right\rfloor + 1, & s_c + \ell \nmid \frac{k-1}{2}, \end{cases}$$

where

$$\theta^\ell := \frac{s_c - \ell - 1 + \eta^\ell}{2}, \quad \eta^\ell := \frac{\ell^2 + \ell}{s_c + \ell}.$$

Therefore, if $s_c + \ell \mid \frac{k-1}{2}$, we only have to prove that there exists no $\ell > -s_c$ such that

$$\theta^\ell < \left\lceil \frac{s_c - \ell - 1}{2} \right\rceil = \begin{cases} \frac{s_c - \ell - 1}{2}, & \ell \text{ is even,} \\ \frac{s_c - \ell}{2}, & \ell \text{ is odd,} \end{cases}$$

or, in other words, that $\eta^\ell < 0$ if ℓ is even, and $\eta^\ell < -1$ if ℓ is odd. Both conditions are trivially satisfied, since $\eta^\ell \geq 0$ for $|\ell| \geq 1$. Finally, if $s_c + \ell \nmid \frac{k-1}{2}$, we obtain the conditions $\eta^\ell < -1$ if ℓ is even and $\eta^\ell < -2$ if ℓ is odd, both of which clearly satisfy since η^ℓ is nonnegative. \square

We are now ready to characterize the optimality of the Paterson–Stockmeyer method for the diagonal Padé approximants to the matrix exponential.

Proposition 3.8. *A degree $k \in \mathbb{N}_0$ is optimal for the evaluation scheme (3.20) if and only if $k = 2$ or*

$$k = 2 \left\lceil \frac{y}{4} \right\rceil \left(y - 2 \left\lfloor \frac{y-1}{4} \right\rfloor \right) + 1, \quad (3.24)$$

for some $y \in \mathbb{N}$.

Proof. First, note that for k to be optimal, both μ_k^e and μ_k^o must be integer multiples of s , since otherwise, we could add more terms at no cost until both conditions are satisfied. Therefore, if at least one of μ_k^e or μ_k^o is greater than 1, then k must be odd: if it were not, then $s \in \mathbb{N}_0$ could not divide both μ_k^o and $\mu_k^e = \mu_k^o + 1$.

It is easy to show that $k = 2$ is an optimal degree for the evaluation scheme (3.20). We have that $s = 1$, $\mu_k^o = 0$, and $\mu_k^e = 1$, which gives $C_1^e(2) = 1$, and

$$\frac{2+j}{2(1+\ell)} = \frac{1}{2} \left(1 - \ell + \eta_j^\ell \right), \quad \eta_j^\ell := \frac{2\ell^2 + j + 1}{1 + \ell}.$$

Since η_j^ℓ is strictly positive, if $1 + \ell \nmid \frac{2+j}{2}$, then $C_{1+\ell}^e(2+j) \geq 2 > C_1^e(2)$, whereas if $1 + \ell \mid \frac{2+j}{2}$, then η_j^ℓ must be an integer larger than 2, which again gives $C_{1+\ell}^e(2+j) \geq 2 > C_1^e(2)$.

It is convenient to split the expression for k into four cases that allow us to get rid of the floor and ceiling operators in (3.24). To that end, we note that if $k \equiv \tilde{t} \pmod{4}$, then $k = 2x(2x + t) + 1$, for some $x \in \mathbb{N}$ and $t = \tilde{t} - 2$.

The three cases $|t| \leq 1$ can be addressed together. We have that $s = 2x + t$ or, equivalently, that $x = \frac{s-t}{2}$, and since $\frac{k-1}{2s} = x$, we can conclude that $C_s^e(k) = 4x + t - 1$. Now let $\ell \in \mathbb{Z}$ be such that $\ell > -s$ and let $j \in \mathbb{N}_0$. We have that

$$\frac{k+j-1}{2(s+\ell)} = \frac{1}{2} \left(\frac{s(s-t)+j}{s+\ell} \right) = \frac{1}{2} (s-\ell-t+\eta_{t,j}^\ell), \quad \eta_{t,j}^\ell := \frac{\ell^2 - t\ell + j}{s+\ell}.$$

Note that $\eta_{t,j}^\ell > 0$. If $s+\ell \nmid \frac{k+j-1}{2}$, then $C_{s+\ell}^e(k+j) \geq 4x+t+1 > C_s^e(k)$. On the other hand, if $s+\ell \mid \frac{k+j-1}{2}$, then $\eta_{t,j}^\ell$ must be a positive integer in order for $\frac{k+j-1}{2(s+\ell)}$ to be integer, which gives that $C_{s+\ell}^e(k+j) = 4x+t > C_s^e(k)$.

Finally we consider the case $t = 2$. From $s = 2x$, we get that $x = s/2$ and $k-1 = s(s+2)$, which gives $C_s^e(k) = 4x+1$. We have that

$$\frac{k+j-1}{2(s+\ell)} = \frac{1}{2} \left(\frac{s(s+2)+j}{s+\ell} \right) = \frac{1}{2} (s-\ell+2+\eta_j^\ell), \quad \eta_j^\ell := \frac{\ell^2 - 2\ell + j}{s+\ell}.$$

It is easy to see that η_j^ℓ is nonnegative, and in particular that $\eta_j^\ell = 0$ only if $j = 1$ and $\ell = 1$. Thus, if $s+\ell \nmid \frac{k+j-1}{2}$, then $C_{s+\ell}^e(k+j) \geq 4x+3 > C_s^e(k)$. When $s+\ell \mid \frac{k+j-1}{2}$, on the other hand, since $s+\ell \nmid \frac{k}{2}$ and η_j^ℓ is positive, in particular η_j^ℓ must be larger than 1. Therefore, we have that $C_{s+\ell}^e(k+j) \geq 4x+2 > C_s^e(k)$.

The converse follows from the same argument as that used in the proof of the analogous result in Proposition 3.6. \square

In view of Proposition 3.8, the sequence of optimal degrees for the evaluation scheme (3.20) is $(a_i^e)_{i \in \mathbb{N}}$, where

$$\begin{aligned} a_0^e &= 1, \\ a_1^e &= 2, \\ a_i^e &= 2 \left\lceil \frac{i-1}{4} \right\rceil \left(i - 3 \left\lfloor \frac{i-1}{4} \right\rfloor \right) + 1, \quad i > 2. \end{aligned} \tag{3.25}$$

Moreover, we have that $C^e(a_i^e) = i$ and that the diagonal Padé approximant to the matrix exponential of highest order that can be evaluated with i matrix multiplications is that of degree $[a_i^e/a_i^e]$.

3.5 CONCLUSION

The scheme (3.3), which gives rise to the Paterson–Stockmeyer method, and the related evaluation schemes (3.15) and (3.20), are customary tools for evaluating truncated Taylor series and diagonal Padé approximants. They all feature a parameter, s , which is usually chosen by approximately solving an optimization problem over the integers. For the evaluation of polynomials of matrices, we showed that the Paterson–Stockmeyer choices $s = \lfloor \sqrt{k} \rfloor$ and $s = \lceil \sqrt{k} \rceil$ always minimize the number of matrix multiplications required to evaluate a polynomial of degree k . For the evaluation of diagonal approximants, we gave sufficient conditions for the parameter s to minimize the computational cost of the corresponding evaluation schemes. Tests not reported here suggest that, for all $k \in \mathbb{N}_0$, the choices $s = s_k^r$ in (3.17) and $s = s_k^e$ in (3.22) minimize the number of matrix multiplications required by the schemes (3.15) and (3.20), respectively, and we believe that exploring this question further might lead to results similar to that in Proposition 3.2 for the Paterson–Stockmeyer method.

When relying on polynomial or rational approximation to evaluate matrix functions, one is usually interested only in approximants whose order is maximal for a given computational cost. By exploiting the results discussed above, we showed that the sequences of optimal orders (in the sense of Definition 3.1) for the three evaluation schemes (3.3), (3.15), and (3.20), are (3.14), (3.19), and (3.25), respectively. We wonder whether similar results can be derived for rational functions of any order, and more generally, for schemes that require the evaluation of three or more polynomials of any degree. This will be the subject of future work.

ACKNOWLEDGEMENTS

The author thanks Stefan Güttel, Nicholas J. Higham, Bruno Iannazzo, Froilán Dopico, and the two anonymous referees for reading early drafts of the manuscript and providing comments that greatly improved the presentation of this work.

BIBLIOGRAPHY

- [1] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989.
- [2] ———, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169.
- [3] A. H. AL-MOHY, N. J. HIGHAM, AND S. D. RELTON, *New algorithms for computing the matrix sine and cosine separately or simultaneously*, SIAM J. Sci. Comput., 37 (2015), pp. A456–A487.
- [4] P. ALONSO, J. IBÁÑEZ, J. SASTRE, J. PEINADO, AND E. DEFEZ, *Efficient and accurate algorithms for computing matrix trigonometric functions*, J. Comput. Appl. Math, 309 (2017), pp. 325–332.
- [5] M. APRAHAMIAN AND N. J. HIGHAM, *Matrix inverse trigonometric and inverse hyperbolic functions: Theory and algorithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1453–1477.
- [6] M. CALIARI AND F. ZIVCOVICH, *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*, J. Comput. Appl. Math, 346 (2019), pp. 532–548.
- [7] S. H. CHENG, N. J. HIGHAM, C. S. KENNEY, AND A. J. LAUB, *Approximating the logarithm of a matrix to specified accuracy*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1112–1125.
- [8] E. DEFEZ, J. IBÁÑEZ, J. SASTRE, J. PEINADO, AND P. ALONSO, *A new efficient and accurate spline algorithm for the matrix exponential computation*, J. Comput. Appl. Math, 337 (2018), pp. 354–365.
- [9] M. FASI AND N. J. HIGHAM, *An arbitrary precision scaling and squaring algorithm for the matrix exponential*, MIMS EPrint 2018.36, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2018.

- [10] M. FASI AND N. J. HIGHAM, *Multiprecision algorithms for computing the matrix logarithm*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 472–491.
- [11] S. GÜTTEL AND Y. NAKATSUKASA, *Scaled and squared subdiagonal Padé approximation for the matrix exponential*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 145–170.
- [12] G. HARGREAVES, *Topics in Matrix Computations: Stability and Efficiency of Algorithms*, PhD thesis, University of Manchester, Manchester, England, 2005.
- [13] N. J. HIGHAM, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 1179–1193.
- [14] ———, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [15] N. HOFFMAN, O. SCHWARTZ, AND S. TOLEDO, *Efficient evaluation of matrix polynomials*, Parallel Proc. Appl. Math., (2018), pp. 24–35.
- [16] D. KRESSNER AND R. LUCE, *Fast computation of the matrix exponential for a Toeplitz matrix*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 23–47.
- [17] M. S. PATERSON AND L. J. STOCKMEYER, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66.
- [18] J. SASTRE, *Efficient evaluation of matrix polynomials*, Linear Algebra Appl., 539 (2018), pp. 229–250.
- [19] J. SASTRE, J. IBÁÑEZ, P. ALONSO, J. PEINADO, AND E. DEFEZ, *Two algorithms for computing the matrix cosine function*, J. Comput. Appl. Math, 312 (2017), pp. 66–77.
- [20] J. SASTRE, J. IBÁÑEZ, AND E. DEFEZ, *Boosting the computation of the matrix exponential*, Appl. Math. Comput., 340 (2019), pp. 206–220.
- [21] J. SASTRE, J. IBÁÑEZ, E. DEFEZ, AND P. RUIZ, *New scaling-squaring Taylor algorithms for computing the matrix exponential*, SIAM J. Matrix Anal. Appl., 37 (2015), pp. A439–A455.

- [22] C. VAN LOAN, *A note on the evaluation of matrix polynomials*, IEEE Trans. Automat. Control, 24 (1979), pp. 320–321.

4

COMPUTING PRIMARY SOLUTIONS OF EQUATIONS INVOLVING PRIMARY MATRIX FUNCTIONS

Abstract. The matrix equation $f(X) = A$, where f is an analytic function and A is a square matrix, is considered. Some results on the classification of solutions are provided. When f is rational, a numerical algorithm is proposed to compute all solutions that can be written as a polynomial of A . For real data, the algorithm yields the real solutions using only real arithmetic. Numerical experiments show that the algorithm performs in a stable fashion when run in finite precision arithmetic.

Keywords: Schur normal form, block triangular matrices, substitution algorithm, matrix equation, matrix function.

2010 MSC: 15A16, 15A24, 65F60.

4.1 INTRODUCTION

We consider the matrix equation

$$f(X) = A, \tag{4.1}$$

where $A, X \in \mathbb{C}^{N \times N}$ and f is a complex function applied to a matrix (in the sense of primary matrix functions, see section 4.2). Remarkable examples of (4.1) are the matrix equations $X^k = A$, $e^X = A$, and $Xe^X = A$, which define the matrix k th root [22], [16], the matrix logarithm [1], and the matrix Lambert W function [8], respectively. Existence and finiteness of real and complex solutions to (4.1) are dis-

cussed, along with other properties of this matrix equation, in the excellent treatise by Evard and Uhlig [7].

In order to better understand the computational properties of the matrices that satisfy (4.1), it is useful to distinguish the solutions that can be written as a polynomial of A , or *primary* solutions, from those that cannot, called *nonprimary*. A useful characterization of primary solutions in terms of their eigenvalues is provided in [7].

After discussing some further properties of primary solutions, we focus our attention on *isolated* solutions, that is, solutions that are unique in a neighbourhood. We show that nonprimary solutions are not isolated, characterize isolated solutions in terms of their eigenvalues, and show that they are in fact primary solutions with some additional properties.

Turning to numerical computation, we restrict our attention to the equation

$$r(X) = A, \tag{4.2}$$

where $r = p/q$, and p and q are polynomials. The algorithm we propose is designed in the spirit of and generalizes the method developed by Björck and Hammarling [3] for the square root of a matrix, tailored for the real case by Higham [13] and extended to the k th root by Smith [22].

First, we consider the case of block upper triangular A and develop an algorithm that, using a sequence of substitutions, computes a primary solution to (4.2) given its diagonal blocks. Next we discuss how the Schur decomposition, which reduces any matrix to block upper triangular form with a similarity transformation, can be exploited to extend our approach to general matrices, and show that the algorithm, if no breakdown occurs, computes a primary solution, if it exists, given its eigenvalues. Finally, we show that the algorithm is applicable with no breakdown if and only if there exists a unique solution with given diagonal blocks (which correspond to a given set of eigenvalues), which, moreover, is proved to be equivalent to requiring that the solution is isolated.

Being restricted to isolated solutions is not a severe limitation, since solutions that are not isolated are typically of little or no computational interest. Indeed a solu-

tion \tilde{X} that is not isolated is either nonprimary or *ill-posed*, in the sense that there exists a neighborhood $\mathcal{U}_{\tilde{X}}$ of \tilde{X} and a matrix E , such that the perturbed equation $r(X) = A + tE$ has no solution in $\mathcal{U}_{\tilde{X}}$ for any sufficiently small $t > 0$. For instance, when computing the square root of a matrix A with the algorithm of Björck and Hammarling, one requires that, if A is singular, then the eigenvalue zero is simple [3], which is a necessary and sufficient condition for a primary solution to $X^2 = A$ to be isolated. Primary square roots can exist when the zero eigenvalue has multiplicity larger than one, but in this case they are not isolated, and there exist arbitrarily small perturbations of A having no square root.

In the next section, we provide some background material, and in the following we give some theoretical results regarding the solutions of matrix equations of the type (4.1). In section 4.4, we consider (4.2) and present our algorithm for block upper triangular matrices, discussing both the complex and the real Schur form. Section 4.5 is devoted to numerical experiments that illustrate the numerical behavior of our algorithm, and in section 4.6 we draw some conclusions and discuss lines of future research.

4.2 BACKGROUND AND NOTATION

Polynomials and rational functions. By convention, a summation is equal to zero if the starting index exceeds the ending one. We denote by $\mathbb{C}[z]$ the polynomials of the complex variable z with complex coefficients, and by $\mathbb{C}_k[z] \subset \mathbb{C}[z]$ the complex polynomials of degree at most k . Let $p(z) := \sum_{k=0}^m c_k z^k \in \mathbb{C}_m[z]$ and $q(z) := \sum_{k=0}^n d_k z^k \in \mathbb{C}_n[z]$ be coprime polynomials with nonzero leading coefficients. The quotient $r(z) := p(z)q(z)^{-1}$ is a rational function of type $[m, n]$. In the following sections, when using p , q , or r , we will always refer to the functions defined above, and in particular, c_0, \dots, c_m will denote the coefficients of p and d_0, \dots, d_n those of q .

In order to evaluate a polynomial p at a point x_0 , we make use of Horner's evaluation scheme [10, Alg. 9.2.1], that is, we define the polynomials $p^{[j]}(z) = \sum_{i=0}^{m-j} c_{i+j} z^i$, for $j = 0, \dots, m$, and evaluate $p^{[0]}(z_0) = p(z_0)$ by means of the recursion

$$\begin{aligned} p^{[m]}(z_0) &= c_m, \\ p^{[j]}(z_0) &= z_0 p^{[j+1]}(z_0) + c_j, \quad \text{for } j = 0, \dots, m-1. \end{aligned}$$

Let $f : \Omega \rightarrow \mathbb{C}$, where $\Omega \subset \mathbb{C}$, and let $x, y \in \Omega$. We denote by $f[x, y]$ the divided difference operator, defined by

$$f[x, y] = \begin{cases} f'(x), & x = y, \\ \frac{f(x) - f(y)}{x - y}, & x \neq y, \end{cases}$$

which implicitly requires f to be differentiable at x , when $x = y$. The divided differences over $k + 1$ numbers, ordered so that equal numbers are contiguous, are

$$f[x_0, \dots, x_k] = \begin{cases} \frac{f^{(k)}(x_0)}{k!}, & x_0 = x_1 = \dots = x_k, \\ \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0}, & \text{otherwise.} \end{cases}$$

This definition can be extended to any set of $k + 1$ numbers by assuming that the divided differences are symmetric functions of their arguments. For the construction above to make sense, the function has to be differentiable t times at any point repeated $t + 1$ times.

Primary matrix functions. Let $A \in \mathbb{C}^{N \times N}$ and let $Z \in \mathbb{C}^{N \times N}$ be such that $Z^{-1}AZ = J = \text{diag}(J(\lambda_1, \tau_1), \dots, J(\lambda_\nu, \tau_\nu))$ is the Jordan canonical form of A , with

$$J(\lambda, m) := \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix} \in \mathbb{C}^{m \times m},$$

where missing entries should be understood as zeros. In order to simplify the notation, we will often omit the diagonal element and the size of the Jordan block and write J_i for $J(\lambda_i, m_i)$. The index of the eigenvalue λ , denoted by $\iota(\lambda)$, is the size of the largest Jordan block where λ appears. An eigenvalue with index one is said to be *semisimple*, otherwise it is said to be *defective*; a semisimple eigenvalue appearing in only one block is said to be *simple*.

Let the complex function f and its derivatives up to the order $\iota(\lambda_k) - 1$ be defined at λ_k for $k = 1, 2, \dots, \nu$. Then we can define the primary matrix function

$$f(A) := Zf(J)Z^{-1} = Z \operatorname{diag}(f(J_1), f(J_2), \dots, f(J_\nu))Z^{-1}, \quad (4.3)$$

where

$$f(J_k) = \begin{bmatrix} f(\lambda_k) & f'(\lambda_k) & \dots & \frac{f^{(m_k-1)}(\lambda_k)}{(m_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & f'(\lambda_k) \\ & & & f(\lambda_k) \end{bmatrix}.$$

This definition does not depend on the matrix Z , and it can be shown that if f is a primary matrix function then $f(M^{-1}AM) = M^{-1}f(A)M$ for any M invertible and A such that $f(A)$ is well-defined. We will refer to this fundamental property as *commutativity with similarities*, and it will be used throughout the paper. A consequence is that if $A = \operatorname{diag}(A_1, \dots, A_\nu)$ is block diagonal, then $f(A) = \operatorname{diag}(f(A_1), \dots, f(A_\nu))$.

Moreover, it is easy to show that $f(A)$ as defined in (4.3) coincides with a polynomial that interpolates f in the Hermite sense on the spectrum of A [14, Rem. 1.10]. Therefore, if $T \in \mathbb{C}^{N \times N}$ is block upper triangular, then $f(T)$ has the same block structure as T , and if $T_{11}, \dots, T_{\nu\nu}$ are the diagonal blocks of T , then the diagonal blocks of

$f(T)$ are $f(T_{11}), \dots, f(T_{vv})$. An explicit formula for the function of an upper triangular matrix is [10, Thm. 9.1.4]

$$\begin{aligned} (f(T))_{ii} &= f(t_{ii}), & 1 \leq i \leq N, \\ (f(T))_{ij} &= \sum_{i_1=i < i_2 < \dots < i_\ell=j} t_{i_1 i_2} t_{i_2 i_3} \cdots t_{i_{\ell-1} i_\ell} f[t_{i_1 i_1}, \dots, t_{i_\ell i_\ell}], & 1 \leq i < j \leq N, \end{aligned} \quad (4.4)$$

where the sum is over all increasing sequences of integers starting with i and ending with j .

Let J be a nontrivial Jordan block in which the eigenvalue λ appears. The Jordan canonical form of $f(J)$ consists of:

1. only one Jordan block associated with $f(\lambda)$, if $f'(\lambda) \neq 0$;
2. two or more Jordan blocks associated with $f(\lambda)$, if $f'(\lambda) = 0$.

In the latter case, we say that the function f *splits the Jordan block* J . A complete description of the Jordan canonical form of $f(A)$ in terms of that of A is given in [15, sect. 6.2.25].

The Fréchet derivative of a matrix function $f : \Omega \rightarrow \mathbb{C}^{N \times N}$ at a point $A \in \Omega \subset \mathbb{C}^{N \times N}$ is the linear operator $Df(A) : \mathbb{C}^{N \times N} \rightarrow \mathbb{C}^{N \times N}$ that satisfies

$$f(A + E) = f(A) + Df(A)[E] + o(\|E\|),$$

for any $E \in \mathbb{C}^{N \times N}$ with sufficiently small norm.

A measure of the sensitivity of matrix functions, with respect to perturbation of the argument A , is given by the relative condition number, which, for any subordinate norm $\|\cdot\|$, is defined as [14, eq. (3.2)]

$$\kappa_f(A) = \lim_{\varepsilon \rightarrow 0} \sup_{\|E\| \leq \varepsilon \|A\|} \frac{\|f(A + E) - f(A)\|}{\varepsilon \|f(A)\|}. \quad (4.5)$$

We conclude this section with a lemma and a corollary that will be useful later on.

Lemma 4.1. Let $A \in \mathbb{C}^{N \times N}$ be upper bidiagonal, let e_i , for $i = 1, \dots, N$, be the standard basis of \mathbb{C}^N , and let $f(A)$ and $f[a_{11}, a_{NN}]$ be well-defined. Then for any $\delta \in \mathbb{C}$ we have that

$$f(A + \delta e_1 e_N^T) = f(A) + \delta f[a_{11}, a_{NN}] e_1 e_N^T. \quad (4.6)$$

Proof. Let $T := A + \delta e_1 e_N^T$ and $F = f(T)$. If we partition $T =: \begin{bmatrix} T_1 & v \\ 0 & t_{NN} \end{bmatrix}$, with $T_1 \in \mathbb{C}^{(N-1) \times (N-1)}$, from the properties of primary matrix functions, we have that $F = \begin{bmatrix} f(T_1) & \tilde{v} \\ 0 & f(t_{NN}) \end{bmatrix}$ and thus that $(F)_{ij} = (f(A))_{ij}$, for $j < N$. Using the partition $T = \begin{bmatrix} t_{11} & w \\ 0 & T_2 \end{bmatrix}$, with $T_2 \in \mathbb{C}^{(N-1) \times (N-1)}$, we get that $(F)_{ij} = (f(A))_{ij}$ for $i < N$. By using (4.4), for the top right element of the matrix we have

$$(F)_{1N} = \sum_{i_1=1 < i_2 < \dots < i_\ell=N} t_{i_1 i_2} t_{i_2 i_3} \dots t_{i_{\ell-1} i_\ell} f[t_{i_1 i_1}, t_{i_2 i_2}, \dots, t_{i_\ell i_\ell}].$$

Since $t_{ij} = 0$ for $i < j - 1$ and $(i, j) \neq (1, N)$, the sum can be restricted to the two sequences $i_1 = 1, i_2 = 2, \dots, i_N = N$ and $i_1 = 1, i_2 = N$, giving

$$(F)_{1N} = t_{12} \dots t_{N-1, N} f[t_{11}, \dots, t_{NN}] + t_{1N} f[t_{11}, t_{NN}] = (f(A))_{1N} + \delta f[a_{11}, a_{NN}],$$

which concludes the proof of the identity (4.6). \square

Corollary 4.2. We have the following relations, with $\delta \in \mathbb{C}$:

- (a) if $A = \lambda I$ and f is differentiable at λ , then $f(A + \delta e_1 e_N^T) = f(A) + \delta f'(\lambda) e_1 e_N^T$;
- (b) if $A = J(\lambda, N)$, and f is differentiable at λ , then $f(A + \delta e_1 e_N^T) = f(A) + \delta f'(\lambda) e_1 e_N^T$;
- (c) if $A = \text{diag}(J(\lambda, k), J(\mu, N - k))$, with $\lambda \neq \mu$ and $1 \leq k < N$, and f is well-defined at A , then $f(A + \delta e_1 e_N^T) = f(A) + \delta f[\lambda, \mu] e_1 e_N^T$.

4.3 CLASSIFICATION OF THE SOLUTIONS

The matrix equation $f(X) = A$, with f analytic, may have zero, finitely many, or infinitely many solutions. All these scenarios are possible, and here we are concerned

with a classification of the solutions in terms of properties that are relevant from a computational viewpoint. In section 4.3.1 we relate the notion of primary solution to that of primary matrix function, in section 4.3.2, we consider isolated solutions and characterize them in several ways, and we conclude by briefly discussing critical solutions in section 4.3.3.

4.3.1 Primary solutions

The matrices that satisfy (4.1) may define a function of the matrix A , but in general solutions to (4.1) need not be primary functions of A , in the sense of section 4.2. The matrix $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, for instance, satisfies the 2×2 matrix equation $X^2 = 0$, but is not a primary function of the zero matrix. Solutions to a matrix equation can be divided into two classes, those that are primary functions of A and those that are not. In this section, we give some clarifications on this topic.

Let $A \in \mathbb{C}^{N \times N}$, let $f : \Omega \rightarrow \mathbb{C}$ be a function analytic on the open set $\Omega \subseteq \mathbb{C}$ and let $X \in \mathbb{C}^{N \times N}$ be a solution to $f(X) = A$ such that f is defined on the spectrum of X . A solution is *primary* if it can be written as a polynomial of A , and *nonprimary* otherwise.

A necessary and sufficient condition for a solution to be primary is provided by the following result, where an eigenvalue ξ of the solution X is said to be *critical* if $f'(\xi) = 0$.

Theorem 4.3 (Evard and Uhlig [7, Thm. 6.1]). *A solution $X \in \mathbb{C}^{N \times N}$ to the equation $f(X) = A$ is primary if and only if the following two conditions are true:*

1. *for any two distinct eigenvalues ξ_1 and ξ_2 of X , we have $f(\xi_1) \neq f(\xi_2)$;*
2. *all critical eigenvalues of X (if any) are semisimple.*

The definition of primary solution as a polynomial of A is related to the concept of primary function of a matrix. Informally, we could say that any primary solution is obtained as “an inverse of f applied to the matrix A ”. We now make this notion precise.

Let $\lambda_1, \dots, \lambda_s$ be the distinct eigenvalues of A , ordered so that the first t are semisimple and the remaining are not. We say that a solution X is *primary in the sense of functions* if $X = \hat{f}^{-1}(A)$ where $\hat{f}^{-1} : \{\lambda_1, \dots, \lambda_t\} \cup \mathcal{U} \rightarrow \mathbb{C}$ is analytic on an open set $\mathcal{U} \supseteq \{\lambda_{t+1}, \dots, \lambda_s\}$ and is such that $(f \circ \hat{f}^{-1})(z) = \text{id}(z)$ for any $z \in \{\lambda_1, \dots, \lambda_t\} \cup \mathcal{U}$.

Requiring that \hat{f}^{-1} is analytic on the eigenvalues that correspond to nontrivial Jordan blocks of A guarantees that \hat{f}^{-1} is defined on the spectrum of A and thus that $\hat{f}^{-1}(A)$ is well-defined in the sense of (4.3). These two definitions are in fact the same, as the following proposition shows.

Proposition 4.4 (Equivalence of definitions of primary solution). *Let f be a complex function analytic on $\Omega \subset \mathbb{C}$ and let $A \in \mathbb{C}^{N \times N}$. A solution $X \in \mathbb{C}^{N \times N}$ to $f(X) = A$ with eigenvalues in Ω can be written as a polynomial of A if and only if it is primary in the sense of functions, i.e., if and only if $X = f^{-1}(A)$, where f^{-1} is an inverse of f defined on the spectrum of A and analytic at the defective eigenvalues of A .*

Proof. Assume that $X = f^{-1}(A)$ for some inverse of f . Since $f^{-1}(A)$ is a primary function of A , there exists a polynomial $p \in \mathbb{C}[z]$ such that $X = f^{-1}(A) = p(A)$, which implies that X is a primary solution to $f(X) = A$.

Conversely, suppose that $X = p(A)$ for some $p \in \mathbb{C}[z]$. From $f(X) = f(p(A)) = A$, it follows that $f(p(\lambda)) = \lambda$ for any eigenvalue λ of A . By taking $\hat{f}^{-1}(\lambda) = p(\lambda)$ for any λ , it is enough to show that if λ is not semisimple, then $\hat{f}^{-1}(\lambda)$ can be extended analytically in a neighborhood of λ to an inverse of f , and that $X = \hat{f}^{-1}(A)$.

Let J be a nontrivial Jordan block of A in which the eigenvalue λ appears. From $f(p(A)) = A$ it follows that $f(p(J)) = J$, which entails that $(f \circ p)'(\lambda) \neq 0$, as $f \circ p$ would otherwise split the Jordan block. The latter inequality implies, in turn, that $f'(p(\lambda)) \neq 0$ and thus that f is invertible in a neighborhood of $p(\lambda) = \hat{f}^{-1}(\lambda)$ with analytic inverse [9, sect. 4.6]. Thus, we can extend \hat{f}^{-1} in an open neighborhood of λ to a function such that $f \circ \hat{f}^{-1} = \text{id}$.

In order to prove that $X = \hat{f}^{-1}(A)$, it suffices to show that $(\hat{f}^{-1})^{(k)}(\lambda) = p^{(k)}(\lambda)$ for $k = 1, \dots, \ell - 1$, where ℓ is the size of \tilde{J} , the largest Jordan block in which λ appears. First observe that $f(p(\tilde{J})) = \tilde{J}$ implies that $(f \circ p)^{(k)}(\lambda) = \text{id}^{(k)}(\lambda)$ and thus that $(f \circ p)^{(k)}(\lambda) = (f \circ \hat{f}^{-1})^{(k)}(\lambda)$, for $k = 1, \dots, \ell - 1$, since $(f \circ \hat{f}^{-1})^{(k)}(\lambda) = \text{id}^{(k)}(\lambda)$.

Next, we show by induction that for any $k > 0$ and any function g such that $f \circ g$ is analytic in a neighborhood of λ , one has that

$$(f \circ g)^{(k)}(\lambda) = f'(g(\lambda))g^{(k)}(\lambda) + h_k(g; \lambda),$$

where h_k is a polynomial in $f''(g(\lambda)), \dots, f^{(k)}(g(\lambda)), g(\lambda), g'(\lambda), \dots, g^{(k-1)}(\lambda)$. Choosing $h_1(g; \lambda) = 0$ verifies the equality for $k = 1$, whereas for the inductive step we have

$$\begin{aligned} (f \circ g)^{(k+1)}(\lambda) &= f'(g(\lambda))g^{(k+1)}(\lambda) + f''(g(\lambda))g'(\lambda)g^{(k)}(\lambda) + h'_k(g; \lambda) \\ &=: f'(g(\lambda))g^{(k+1)}(\lambda) + h_{k+1}(g; \lambda), \end{aligned}$$

where $h_{k+1}(g; \lambda)$ is a polynomial in $f''(g(\lambda)), \dots, f^{(k+1)}(g(\lambda)), g(\lambda), \dots, g^{(k)}(\lambda)$.

Finally, we can prove that $(\hat{f}^{-1})^{(k)}(\lambda) = p^{(k)}(\lambda)$ for $k = 0, \dots, \ell - 1$. For $k = 0$, this holds by definition of \hat{f}^{-1} , while for $k < \ell - 1$, from $(f \circ p)^{(k+1)}(\lambda) = (f \circ \hat{f}^{-1})^{(k+1)}(\lambda)$ we have that $f'(p(\lambda))p^{(k+1)}(\lambda) + h_{k+1}(p; \lambda) = f'(\hat{f}^{-1}(\lambda))(\hat{f}^{-1})^{(k+1)}(\lambda) + h_{k+1}(\hat{f}^{-1}; \lambda)$. By the inductive hypothesis we have that $h_{k+1}(g; \lambda) = h_{k+1}(\hat{f}^{-1}; \lambda)$, and by recalling that $f'(\hat{f}^{-1}(\lambda)) = f'(p(\lambda)) \neq 0$, we can conclude that $p^{(k+1)}(\lambda) = (\hat{f}^{-1})^{(k+1)}(\lambda)$. \square

Another property of nonprimary solutions is that they are not isolated, as we show in the next section.

4.3.2 Isolated solutions

A solution X to $f(X) = A$ is *isolated* (in the topology induced by any matrix norm on $\mathbb{C}^{N \times N}$) if there exists a neighborhood \mathcal{U} of X where the matrix equation has a unique solution. We will characterize isolated solution in several ways, and will start by showing that nonprimary solutions are not isolated.

Theorem 4.5. *Let $A \in \mathbb{C}^{N \times N}$, and let $X \in \mathbb{C}^{N \times N}$ be a nonprimary solution to the matrix equation $f(X) = A$ where f is a complex function analytic at the spectrum of X . Then X*

is not isolated. Moreover, the set of solutions is unbounded and there are infinitely many solutions having the same spectrum as X .

Proof. In view of Theorem 4.3, if X is nonprimary, then necessarily either one of its critical eigenvalues, ξ say, is defective, or f takes the same value at two distinct eigenvalues $\xi_i \neq \xi_j$.

If ξ is defective, then there exists an invertible matrix M such that $M^{-1}XM = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}$, where $J_2 = J(\xi, k)$ is a Jordan block of size $k > 1$ associated with ξ . Using the notation of Corollary 4.2, we define the parametrized matrix

$$X(\delta) := M \begin{bmatrix} J_1 & 0 \\ 0 & J_2 + \delta e_1 e_k^T \end{bmatrix} M^{-1},$$

with $\delta \in \mathbb{C}$. Noticing that

$$f(X(\delta)) = M \begin{bmatrix} f(J_1) & 0 \\ 0 & f(J_2 + \delta e_1 e_k^T) \end{bmatrix} M^{-1} = M \begin{bmatrix} f(J_1) & 0 \\ 0 & f(J_2) \end{bmatrix} M^{-1} = f(X),$$

where the second equality follows from Corollary 4.2(b) with $f'(\xi) = 0$, shows that $X(\delta)$ is a solution to $f(X) = A$ for any δ , and since $\lim_{\delta \rightarrow 0} X(\delta) = X$, we conclude that X is not isolated.

If the matrix has two distinct eigenvalues $\xi_i \neq \xi_j$ such that $f(\xi_i) = f(\xi_j)$, the proof is similar, and it suffices to consider the block $J_2 = \begin{bmatrix} J(\xi_1, k_1) & 0 \\ 0 & J(\xi_2, k_2) \end{bmatrix}$, with $k_1, k_2 \geq 1$, and use Corollary 4.2(c) with $f[\xi_i, \xi_j] = 0$.

In both cases, for any $\delta \in \mathbb{C}$ the matrix $X(\delta)$ has the same spectrum as X by construction, and the set $\{X(\delta) : \delta \in \mathbb{C}\}$ is infinite and unbounded. \square

The converse of the Theorem 4.5 is not true. Indeed, the set of isolated solutions may be a strict subset of primary solutions. The next results provides several interesting characterizations of the isolated solutions of $f(X) = A$. As we will see, the algorithm we introduce in section 4.4 to solve (4.2), can compute a solution if and only if it is isolated.

Theorem 4.6. Let $f : \Omega \rightarrow \mathbb{C}$ be an analytic non-constant function on the domain $\Omega \subset \mathbb{C}$. Let $A \in \mathbb{C}^{N \times N}$, and let $X \in \mathbb{C}^{N \times N}$ be a solution to $f(X) = A$, with eigenvalues ξ_1, \dots, ξ_N in Ω . The following are equivalent:

- (a) X is isolated;
- (b) X is primary with simple or no critical eigenvalues, that is,
 1. for any two distinct eigenvalues ξ_i and ξ_j of X , we have $f(\xi_i) \neq f(\xi_j)$;
 2. all critical eigenvalues of X (if any) are simple;
- (c) X is the unique solution with eigenvalues ξ_1, \dots, ξ_N ;
- (d) $f[\xi_i, \xi_j] \neq 0$ for $i, j = 1, \dots, N$, with $i \neq j$.

Proof. (a) \Rightarrow (b). By Theorem 4.5, if X is isolated, then it is primary, and we need to prove only that all its critical eigenvalues are simple (we know that they are semisimple by Theorem 4.3). By contradiction, assume that ξ is a semisimple critical eigenvalue of X with multiplicity at least 2. Then there exists an invertible matrix M such that $M^{-1}XM = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix}$, where $J_2 = \xi I$, where I has size $\ell > 1$. With the notation of Corollary 4.2, the matrix

$$X(\delta) = M \begin{bmatrix} J_1 & 0 \\ 0 & J_2 + \delta e_1 e_\ell^T \end{bmatrix} M^{-1}$$

is a solution to $f(X) = A$ for any $\delta \in \mathbb{C}$, since

$$f(X(\delta)) = M \begin{bmatrix} f(J_1) & 0 \\ 0 & f(J_2 + \delta e_1 e_\ell^T) \end{bmatrix} M^{-1} = M \begin{bmatrix} f(J_1) & 0 \\ 0 & f(J_2) \end{bmatrix} M^{-1} = f(X)$$

where the second equality follows from Corollary 4.2(a) with $f'(\xi) = 0$. Since $\lim_{\delta \rightarrow 0} X(\delta) = X$, X is not isolated.

(b) \Rightarrow (c). The eigenvalues of A are the image under f of the eigenvalues of any solution, in particular, they are $f(\xi_1), \dots, f(\xi_N)$. Assume that X is primary with simple critical eigenvalues, and let Y be a solution with the same eigenvalues as X . This implies that Y has simple critical eigenvalues, and that $f(\xi_i) \neq f(\xi_j)$ for

any pair of distinct eigenvalues $\xi_i \neq \xi_j$. Therefore, by Theorem 4.3, Y must be primary; moreover, the images of these critical eigenvalues are simple eigenvalues of A as well. In particular, the defective eigenvalues of A (if any) are image of noncritical eigenvalues of X . By Proposition 4.4, $Y = \hat{f}^{-1}(A)$ and $X = f^{-1}(A)$, where both \hat{f}^{-1} and f^{-1} are inverses of f and are analytic at the images of noncritical eigenvalues of X (and Y), and thus are analytic at the defective eigenvalues of A . Since $\hat{f}^{-1}(\lambda) = f^{-1}(\lambda)$ for any eigenvalue λ of A , and the two functions are analytic and coincide in a neighborhood of λ if the eigenvalue is defective (the inverse of an analytic function is unique), we have that $Y = \hat{f}^{-1}(A) = f^{-1}(A) = X$.

(c) \Rightarrow (a). Without loss of generality, assume that if $\xi_i \neq \xi_k$ then $f(\xi_i) \neq f(\xi_k)$, since otherwise the solution X would be nonprimary and, by Theorem 4.5, there would be solutions other than X , but with the same spectrum as X .

Since the eigenvalues of A are $f(\xi_1), \dots, f(\xi_N)$, any solution to $f(X) = A$ has eigenvalues ζ_1, \dots, ζ_N such that $f(\zeta_1) = f(\xi_1), \dots, f(\zeta_N) = f(\xi_N)$. Let $\{\tau_j^{(i)}\}_{j \in \mathcal{J}_i}$ be the (possibly empty) set of solutions to $f(x) = f(\xi_i)$ other than ξ_i . If \mathcal{J}_i is empty for each i , then the eigenvalues of any solution must be ξ_1, \dots, ξ_N and X is the unique solution, hence it is isolated.

Let us now assume that some of the \mathcal{J}_i are nonempty. If \mathcal{J}_i is nonempty, then $\tau_j^{(i)} \neq \xi_k$ for each j and k : this is true by definition when $\xi_i = \xi_k$, and when $\xi_i \neq \xi_k$, by the assumption above, since $f(\tau_j^{(i)}) = f(\xi_i) \neq f(\xi_k)$. Moreover, since the zeros of a non-constant analytic function cannot have accumulation points in the domain of analyticity [4, sect. 143], ξ_k cannot be an accumulation point of the set $\{\tau_j^{(i)}\}_{j \in \mathcal{J}_i}$ and hence $\varepsilon_{i,k} := \inf_{j \in \mathcal{J}_i} |\tau_j^{(i)} - \xi_k|$ must be positive for each k . Set $\varepsilon := \min_{i: \mathcal{J}_i \neq \emptyset} \min_{k=1, \dots, N} \varepsilon_{i,k}$, and note that $\varepsilon > 0$.

A solution $Y \neq X$, must have at least one eigenvalue of the type $\hat{\tau} := \tau_j^{(i)}$ for some i and j . If that is the case, then $\min_{k=1, \dots, N} |\hat{\tau} - \xi_k| \geq \varepsilon$ or, in other words, at least one eigenvalue of Y has distance at least ε from any eigenvalue of X . On the other hand,

since the eigenvalues are continuous functions of the entries of a matrix, there exists a neighborhood \mathcal{U} of X , such that for any $Z \in \mathcal{U}$, we have

$$\max_{\eta \in \sigma(Z)} \min_{k=1, \dots, N} |\eta - \tilde{\zeta}_k| < \varepsilon/2,$$

where $\sigma(Z)$ is the spectrum of Z . Therefore Y does not belong to \mathcal{U} , and X is isolated.

(b) \Leftrightarrow (d). A necessary and sufficient condition for $f[\tilde{\zeta}_i, \tilde{\zeta}_j] = 0$ for $i \neq j$, is that either $\tilde{\zeta}_i \neq \tilde{\zeta}_j$, with $f(\tilde{\zeta}_i) = f(\tilde{\zeta}_j)$ or $\tilde{\zeta}_i = \tilde{\zeta}_j$ and $f'(\tilde{\zeta}_i) = 0$. These two conditions are equivalent to X being either nonprimary or primary with multiple critical eigenvalues. \square

We observe that, when a primary solution X of $f(X) = A$ is not isolated, the corresponding solution X is *ill-posed*, that is, a small perturbation of A may produce an equation that has no solutions near X .

By Theorem 4.6, a primary solution X that is not isolated has at least one semisimple eigenvalue $\tilde{\zeta}$ with multiplicity $k > 1$ and such that $f'(\tilde{\zeta}) = 0$. Hence $\lambda = f(\tilde{\zeta})$ is a semisimple eigenvalue of A , with the same multiplicity as $\tilde{\zeta}$ since X is primary. There exists a nonsingular matrix M such that $M^{-1}AM = \begin{bmatrix} J & 0 \\ 0 & \lambda I_k \end{bmatrix}$, where λ is not an eigenvalue of J . For $\varepsilon > 0$, the perturbed equation $f(X) = A(\varepsilon)$ where

$$A(\varepsilon) = M \operatorname{diag} \left(J, \begin{bmatrix} \lambda & \varepsilon & & \\ & \ddots & \ddots & \\ & & \lambda & \varepsilon \\ & & & \lambda \end{bmatrix} \right) M^{-1},$$

has no solutions with eigenvalue $\tilde{\zeta}$. Indeed, as primary matrix functions split Jordan blocks in presence of critical eigenvalues, if there exists $X(\varepsilon)$ such that $f(X(\varepsilon)) = A(\varepsilon)$, it must have an eigenvalue μ , such that $f(\mu) = \lambda$ and $f'(\mu) \neq 0$, which in turn implies that $\mu \neq \tilde{\zeta}$. Therefore, the solution $X(\varepsilon)$ can be ruled out from a sufficiently small neighborhood of X .

4.3.3 Critical solutions

Let f be an analytic complex function and let $Df(M) : \mathbb{C}^{N \times N} \rightarrow \mathbb{C}^{N \times N}$ be the Fréchet derivative of f at the matrix $M \in \mathbb{C}^{N \times N}$. A solution X to the equation $f(X) = A$ is said to be *critical* if $Df(X)$ is singular, and *noncritical* otherwise. We may easily characterize critical solutions.

Proposition 4.7. *Let f be a complex function, let $A \in \mathbb{C}^{N \times N}$, and let $X \in \mathbb{C}^{N \times N}$ be a solution to the matrix equation $f(X) = A$. If f is differentiable at X , then the derivative $Df(X)$ is nonsingular if and only if the following two conditions are fulfilled:*

1. *for any two distinct eigenvalues ξ_i and ξ_j of X , we have $f(\xi_i) \neq f(\xi_j)$;*
2. *none of the eigenvalues of X is critical for f .*

Moreover, these conditions are equivalent to requiring that $f[\xi_i, \xi_j] \neq 0$, for $i, j = 1, \dots, N$, where ξ_1, \dots, ξ_N are the eigenvalues of X .

Proof. Observe that the two conditions hold if and only if the divided differences of any two eigenvalues of X is not zero. Since the the eigenvalues of $Df(X)$ are the divided differences of eigenvalues of X [14, Thm. 3.9], this is equivalent to requiring that $Df(X)$ is nonsingular. \square

A further property of nonprimary solutions is that of being critical.

Proposition 4.8. *Let f be an analytic complex function, let $A \in \mathbb{C}^{N \times N}$, and let $X \in \mathbb{C}^{N \times N}$ be a nonprimary solution to the matrix equation $f(X) = A$. Then $Df(X)$ is singular.*

Proof. In view of Theorem 4.3, if X is not primary, then X has either two distinct eigenvalues ξ_i and ξ_j such that $f(\xi_i) = f(\xi_j)$ and thus $f[\xi_i, \xi_j] = 0$, or a defective eigenvalue ξ such that $f[\xi, \xi] = f'(\xi) = 0$. Since the eigenvalues of $Df(X)$ are the divided differences of two eigenvalues of X [14, Thm. 3.9], both cases yield a singular derivative. \square

4.4 A SUBSTITUTION ALGORITHM

Given $A \in \mathbb{C}^{N \times N}$, we want to find the primary solutions $X \in \mathbb{C}^{N \times N}$ to (4.2). To this end, we first reduce this equation to

$$p(X) = Aq(X), \quad (4.7)$$

then consider a (block) triangular form of A , such as the Schur form, and devise an algorithm to compute the entries of X . We begin by showing that (4.2) and (4.7) are equivalent.

In the scalar case, if p and q are coprime, then a root of p cannot be a root of q and vice versa, and thus the scalar equation $\frac{p(x)}{q(x)} = a$ has a solution if and only if $p(x) = aq(x)$ does. The matrix version of this implication is also true, as the following result shows.

Proposition 4.9. *Let $p \in \mathbb{C}_m[z], q \in \mathbb{C}_n[z]$ be coprime. Then $X \in \mathbb{C}^{N \times N}$ is a solution to $p(X)q(X)^{-1} = A$ if and only if it satisfies $p(X) = Aq(X)$.*

Proof. If X is such that $p(X)q(X)^{-1} = A$, then $p(X) = Aq(X)$. For the other implication, first note that if X is such that $p(X) = Aq(X)$ and $q(X)$ is nonsingular, then $p(X)q(X)^{-1} = A$, hence it is enough to show that $q(X)$ is nonsingular.

For the sake of contradiction, assume that $q(X)$ is singular. Then there exists a nonzero vector $b \in \mathbb{C}^N$, such that $q(X)b = 0$, and thus $p(X)b = Aq(X)b = 0$. Since the set $\mathcal{I} = \{s \in \mathbb{C}[z] : s(X)b = 0\}$ is an ideal in a principal ideal domain, it is generated by a minimal polynomial $s(x) \in \mathbb{C}[z]$, that is not constant since $b \neq 0$ and thus $\mathcal{I} \neq \mathbb{C}[z]$. Hence, $s(x)|q(x)$ and $s(x)|p(x)$, which leads to a contradiction since $p(x)$ and $q(x)$ are coprime. \square

Let us consider a similarity transformation that reduces A to a block upper triangular matrix $U^{-1}AU =: T = [T_{ij}]_{i,j=1,\dots,\nu} \in \mathbb{C}^{N \times N}$, where $T_{ij} \in \mathbb{C}^{\tau_i \times \tau_j}$, with $\sum_{i=1}^{\nu} \tau_i = N$ and $T_{ij} = 0$ for $i > j$.

We are mostly interested in the Schur decomposition, where U is unitary and T is upper triangular, and, for $A \in \mathbb{R}^{N \times N}$, in the the real Schur decomposition, where U

is real orthogonal and T is upper quasi-triangular. Nevertheless, we prefer to work in greater generality, as a different blocking strategy may allow for more efficient implementations of the algorithms (for instance, in order to exploit caching and parallelism in modern computer architectures).

Since matrix polynomials commute with similarities, X is a solution to (4.2) if and only if $Y := U^{-1}XU$ satisfies $r(Y) = T$, and in view of Proposition 4.9, in order to solve (4.2) we can work with the simpler matrix equation $p(Y) = Tq(Y)$. By exploiting Horner's scheme for polynomial evaluation [10, Alg. 9.2.1], we can rewrite the latter equation as $P^{[0]} = TQ^{[0]}$, where $P^{[0]} = p(Y)$ and $Q^{[0]} = q(Y)$, are defined recursively by

$$\begin{aligned}
 P^{[m]} &= c_m I, & Q^{[n]} &= d_n I. \\
 P^{[m-1]} &= c_{m-1} I + Y P^{[m]}, & Q^{[n-1]} &= d_{n-1} I + Y Q^{[n]}, \\
 &\vdots & &\vdots \\
 P^{[1]} &= c_1 I + Y P^{[2]}, & Q^{[1]} &= d_1 I + Y Q^{[2]}, \\
 P^{[0]} &= c_0 I + Y P^{[1]}, & Q^{[0]} &= d_0 I + Y Q^{[1]},
 \end{aligned} \tag{4.8}$$

If we look for primary solutions only, we may assume that Y is block upper triangular with the same block structure as T , which implies in turn that all $P^{[u]}$ s and $Q^{[v]}$ s have the same block upper triangular structure. We adopt the following notation: for a matrix M with the same block structure as T , we denote by M_{ij} the block in position (i, j) of M .

We assume that the ν blocks along the diagonal of Y are known, for instance they can be deduced by a direct formula when the size is 1 or 2. Note that in most cases the diagonal blocks can be chosen in several ways, and that this choice determines what solution the algorithm will compute among all those that are primary. We discuss these points in details in the next section.

The blocks along the diagonal of the matrices $P^{[u]}$ and $Q^{[v]}$, for $u = 0, \dots, m-1$ and $v = 0, \dots, n-1$, can be uniquely determined by means of (4.8), and in order to

compute the blocks in the upper triangular part of Y , $P^{[u]}$ and $Q^{[v]}$, note that for $1 \leq i < j \leq \nu$, we have

$$\begin{aligned} P_{ij}^{[u]} &= \sum_{k=i}^j Y_{ik} P_{kj}^{[u+1]} = Y_{ii} P_{ij}^{[u+1]} + Y_{ij} P_{jj}^{[u+1]} + \sum_{k=i+1}^{j-1} Y_{ik} P_{kj}^{[u+1]}, & u = 0, \dots, m-1, \\ Q_{ij}^{[v]} &= \sum_{k=i}^j Y_{ik} Q_{kj}^{[v+1]} = Y_{ii} Q_{ij}^{[v+1]} + Y_{ij} Q_{jj}^{[v+1]} + \sum_{k=i+1}^{j-1} Y_{ik} Q_{kj}^{[v+1]}, & v = 0, \dots, n-1. \end{aligned} \quad (4.9)$$

By substituting (4.9) for $P_{ij}^{[u+1]}$ and $Q_{ij}^{[v+1]}$ into those for $P_{ij}^{[u]}$ and $Q_{ij}^{[v]}$, respectively, and recursively repeating this procedure, we get, as shown in the following proposition, an expression where Y_{ij} appears together with blocks of Y , $P^{[u]}$, and $Q^{[v]}$ lying to the left of the block in position (i, j) or below it. This discussion translates immediately into a two-step algorithm for computing Y : first compute the diagonal blocks and then compute the off-diagonal blocks a superdiagonal at a time.

Proposition 4.10. *Let $p \in \mathbb{C}_m[z]$ and $q \in \mathbb{C}_n[z]$ be coprime, let $T \in \mathbb{C}^{N \times N}$ be block upper triangular, let $Y \in \mathbb{C}^{N \times N}$ be a solution to the matrix equation $p(Y) = Tq(Y)$ with the same block structure as T , and let $P^{[u]}, Q^{[v]} \in \mathbb{C}^{N \times N}$, for $u = 0, \dots, m$ and $v = 0, \dots, n$, be as in (4.8). Then $P^{[u]}$ and $Q^{[v]}$ have the same block structure as T , and their off-diagonal blocks, for $1 \leq i < j \leq \nu$, are given by the formulae*

$$\begin{aligned} P_{ij}^{[u]} &= \sum_{e=1}^{m-u} Y_{ii}^{e-1} Y_{ij} P_{jj}^{[u+e]} + \sum_{f=1}^{m-u-1} Y_{ii}^{f-1} C_{ij}^{[u+f]}, & u = 0, \dots, m-1, \\ Q_{ij}^{[v]} &= \sum_{g=1}^{n-v} Y_{ii}^{g-1} Y_{ij} Q_{jj}^{[v+g]} + \sum_{h=1}^{n-v-1} Y_{ii}^{h-1} D_{ij}^{[v+h]}, & v = 0, \dots, n-1, \end{aligned} \quad (4.10)$$

where

$$C_{ij}^{[u]} = \sum_{k=i+1}^{j-1} Y_{ik} P_{kj}^{[u]}, \quad D_{ij}^{[v]} = \sum_{k=i+1}^{j-1} Y_{ik} Q_{kj}^{[v]}.$$

Moreover, one has the following

$$\begin{aligned} \sum_{e=1}^m Y_{ii}^{e-1} Y_{ij} P_{jj}^{[e]} - T_{ii} \sum_{g=1}^n Y_{ii}^{g-1} Y_{ij} Q_{jj}^{[g]} \\ = \sum_{k=i+1}^j T_{ik} Q_{kj}^{[0]} - \sum_{f=1}^{m-1} Y_{ii}^{f-1} C_{ij}^{[f]} + T_{ii} \sum_{h=1}^{n-1} Y_{ii}^{h-1} D_{ij}^{[h]}. \end{aligned} \quad (4.11)$$

Proof. The two claims in (4.10) can be proved by induction on an auxiliary variable k . We limit ourselves to the recurrence for $P^{[u]}$, the proof for $Q^{[v]}$ being analogous. For $u = m - 1$, equation (4.10) reduces to $P_{ij}^{[m-1]} = c_m Y_{ij}$, which follows directly from the definition of $P^{[m-1]}$ in (4.8). For the inductive step, we have, for $1 < k \leq m$,

$$\begin{aligned} P_{ij}^{[m-k]} &= Y_{ii} P_{ij}^{[m-k+1]} + Y_{ij} P_{jj}^{[m-k+1]} + \sum_{k=i+1}^{j-1} Y_{ik} P_{kj}^{[m-k+1]} \\ &= \sum_{e=1}^k Y_{ii}^{e-1} Y_{ij} P_{jj}^{[m-k+e]} + \sum_{f=1}^{k-1} Y_{ii}^{f-1} C_{ij}^{[m-k+f]}. \end{aligned}$$

In order to establish (4.11), note that one can rewrite $P^{[0]} = TQ^{[0]}$ as

$$P_{ij}^{[0]} - T_{ii} Q_{ij}^{[0]} = \sum_{k=i+1}^j T_{ik} Q_{kj}^{[0]}.$$

Substituting (4.10) for $P_{ij}^{[0]}$ and $Q_{ij}^{[0]}$ and simplifying concludes the proof. \square

4.4.1 Complex Schur form

When $T \in \mathbb{C}^{N \times N}$ is upper triangular, the blocks along the diagonal of T are of size 1×1 and $\nu = N$. Equation (4.11) involves just scalars and can be written as $\psi_{ij} y_{ij} = \varphi_{ij}$, where

$$\psi_{ij} := \sum_{e=1}^m y_{ii}^{e-1} p_{jj}^{[e]} - t_{ii} \sum_{g=1}^n y_{ii}^{g-1} q_{jj}^{[g]}, \quad (4.12)$$

and

$$\varphi_{ij} := \sum_{k=i+1}^j t_{ik} q_{kj}^{[0]} - \sum_{f=1}^{m-1} y_{ii}^{f-1} C_{ij}^{[f]} + t_{ii} \sum_{h=1}^{n-1} y_{ii}^{h-1} D_{ij}^{[h]}. \quad (4.13)$$

If t_{ii} is a diagonal element of T , then for $i = 1, \dots, N$, y_{ii} will be any of the at most $\max(m, n)$ distinct roots of the polynomial $p(x) - t_{ii}q(x) = 0$. In order to compute the off-diagonal elements of Y , we can see the relation $\psi_{ij}y_{ij} = \varphi_{ij}$ as an equation

$$\psi_{ij}x = \varphi_{ij}, \quad (4.14)$$

whose unique solution is y_{ij} when $\psi_{ij} \neq 0$ and the values y_{hk} with $h - k < i - j$ are known quantities.

We give necessary and sufficient conditions for (4.14) to have unique solution, and relate them to the characterization of isolated solutions given in section 4.3. We start with a couple of technical lemmas, then we give the main theorem.

Lemma 4.11. *Let $p(x) = \sum_{i=0}^m c_i x^i$, let $a, b \in \mathbb{C}$ and let $p^{[k]}(x) = \sum_{i=0}^{m-k} c_{k+i} x^i$, for $k = 0, \dots, m$, be the sequence of stages of Horner's rule applied to p . Then*

$$\chi := \sum_{k=1}^m a^{k-1} p^{[k]}(b) = p[a, b]. \quad (4.15)$$

Proof. By definition of $p[a, b]$, we have to prove that if $a = b$ then $\chi = p'(a)$, whereas if $a \neq b$ $\chi = \frac{p(a) - p(b)}{a - b}$. In both cases we have

$$\sum_{k=1}^m a^{k-1} p^{[k]}(b) = \sum_{k=1}^m a^{k-1} \left(\sum_{i=0}^{m-k} c_{k+i} b^i \right) = \sum_{\ell=1}^m c_{\ell} \left(\sum_{k=0}^{\ell-1} a^k b^{\ell-k-1} \right).$$

If $a = b$, then we get

$$\sum_{k=1}^m a^{k-1} p^{[k]}(b) = \sum_{\ell=1}^m c_{\ell} \ell a^{\ell-1} = p'(a),$$

whereas, for $a \neq b$ we have

$$\sum_{k=1}^m a^{k-1} p^{[k]}(b) = \sum_{\ell=1}^m c_{\ell} \frac{a^{\ell} - b^{\ell}}{a - b} = \frac{1}{a - b} \left(\sum_{\ell=0}^m c_{\ell} a^{\ell} - \sum_{\ell=0}^m c_{\ell} b^{\ell} \right) = \frac{p(a) - p(b)}{a - b},$$

which concludes the proof. \square

Lemma 4.12. *Let $p(x) = \sum_{i=0}^m c_i x^i$ and $q(x) = \sum_{j=0}^n d_j x^j$, let $r = p/q$, and let $a, b \in \mathbb{C}$ be such that $q(a) \neq 0$ and $q(b) \neq 0$. Then*

$$\psi := \sum_{i=1}^m a^{i-1} p^{[i]}(b) - r(a) \sum_{j=1}^n a^{j-1} q^{[j]}(b) \neq 0$$

if and only if either $a \neq b$ and $r(a) \neq r(b)$ or $a = b$ and $r'(a) \neq 0$.

Proof. By Lemma 4.11, when $a \neq b$ we have that

$$\psi = \frac{p(a) - p(b) - r(a)(q(a) - q(b))}{a - b}, \quad (4.16)$$

which is nonzero if and only if

$$p(a) - p(b) - \frac{p(a)}{q(a)}(q(a) - q(b)) \neq 0, \quad (4.17)$$

or equivalently

$$r(a) \neq r(b).$$

On the other hand, if $a = b$, then

$$\psi = p'(a) - r(a)q'(a) = \frac{p'(a)q(a) - p(a)q'(a)}{q(a)} = r'(a)q(a), \quad (4.18)$$

which is nonzero if and only if $r'(a) \neq 0$. \square

Theorem 4.13. *Let $T \in \mathbb{C}^{N \times N}$ be upper triangular, let $p, q, Y, P^{[u]}$, for $u = 0, \dots, m$, and $Q^{[v]}$, for $v = 0, \dots, n$, be as in Proposition 4.10, and let $r(x) = p(x)q(x)^{-1}$. Then equation (4.14) has a unique solution y_{ij} for all $1 \leq i < j \leq N$ if and only if $r[y_{ii}, y_{jj}]q(y_{jj}) \neq 0$.*

Proof. It is enough to show that for ψ_{ij} in (4.12), we have that $\psi_{ij} = r[y_{ii}, y_{jj}]q(y_{jj})$. If $y_{ii} = y_{jj}$, then the proof is the same as in (4.18). When $y_{ii} \neq y_{jj}$, by using (4.16), we get that

$$\psi_{ij} = \frac{-p(y_{jj}) + p(y_{ii})q(y_{jj})/q(y_{ii})}{y_{ii} - y_{jj}} = \frac{r(y_{ii}) - r(y_{jj})}{y_{ii} - y_{jj}}q(y_{jj}) = r[y_{ii}, y_{jj}]q(y_{jj}),$$

as required. \square

Corollary 4.14 (Applicability of the Schur algorithm for isolated solutions). *Let $r = p/q$ be a rational function, with $p \in \mathbb{C}_m[z]$ and $q \in \mathbb{C}_n[z]$ coprime, and let $Y \in \mathbb{C}^{N \times N}$ be a solution to $r(Y) = T$, with $T \in \mathbb{C}^{N \times N}$ upper triangular. Let $P^{[u]}$ for $u = 0, \dots, m$, and $Q^{[v]}$ for $v = 0, \dots, n$, be as in (4.9). Then the following two conditions are equivalent:*

- (a) *Y is an isolated solution;*
- (b) *the Schur algorithm is applicable and computes Y , if we choose y_{ii} as solution of the equation $p(x) - t_{ii}q(x) = 0$, for $i = 1, \dots, N$, that is, equation (4.14) has y_{ij} as unique solution, for $1 \leq i < j \leq N$.*

Proof. By Theorem 4.13, (4.14) has unique solution if and only if $r[y_{ii}, y_{jj}]q(y_{jj}) \neq 0$, for $1 \leq i < j \leq N$. Proposition 4.9 ensures that $q(y_{jj}) \neq 0$, for $j = 1, \dots, N$, since $q(Y)$ is nonsingular (recall that the eigenvalues of $q(Y)$ are $q(y_{11}), \dots, q(y_{NN})$). Thus, equation (4.14) has a unique solution if and only if $r[y_{ii}, y_{jj}] \neq 0$ for $1 \leq i < j \leq N$, which in turn, by the symmetry of divided differences, is equivalent Theorem 4.6(d), that is equivalent to requiring that Y is isolated. \square

These results show that if we focus on a primary solution with simple critical eigenvalues, then we can compute the solution to the triangular equation $r(Y) = T$, by first computing the diagonal elements of Y , taking care of choosing the same branch for the same eigenvalue of T , and then computing the elements y_{ij} , for $i < j$, by means of (4.14), one superdiagonal at a time. This is the basis of Algorithm 4.1, which we call the Schur algorithm.

Algorithm 4.1: Schur algorithm for rational matrix equations.

Input : $A \in \mathbb{C}^{N \times N}$, $c \in \mathbb{C}^{m+1}$ coefficients of p , $d \in \mathbb{C}^{n+1}$ coefficients of q .
Output: $X \in \mathbb{C}^{N \times N}$ such that $p(X)q^{-1}(X) \approx A$.

- 1 Compute the complex Schur decomposition $A := UTU^*$.
- 2 **for** $i = 1$ **to** N **do**
- 3 $y_{ii} \leftarrow$ a solution to $p(x) - t_{ii}q(x) = 0$
- 4 $p_{ii}^{[m-1]} \leftarrow c_{m-1} + c_m y_{ii}$
- 5 **for** $u = m - 2$ **down to** 0 **do**
- 6 $p_{ii}^{[u]} \leftarrow c_u + y_{ii} p_{ii}^{[u+1]}$
- 7 $q_{ii}^{[n-1]} \leftarrow d_{n-1} + d_n y_{ii}$
- 8 **for** $v = n - 2$ **down to** 0 **do**
- 9 $q_{ii}^{[v]} \leftarrow d_v + y_{ii} q_{ii}^{[v+1]}$
- 10 **for** $\ell = 1$ **to** $N - 1$ **do**
- 11 **for** $i = 1$ **to** $N - \ell$ **do**
- 12 $j \leftarrow i + \ell$
- 13 **for** $f = 1$ **to** $m - 1$ **do**
- 14 $C_{ij}^{[f]} = \sum_{k=i+1}^{j-1} y_{ik} p_{kj}^{[f]}$
- 15 **for** $h = 1$ **to** $n - 1$ **do**
- 16 $D_{ij}^{[h]} = \sum_{k=i+1}^{j-1} y_{ik} q_{kj}^{[h]}$
- 17 $rhs \leftarrow \sum_{k=i+1}^j t_{ik} q_{kj}^{[0]} - \sum_{f=1}^{m-1} y_{ii}^{f-1} C_{ij}^{[f]} + t_{ii} \sum_{h=1}^{n-1} y_{ii}^{h-1} D_{ij}^{[h]}$
- 18 $lhs \leftarrow \sum_{e=1}^m y_{ii}^{e-1} p_{jj}^{[e]} - t_{ii} \sum_{g=1}^n y_{ii}^{g-1} q_{jj}^{[g]}$
- 19 $y_{ij} \leftarrow rhs/lhs$
- 20 $p_{ij}^{[m-1]} \leftarrow c_m y_{ij}$
- 21 **for** $u = m - 2$ **down to** 1 **do**
- 22 $p_{ij}^{[u]} \leftarrow y_{ii} p_{ij}^{[u+1]} + y_{ij} p_{jj}^{[u+1]} + C_{ij}^{[u+1]}$
- 23 $q_{ij}^{[n-1]} \leftarrow d_n y_{ij}$
- 24 **for** $v = n - 2$ **down to** 0 **do**
- 25 $q_{ij}^{[v]} \leftarrow y_{ii} q_{ij}^{[v+1]} + y_{ij} q_{jj}^{[v+1]} + D_{ij}^{[v+1]}$
- 26 $X \leftarrow UYU^*$

We now discuss the cost of the algorithm. Computing the Schur decomposition of a square matrix of size N and recovering the result require $25N^3$ and $3N^3$ flops, respectively. The for loop at line 2 requires $O((m+n)N)$ flops, those on line 13 and 15 require $(m-1)N^3/3$ and $(n-1)N^3/3$, respectively, and evaluating the expression on line 17 requires $N^3/3$ flops. All the other operations within the loop on line 10 require $O((m+n)N^2)$. Therefore the asymptotic cost of the algorithm is $(28 + \frac{m+n-1}{3})N^3$.

Remark. Corollary 4.14 shows that our algorithm cannot compute primary solutions with semisimple critical eigenvalues with multiplicity greater than one. We now describe how the algorithm can be modified in order to compute these ill-posed solutions.

Let Y be a primary solution to $r(Y) = T$ and let ξ_1, \dots, ξ_s , with $s > 0$, be its critical, and thus semisimple, eigenvalues with multiplicities ν_1, \dots, ν_s , greater than one. We have that $\lambda_\ell = r(\xi_\ell)$, for $\ell = 1, \dots, s$, is a semisimple eigenvalue of T with the same multiplicity as ξ_ℓ (the multiplicity cannot be larger since Y is primary).

Using the procedure described in [2], it is possible to reorder the matrix T so that, for $\ell = 1, \dots, s$, the occurrences of λ_ℓ are adjacent along the diagonal of T . By doing so, we get a new matrix $\tilde{T} = Q^*TQ$, where Q is the unitary matrix that performs the reordering. Since λ_ℓ is semisimple, the diagonal block of \tilde{T} corresponding to λ_ℓ is $\lambda_\ell I$, and we get

$$\tilde{T} = \begin{bmatrix} \tilde{T}_{11} & * & \cdots & * \\ & \lambda_1 I & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda_s I \end{bmatrix},$$

where the asterisks represent possibly nonzero blocks and \tilde{T}_{11} is a triangular block collecting all the eigenvalues other than $\lambda_1, \dots, \lambda_s$.

Any solution \tilde{Y} to $r(\tilde{Y}) = \tilde{T}$ yields the solution $Y = Q\tilde{Y}Q^*$ of $r(Y) = T$, with the same eigenvalues. Moreover, since \tilde{Y} is a primary function of \tilde{T} , it has the structure

$$\tilde{Y} = \begin{bmatrix} \tilde{Y}_{11} & * & \cdots & * \\ & \zeta_1 I & \ddots & \vdots \\ & & \ddots & * \\ & & & \zeta_s I \end{bmatrix},$$

where, $\zeta_i \neq \zeta_j$ for $i \neq j$, and \tilde{Y}_{11} collects all the eigenvalues not in the set $\{\zeta_1, \dots, \zeta_s\}$.

This implies that $\tilde{y}_{ij} = 0$ when $\tilde{y}_{ii} = \tilde{y}_{jj} = \zeta_\ell$ for some ℓ , and thus we can determine \tilde{y}_{ij} , without solving (4.11), while (4.11) can be used for all other entries of the upper triangular part of \tilde{Y} , for which the solution is unique. Therefore, in principle, any primary solution could be computed using (a variation) of Algorithm 4.1, but in practice, the problem is ill-posed and we focus our attention on solutions with simple critical eigenvalues.

4.4.2 Real Schur form

When $A \in \mathbb{R}^{N \times N}$ and one is interested in real solutions to (4.2), in order to use real arithmetic only, we consider the real Schur decomposition $A := UTU^T$, where $U \in \mathbb{R}^{N \times N}$ is orthogonal, and $T \in \mathbb{R}^{N \times N}$ is upper quasi-triangular and has $\nu \leq N$ diagonal blocks of size either 1×1 or 2×2 . In the former case, the diagonal block Y_{ii} can be computed as discussed in the previous section. Otherwise, we can rely on the following result.

Proposition 4.15. *Let $M \in \mathbb{R}^{2 \times 2}$, and let $V \in \mathbb{R}^{2 \times 2}$ be such that $V^{-1}MV = \text{diag}(\mu, \bar{\mu})$, for some $\mu = a + ib$, with $b \neq 0$. Let $f : \{\mu, \bar{\mu}\} \rightarrow \mathbb{C}$ be a function such that $f(\bar{\mu}) = \overline{f(\mu)}$, and let $f(\mu) = c + id$. Then*

$$f(M) = \frac{d}{b}M + \left(c - \frac{ad}{b}\right)I. \quad (4.19)$$

Algorithm 4.2: Real Schur algorithm for rational matrix equations.

Input : $A \in \mathbb{C}^{N \times N}$, $c \in \mathbb{C}^{m+1}$ coefficients of p , $d \in \mathbb{C}^{n+1}$ coefficients of q .
Output: $X \in \mathbb{C}^{N \times N}$ such that $p(X)q^{-1}(X) \approx A$.

- 1 Compute the real Schur decomposition $A := UTU^*$.
- 2 **for** $i = 1$ **to** v **do**
- 3 $Y_{ii} \leftarrow$ a solution to $p(X) - T_{ii}q(X) = 0$
- 4 $P_{ii}^{[m-1]} \leftarrow c_{m-1}I_{\tau_i} + c_m Y_{ii}$
- 5 **for** $u = m - 2$ **down to** 0 **do**
- 6 $P_{ii}^{[u]} \leftarrow c_u I_{\tau_i} + Y_{ii} P_{ii}^{[u+1]}$
- 7 $Q_{ii}^{[n-1]} \leftarrow d_{n-1}I_{\tau_i} + d_n Y_{ii}$
- 8 **for** $v = n - 2$ **down to** 0 **do**
- 9 $Q_{ii}^{[v]} \leftarrow d_v I_{\tau_i} + Y_{ii} Q_{ii}^{[v+1]}$
- 10 **for** $\ell = 1$ **to** $N - 1$ **do**
- 11 **for** $i = 1$ **to** $N - \ell$ **do**
- 12 $j \leftarrow i + \ell$
- 13 **for** $f = 1$ **to** $m - 1$ **do**
- 14 $C_{ij}^{[f]} = \sum_{k=i+1}^{j-1} Y_{ik} P_{kj}^{[f]}$
- 15 **for** $h = 1$ **to** $n - 1$ **do**
- 16 $D_{ij}^{[h]} = \sum_{k=i+1}^{j-1} Y_{ik} Q_{kj}^{[h]}$
- 17 $s_{ij} \leftarrow \text{vec} \left(\sum_{k=i+1}^j T_{ik} Q_{kj}^{[0]} - \sum_{f=1}^{m-1} Y_{ii}^{f-1} C_{ij}^{[f]} + T_{ii} \sum_{h=1}^{n-1} Y_{ii}^{h-1} D_{ij}^{[h]} \right)$
- 18 $M_{ij} \leftarrow \sum_{e=1}^m (P_{jj}^{[e]})^T \otimes Y_{ii}^{e-1} - \sum_{g=1}^n (Q_{jj}^{[g]})^T \otimes (T_{ii} Y_{ii}^{g-1})$
- 19 $\text{vec}(Y_{ij}) \leftarrow M_{ij}^{-1} s_{ij}$
- 20 $P_{ij}^{[m-1]} \leftarrow c_m Y_{ij}$
- 21 **for** $u = m - 2$ **down to** 1 **do**
- 22 $P_{ij}^{[u]} \leftarrow Y_{ii} P_{ij}^{[u+1]} + Y_{ij} P_{jj}^{[u+1]} + C_{ij}^{[u+1]}$
- 23 $Q_{ij}^{[n-1]} \leftarrow d_n Y_{ij}$
- 24 **for** $v = n - 2$ **down to** 0 **do**
- 25 $Q_{ij}^{[v]} \leftarrow Y_{ii} Q_{ij}^{[v+1]} + Y_{ij} Q_{jj}^{[v+1]} + D_{ij}^{[v+1]}$
- 26 $X \leftarrow UYU^T$

Proof. It is well known [14, Thm. 1.12] that $f(M)$ coincides with the interpolating polynomial of f at the eigenvalues of M , that is

$$p(x) = f(\mu) \frac{x - \bar{\mu}}{\mu - \bar{\mu}} + f(\bar{\mu}) \frac{x - \mu}{\bar{\mu} - \mu} = \frac{f(\mu) - f(\bar{\mu})}{\mu - \bar{\mu}} x + \frac{\mu f(\bar{\mu}) - \bar{\mu} f(\mu)}{\mu - \bar{\mu}}. \quad (4.20)$$

By replacing the definitions of μ and $f(\mu)$ and simplifying, one obtains (4.19). \square

In order to compute the off-diagonal blocks of Y , we need to solve for the block Y_{ij} the matrix equation (4.11), which, by using the vec operator, can be rewritten as the linear system

$$M_{ij} \text{vec}(Y_{ij}) = \text{vec} \left(\sum_{k=i+1}^j T_{ik} Q_{kj}^{[0]} - \sum_{f=1}^{m-1} Y_{ii}^{f-1} C_{ij}^{[f]} + T_{ii} \sum_{h=1}^{n-1} Y_{ii}^{h-1} D_{ij}^{[h]} \right),$$

where the coefficient matrix

$$M_{ij} = \sum_{e=1}^m \left(P_{jj}^{[e]} \right)^T \otimes Y_{ii}^{e-1} - \sum_{g=1}^n \left(Q_{jj}^{[g]} \right)^T \otimes (T_{ii} Y_{ii}^{g-1}) \quad (4.21)$$

can be of size 1, 2, or 4, depending on the size of the blocks Y_{ii} and Y_{jj} . In the following, we give necessary and sufficient conditions for M to be nonsingular.

Theorem 4.16. *Let $T \in \mathbb{C}^{N \times N}$ be upper quasi-triangular, let $p, q, Y, P^{[u]}$, for $u = 0, \dots, m$, and $Q^{[v]}$, for $v = 0, \dots, n$, be as in Proposition 4.10, and let $r(x) = p(x)q(x)^{-1}$. Then M_{ij} in (4.21) is nonsingular for all $1 \leq i < j \leq v$ if and only if Y is a primary solution to (4.2) with simple critical eigenvalues (if any).*

Proof. Let (ξ_i, u_i) be an eigenpair of Y_{ii} and let (ξ_j, u_j) be an eigenpair of Y_{jj} . Then by using the properties of the Kronecker product, we observe that

$$\begin{aligned} M_{ij}(u_j \otimes u_i) &= \left(\sum_{e=1}^m \left(P_{jj}^{[e]} \right)^T \otimes Y_{ii}^{e-1} - \sum_{g=1}^n \left(Q_{jj}^{[g]} \right)^T \otimes (T_{ii} Y_{ii}^{g-1}) \right) (u_j \otimes u_i) \\ &= \left(\sum_{e=1}^m p^{[e]}(\xi_j) \xi_i^{e-1} - r(\xi_i) \sum_{g=1}^n q^{[g]}(\xi_j) \xi_i^{g-1} \right) (u_j \otimes u_i) \\ &=: \zeta(u_j \otimes u_i), \end{aligned}$$

and conclude that $(\zeta, u_j \otimes u_i)$ is an eigenpair of M_{ij} . Since the eigenpairs of Y_{ii} and Y_{jj} are chosen arbitrarily and everything is diagonalizable, all the eigenvalues of M_{ij} have this form, and we can conclude that the matrix M_{ij} is nonsingular if and only if $\zeta \neq 0$, which is guaranteed by Lemma 4.12, since Y is a primary solution to (4.2) with simple or no critical eigenvalues.

Conversely, let ξ_i, ξ_j be eigenvalues of different diagonal blocks of Y , Y_{ii} and Y_{jj} say, then there exist (ξ_i, u_i) and (ξ_j, v_j) eigenpairs of Y_{ii} and Y_{jj} , respectively. Since M_{ij} is nonsingular, its eigenvalue $\sum_{e=1}^m p^{[e]}(\xi_j) \xi_i^{e-1} - r(\xi_i) \sum_{g=1}^n q^{[g]}(\xi_j) \xi_i^{g-1}$ is nonzero, thus by Lemma 4.12 either $\xi_i \neq \xi_j$ and $r(\xi_i) \neq r(\xi_j)$ or $\xi_i = \xi_j$ and $r'(\xi_i) \neq 0$. If ξ_i and ξ_j belong to the same block, then either the block is of size 1×1 or ξ_i is the complex conjugate of ξ_j , and again, $\xi_i \neq \xi_j$ and $r(\xi_i) \neq r(\xi_j)$. Since ξ_i and ξ_j were chosen arbitrarily, the same relation is true for any chosen pair of eigenvalues, and Y is thus a primary solution to (4.2). \square

4.5 NUMERICAL EXPERIMENTS

To the best of our knowledge, no other algorithm exists for the solution of the general matrix equation $r(X) = A$, thus we compare our approach with well-established techniques for the computation of primary matrix functions. We consider the (approximate) diagonalization method [5] and the Schur–Parlett algorithm [6, 20], applied to the function $r^{-1}(z)$, that is, the chosen inverse of $r(z)$ in a neighborhood of the eigenvalues of A .

If A is a normal, then its Schur form $T = U^*AU = \text{diag}(\lambda_1, \dots, \lambda_N)$ is diagonal, and the solution to $r(X) = A$ is $X = U \text{diag}(r^{-1}(\lambda_1), \dots, r^{-1}(\lambda_N))U^*$, and in this case our algorithm coincides with the diagonalization. If A is nonnormal, then the diagonalization algorithm cannot be applied if A does not have a basis of eigenvectors. In principle, this is not a severe restriction, since a small perturbation can make it diagonalizable, but the eigenvectors can still be severely ill-conditioned, and this may lead to a significant loss of accuracy, as shown in Test 4.1.

On the other hand, the Schur–Parlett algorithm is a suitable choice for entire functions, but none of the branches of $r^{-1}(z)$ is. This algorithm reduces the computation of a primary matrix function to the evaluation of the same function on matrices whose eigenvalues lie in a small ball, and the latter evaluation is performed by using a truncation of the Taylor series expansion of f . This is a severe restriction, as the

Taylor series of $r^{-1}(x)$ in a neighborhood of the eigenvalue λ_i of A need not converge to $r^{-1}(\lambda_j)$, where λ_j is another eigenvalue of A near λ_i . For instance, the Taylor series expansion of the square root $z^{1/2}$ at $z_0 = -10 - i$, when evaluated at $z = -10 + i$, converges to $-(-10 + i)^{1/2}$ rather than to $(-10 + i)^{1/2}$. Moreover, if $r^{-1}(\lambda_i)$ is a critical point of r , then there exists no differentiable inverse of r extending $r^{-1}(\lambda)$ in a neighborhood of λ_i . For these reasons, we cannot consider the Schur–Parlett method in our experiments, and instead we focus our attention on the following algorithms.

- `invrat`: an implementation of Algorithm 4.1.
- `diag`: an implementation of the diagonalization approach to the evaluation of matrix functions. In order to evaluate $f(A)$, this algorithm exploits the eigen-decomposition $A =: UDU^{-1}$, with $U \in \mathbb{C}^{N \times N}$ nonsingular and $D \in \mathbb{C}^{N \times N}$ diagonal, and approximates $f(A)$ as $Uf(D)U^{-1}$. This algorithm works for diagonalizable matrices only.
- `approx_diag`: the variant of `diag` discussed by Davies [5]. In order to improve the stability of the diagonalization approach, this algorithm computes the eigen-decomposition of a nearby matrix $A + \varepsilon I = \tilde{U}\tilde{D}\tilde{U}^{-1}$ and then approximates $f(A)$ as $\tilde{U}f(\tilde{D})\tilde{U}^{-1}$.

The experiments were performed using the 64-bit version of MATLAB 2017b on a machine equipped with an Intel I5-5287U processor, running at 2.90GHz, and 8GiB of RAM. The accuracy of the algorithms is measured by the relative error, in the spectral norm, with respect to a reference solution computed by running `invrat` with about 512 digits of accuracy using the Advanpix Multiprecision Computing Toolbox [19]. We will denote the machine precision by u .

Test 4.1 (Forward stability). In this test, we investigate experimentally the forward stability of `invrat`, `diag`, and `approx_diag`. We consider the matrix equation $r(X) = A$, where

$$r(z) = \frac{\frac{z^3}{120} + \frac{z^2}{10} + \frac{z}{2} + 1}{-\frac{z^3}{120} + \frac{z^2}{10} - \frac{z}{2} + 1}$$

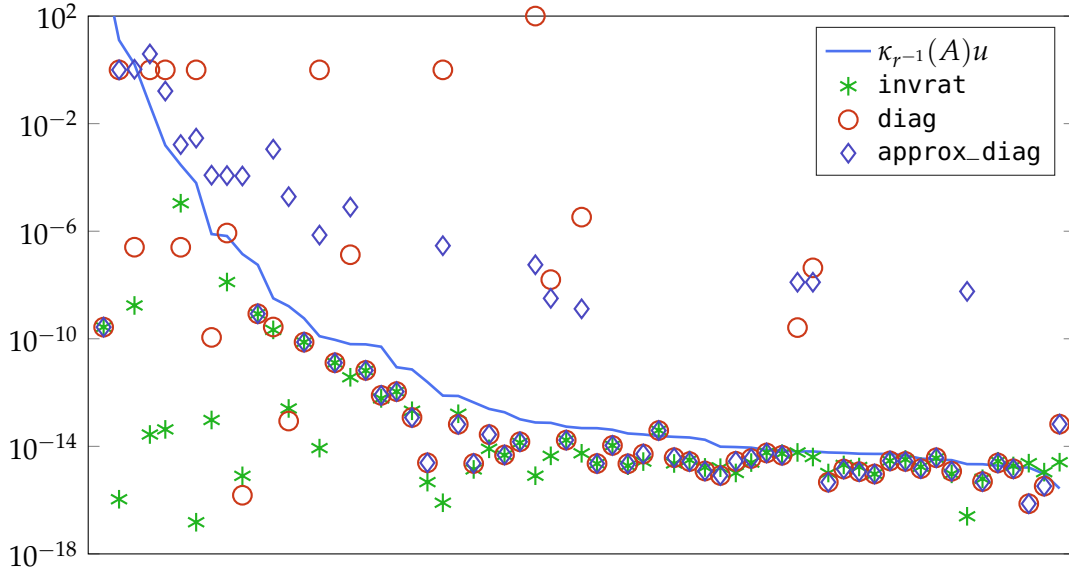


Figure 4.1: Relative forward errors of `invrat`, `diag`, and `approx_diag` on the test set.

is the $[3/3]$ Padé approximant to the exponential at 0 . For A , we consider a test set including 63 real and complex nonnormal matrices, of size between 2×2 and 10×10 , from the MATLAB gallery function and from the literature of the matrix logarithm.

Figure 4.1 compares the relative forward error of the three algorithms with the quantity $\kappa_{r^{-1}}(A)u$, the 1-norm condition number of a branch of r^{-1} that extends a real branch that contains 0 to the whole complex plane, estimated by means of the `funm_condest1` function from Higham's Matrix Function Toolbox [12].

Out of the three algorithms we consider, `diag` appears to be the most unreliable, as the relative forward error is of the order of 1 on more than 10% of the data set, and often several orders of magnitude larger than $\kappa_{r^{-1}}(A)u$. The forward error of `approx_diag` is larger than $\kappa_{r^{-1}}(A)u$ on almost 30% of the data set, but is of the order of 1 for four of the most ill-conditioned matrices only. Finally, the forward error of `invrat` is approximately bounded by $\kappa_{r^{-1}}(A)u$, which seems to indicate that the algorithm behaves in a forward stable manner.

Test 4.2. Critical solutions to the scalar equation $f(x) = y$ are ill-conditioned, and the effects of the ill-conditioning become obvious as the derivative of f approaches zero. This is the case for matrices as well, thus the accuracy of our algorithm, as that of any stable algorithm, will be affected by what solution is being computed. Since

an isolated solution is uniquely determined by its eigenvalues, choosing a solution of the scalar equation $r(x) = \lambda_i$, for each distinct eigenvalue λ_i of A is enough to fix what solution to $r(X) = A$ will be computed. This is equivalent to choosing an inverse r^{-1} of r and computing $X = r^{-1}(A)$, as discussed in Proposition 4.4.

In order to illustrate the numerical behavior of the Schur recurrence algorithm in computing different solutions of a matrix equation, we consider the equation $r(X) = A$, where $r(z) = -z/(z^2 + 1)$. This equation is equivalent to $AX^2 + X + A = 0$, which was considered for theoretical purposes in [17] and [18].

It is easy to show [17, Lem. 3] that the equation $r(z) = \lambda$, with $\lambda \in \mathbb{C}$ has two distinct solutions if and only if $\lambda \notin \{0, \pm 1/2\}$, while

- if $\lambda \in (-\infty, -1/2] \cup [1/2, +\infty)$ then the solutions have modulus 1;
- if $\lambda \in \mathcal{D} := (\mathbb{C} \setminus \mathbb{R}) \cup (-1/2, 0) \cup (0, 1/2)$, then one solution lies inside the unit disc, while the other lies outside.

We can identify two analytic branches for the inverse: $r_1^{-1} : \mathcal{D} \rightarrow \{z \in \mathbb{C} : |z| > 1\}$ and $r_2^{-1} : \mathcal{D} \cup \{0\} \rightarrow \{z \in \mathbb{C} : |z| < 1\}$, with branch cuts $(-\infty, -1/2] \cup [1/2, +\infty)$. The points $z = \pm 1$ are critical points for $r(z)$, indeed $r(\pm 1) = \mp 1/2$.

We show how the accuracy of a solution \tilde{X} to $r(X) = A$ degrades as the derivative of the function r at \tilde{X} approaches a singular matrix. This can occur in two cases: when two eigenvalues of A are close to each other but the corresponding eigenvalues of X are far apart (this may happen also when we choose the same branch for two nearby eigenvalues, if there is a branch cut in the middle); or when an eigenvalue of A is close to the image of a critical value of r and the corresponding eigenvalue of X is close to a critical point of r . We will examine one example for each situation.

Let us first consider the matrix $A = M \operatorname{diag}(1 - \varepsilon i, 1 + 2\varepsilon + \varepsilon i, 1 + 3\varepsilon + \varepsilon i) M^{-1}$, where $\varepsilon = 10^{-10}$, and $M \in \mathbb{R}^{3 \times 3}$ is a matrix with entries drawn from a standard normal distribution. As one can choose two branches of the inverse of r for each of the eigenvalues of A , there exist eight isolated primary solutions X . For each of them, we report in Table 4.1 the magnitude of the smallest eigenvalue of $Dr(X)$ and the forward error of the solution \tilde{X} computed by `invrat`. The solutions that select a different branch of the inverse of r for the eigenvalues on the opposite sides of the

Table 4.1: Solutions of the equation $r(X) = A$ in Test 4.2. The three columns contain the spectrum of X , the magnitude of the smallest eigenvalue of $Dr(X)$, and the relative error of the solution computed by `invrat`.

eigenvalues of X	$\kappa(Dr(X))$	$\ \tilde{X} - X\ _2 / \ X\ _2$
$\{r_1^{-1}(\lambda_1), r_1^{-1}(\lambda_2), r_1^{-1}(\lambda_3)\}$	$1.34 \times 10^{+10}$	1.86×10^{-06}
$\{r_1^{-1}(\lambda_1), r_1^{-1}(\lambda_2), r_2^{-1}(\lambda_3)\}$	$3.00 \times 10^{+10}$	3.57×10^{-06}
$\{r_1^{-1}(\lambda_1), r_2^{-1}(\lambda_2), r_1^{-1}(\lambda_3)\}$	$3.00 \times 10^{+10}$	3.60×10^{-06}
$\{r_1^{-1}(\lambda_1), r_2^{-1}(\lambda_2), r_2^{-1}(\lambda_3)\}$	$1.00 \times 10^{+00}$	1.96×10^{-16}
$\{r_2^{-1}(\lambda_1), r_1^{-1}(\lambda_2), r_1^{-1}(\lambda_3)\}$	$1.00 \times 10^{+00}$	3.32×10^{-16}
$\{r_2^{-1}(\lambda_1), r_1^{-1}(\lambda_2), r_2^{-1}(\lambda_3)\}$	$3.00 \times 10^{+10}$	1.84×10^{-06}
$\{r_2^{-1}(\lambda_1), r_2^{-1}(\lambda_2), r_1^{-1}(\lambda_3)\}$	$3.00 \times 10^{+10}$	4.04×10^{-06}
$\{r_2^{-1}(\lambda_1), r_2^{-1}(\lambda_2), r_2^{-1}(\lambda_3)\}$	$1.34 \times 10^{+10}$	4.91×10^{-07}

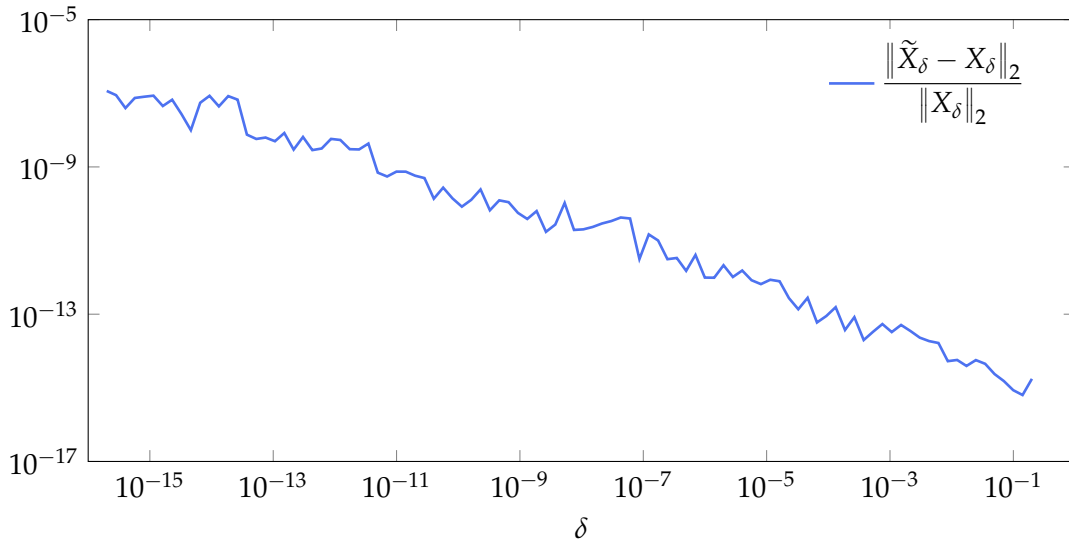


Figure 4.2: Relative error of the Schur algorithm for computing the solution of the matrix equation $r(X_\delta) = A_\delta$ in Test 4.2 with spectrum $\{r_1^{-1}(\lambda_1), r_1^{-1}(\lambda_2), r_1^{-1}(\lambda_3)\}$.

branch cut lead to a better conditioned Fréchet derivative, and the solution computed by `invrat` in this case has almost perfect accuracy.

To show that the accuracy of the solution computed by `invrat` are influenced by the distance of the eigenvalues of A from the images of critical points, we investigate the behavior of the algorithm when trying to compute solutions with almost critical eigenvalues. We consider the matrix $A_\delta = M \text{diag}(1/2 - \delta i, 1/2 - \delta, 1 + \delta i) M^{-1}$, where $M \in \mathbb{R}^{3 \times 3}$ is a random matrix as in the previous test. Note that the eigenvalues of A tend to the image of the branch point of r as $\delta > 0$ tends to zero. Figure 4.2 shows the relative error of the primary solution to $r(X_\delta) = A_\delta$ computed by `invrat`, as δ

varies between 2×10^{-16} and 2×10^{-1} . As expected, the accuracy of the solution is adversely affected by the proximity of the eigenvalues of A to the image of a critical point of r .

4.6 CONCLUSIONS

After discussing some properties of the solutions to the matrix equation $f(X) = A$, with f analytic, we developed an algorithm for computing primary solutions to the matrix equation $r(X) = A$, where r is a rational function. Our approach relies on a substitution algorithm based on Horner's scheme for the evaluation of numerator and denominator of r .

In previous work [11], [16] it has been shown that, for the k th root, the computational cost of the straightforward algorithm [22] can be reduced by considering substitution algorithms that exploit more efficient matrix powering schemes. However, a fraction can be evaluated in several different ways, and some approaches require fewer matrix multiplications than applying Horner's method twice. One such example is the Paterson–Stockmeyer method [21], which can require considerably fewer matrix multiplications for polynomials of high degree.

In principle, any of these alternative schemes could produce a substitution algorithm for the solution of the matrix equation $r(X) = A$. The computational cost of the substitution algorithm induced by a given evaluation scheme would be the same as the cost of the evaluation scheme itself, since the number of intermediate matrices to be computed depends on the number of matrix multiplications needed to evaluate numerator and denominator. Therefore, starting with a cheaper evaluation scheme for rational functions, it might be possible to develop cheaper algorithms for the solution of matrix functions of the form $r(X) = A$: this will be the subject of future investigation.

ACKNOWLEDGEMENTS

The authors are grateful to Nicholas J. Higham and an anonymous referee for providing feedback on the manuscript and useful comments which improved the presentation of the paper.

BIBLIOGRAPHY

- [1] A. H. AL-MOHY AND N. J. HIGHAM, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169.
- [2] Z. BAI AND J. W. DEMMEL, *On swapping diagonal blocks in real Schur form*, Linear Algebra Appl., 186 (1993), pp. 75–95.
- [3] Å. BJÖRCK AND S. HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
- [4] C. CARATHÉODORY, *Theory of Functions of a Complex Variable*, vol. 1, Chelsea Publishing, New York, NY, USA, 2nd ed., 1958.
- [5] E. B. DAVIES, *Approximate diagonalization*, SIAM J. Matrix Anal. Appl., 29 (2008), pp. 1051–1064.
- [6] P. I. DAVIES AND N. J. HIGHAM, *A Schur–Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 464–485.
- [7] J.-C. EVARD AND F. UHLIG, *On the matrix equation $f(X) = A$* , Linear Algebra Appl., 162-164 (1992), pp. 447–519.
- [8] M. FASI, N. J. HIGHAM, AND B. IANNAZZO, *An algorithm for the matrix Lambert W function*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 669–685.
- [9] W. H. FLEMING, *Functions of Several Variables*, Springer-Verlag, New York, NY, USA, 2nd ed., 1977.

- [10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4th ed., 2013.
- [11] F. GRECO AND B. IANNAZZO, *A binary powering Schur algorithm for computing primary matrix roots*, Numer. Algorithms, 55 (2010), pp. 59–78.
- [12] N. J. HIGHAM, *The Matrix Function Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mfttoolbox>.
- [13] —, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.
- [14] —, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [15] R. A. HORN AND C. R. JOHNSON, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.
- [16] B. IANNAZZO AND C. MANASSE, *A Schur logarithmic algorithm for fractional powers of matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 794–813.
- [17] B. IANNAZZO AND B. MEINI, *Palindromic matrix polynomials, matrix functions and integral representations*, Linear Algebra Appl., 434 (2011), pp. 174–184.
- [18] —, *The palindromic cyclic reduction and related algorithms*, Calcolo, 52 (2015), pp. 25–43.
- [19] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. <http://www.advanpix.com>.
- [20] B. N. PARLETT, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra Appl., 14 (1976), pp. 117–121.
- [21] M. S. PATERSON AND L. J. STOCKMEYER, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66.
- [22] M. I. SMITH, *A Schur algorithm for computing matrix p th roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989.

5

AN ARBITRARY PRECISION SCALING AND SQUARING ALGORITHM FOR THE MATRIX EXPONENTIAL

Abstract. The most popular algorithms for computing the matrix exponential are those based on the scaling and squaring technique. For optimal efficiency these are usually tuned to a particular precision of floating-point arithmetic. We design a new scaling and squaring algorithm that takes the unit roundoff of the arithmetic as input and chooses the algorithmic parameters in order to keep the forward error in the underlying Padé approximation below the unit roundoff. To do so, we derive an explicit expression for all the coefficients in an error expansion for Padé approximants to the exponential and use it to obtain a new bound for the truncation error. We also derive a new technique for selecting the internal parameters used by the algorithm, which at each step decides whether to scale or to increase the degree of the approximant. The algorithm can employ diagonal Padé approximants or Taylor approximants and can be used with a Schur decomposition or in transformation-free form. Our numerical experiments show that the new algorithm performs in a forward stable way for a wide range of precisions and that the most accurate of our implementations, the Taylor-based transformation-free variant, is superior to existing alternatives.

Keywords: multiprecision arithmetic, matrix exponential, matrix function, scaling and squaring method, Padé approximation, Taylor approximation, forward error analysis, MATLAB, `expm`.

2010 MSC: 15A16, 65F60.

5.1 INTRODUCTION

The exponential of a matrix has been the subject of much research in the 150 years or so since Laguerre first defined it [31], thanks to its many applications and in particular its central role in the solution of differential equations. Several equivalent definitions of this matrix function exist [20, Table 10.1], of which perhaps the most well known is the representation via its Taylor series expansion: the exponential of $A \in \mathbb{C}^{n \times n}$ is the matrix

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \quad (5.1)$$

Since the analogous series expansion of e^z , for $z \in \mathbb{C}$, has an infinite radius of convergence, the power series (5.1) is convergent for any $A \in \mathbb{C}^{n \times n}$ [20, Thm. 4.6], and truncating it to the first few terms gives a crude algorithm for approximating e^A . This method is known to be unsatisfactory—so much so that Moler and Van Loan [36], [37] take it as a lower bound on the performance of any algorithm for computing the matrix exponential.

The most popular method for computing the exponential of a matrix is the scaling and squaring algorithm paired with Padé approximation. This technique, originally proposed by Lawson [32], and further developed and analyzed by various authors over the past half-century, proves remarkably reliable in finite precision arithmetic, but its numerical stability is not fully understood. The method owes its name to the identity

$$e^A = (e^{2^{-s}A})^{2^s}, \quad (5.2)$$

and relies on the approximation

$$e^A \approx r_{km}(2^{-s}A)^{2^s}, \quad (5.3)$$

where $r_{km}(z)$ is the $[k/m]$ Padé approximant to e^z at 0 and the nonnegative integers k , m , and s are chosen so that $r_{km}(2^{-s}A)$ achieves a prescribed accuracy while minimizing the computational cost of the algorithm. In practice, diagonal approximants $r_m := r_{mm}$ are the most common choice, as symmetries in the coefficients of the numerator and denominator enable an efficient evaluation of $r_m(A)$.

In recent years there has been a sharp rise of interest in multiprecision computation, and the number of programming languages that support arbitrary precision floating point arithmetic, either natively or through dedicated libraries, is growing. In many cases, a wide range of arbitrary precision linear algebra kernels is available. Numerical routines for the evaluation of matrix functions are also sometimes provided, as we now explain.

The computer algebra systems Maple [33] and Mathematica [34] offer functions that can evaluate in arbitrary precision real matrix powers, the matrix logarithm, the matrix exponential, and a function that computes $f(A)$ given a scalar function f and a square matrix A . The open source computer algebra system Sage [42], [47] supports arbitrary precision floating point arithmetic, but does not implement any algorithms for the evaluation of matrix functions.

Turning to software focused on floating point arithmetic, the mpmath library [28] for Python provides functions for evaluating in arbitrary precision a wide range of matrix functions, including real powers, exponential, logarithm, sine, and cosine. MATLAB does not support arbitrary precision floating point arithmetic natively, but arbitrary precision floating-point data types are provided by the Symbolic Math Toolbox [44] and the Multiprecision Computing Toolbox [38]. Both toolboxes implement algorithms for the matrix square root, the exponential, the logarithm, and general matrix functions, and the Multiprecision Computing Toolbox also includes the hyperbolic and trigonometric sine and cosine of a matrix. Finally, the Julia language [6] supports multiprecision floating-point numbers by means of the built-in data type `BigFloat`, which provides only a few basic linear algebra kernels for arbitrary precision computation, and the `ArbFloats` package, a wrapper to the C library `Arb` [27] for

arbitrary-precision ball arithmetic, which is capable of computing the matrix square root and exponential.

The algorithms underlying the functions described above are not publicly available, to our knowledge. Nor are details of the implementations (albeit embodied in the source code of the open source packages), which in some cases may involve symbolic arithmetic.

The MATLAB function `expm` is a careful implementation of the algorithm of Al-Mohy and Higham [2], which relies on diagonal Padé approximants and exploits precomputed constants θ_m that specify how small the 1-norm of certain powers of a matrix A must be in order for $r_m(A)$ to provide an accurate approximation to e^A in IEEE double precision arithmetic. These constants are obtained by combining a floating point backward error analysis with a mix of symbolic and high precision computations, and, at the price of a computationally expensive algorithm design stage, provide a very efficient algorithm. For arbitrary precision computations, however, a new approach is required, since this procedure, despite being in principle repeatable for any given precision, is impractical to carry out when the accuracy at which the function should be evaluated is known only at runtime and should hence be treated as an input parameter to the algorithm.

The only published algorithm that we are aware of for computing the matrix exponential in arbitrary precision is that of Caliari and Zivcovich [7], which employs a scaling and squaring algorithm based upon Taylor approximation. It includes a new shifting technique less prone to overflow than the classic approach in [20, sect. 10.7.3] and a novel way to compute at runtime a bound on the backward error of the truncated Taylor series. The underlying backward error analysis relies on an explicit series expansion for the backward error of truncated Taylor series approximants [43] that does not readily extend to general Padé approximants, and the technique used to bound the error relies on a conjecture on the decay rate of the terms of this series expansion.

The goal of this work is to develop an algorithm for evaluating the exponential of a matrix in arbitrary precision floating point arithmetic that can be used with diagonal

Padé approximants or Taylor approximants and is fully rigorous. We wish to avoid symbolic computation and we are particularly interested in precisions higher than double. The algorithms we develop work in lower precision arithmetic as well, but they can suffer from overflow or underflow when formats with limited range, such as IEEE half precision, are used.

The techniques discussed here, together with those in [14], provide algorithms for evaluating in arbitrary precision most matrix functions that appear in applications. In particular, the inverse scaling and squaring algorithms for the matrix logarithm developed in [14] can be adapted in a straightforward way to the evaluation of fractional powers of a matrix [23], [24], whereas the algorithms proposed here can be used to compute trigonometric [1], [4], [17], [20, Chap. 12], [22], [25] and hyperbolic functions [1], [8], [22], and their inverses [5], by relying on functional identities involving only the matrix exponential.

The broad need for arbitrary precision matrix functions is clear from their inclusion in the software mentioned above. The need to compute the matrix exponential to high precision is needed, for example, in order to compute accurate solutions to the burnup equations in nuclear engineering [41]. Our particular interest in the matrix exponential stems not only from its many applications but also from algorithm development. Estimating the forward error of algorithms for matrix functions requires a reference solution computed in higher precision, and an arbitrary precision algorithm for the matrix exponential can be used both for the exponential and for other types of functions as mentioned above. Furthermore, such an algorithm allows us to estimate the backward error of algorithms for evaluating matrix functions defined implicitly by equations involving the exponential, such as the logarithm [3], [14], the Lambert W function [15], and inverse trigonometric and hyperbolic functions [5].

We derive in section 5.2 a new bound on the forward error of Padé approximants to the matrix exponential. We also make a conjecture that, if true, would lead to a more cheaply computable error bound. In section 5.3 we develop a novel algorithm for evaluating the exponential of a matrix in arbitrary precision. In section 5.4 we test experimentally several versions of this algorithm and compare their performance with

that of existing algorithms. In section 5.5 we summarize our findings and discuss future lines of research.

Finally, we introduce some notation. We denote by $\mathbb{R}^+ = \{x \in \mathbb{R} : x \geq 0\}$ the set of nonnegative real numbers, by \mathbb{N} the set of nonnegative integers, and by $\|\cdot\|$ any consistent matrix norm. The spectrum of $A \in \mathbb{C}^{n \times n}$ is denoted by $\sigma(A)$, its spectral radius by $\rho(A) = \max\{|\lambda| : \lambda \in \sigma(A)\}$, and the unit roundoff of floating point arithmetic by u . Given $f : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ and $A \in \mathbb{C}^{n \times n}$, we measure the sensitivity of $f(A)$ by means of the relative condition number

$$\kappa_f(A) = \lim_{\delta \rightarrow 0} \sup_{\|E\| \leq \delta \|A\|} \frac{\|f(A+E) - f(A)\|}{\delta \|f(A)\|},$$

which is given explicitly by [20, Thm. 3.1]

$$\kappa_f(A) = \frac{\|D_f(A)\| \|A\|}{\|f(A)\|}, \quad (5.4)$$

where $D_f : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ is the Fréchet derivative of f at A , which is the unique linear operator that, for all $E \in \mathbb{C}^{n \times n}$, satisfies $f(A+E) = f(A) + D_f(A)[E] + o(\|E\|)$.

5.2 PADÉ APPROXIMATION OF MATRIX EXPONENTIAL

The state of the art scaling and squaring algorithm for the matrix exponential relies on a bound on the relative backward error of Padé approximation in order to select suitable algorithmic parameters [2]. This approach requires an expensive precision-dependent design step that is unpractical to carry out when the precision at which the computation will be performed is known only at runtime. For that reason, we prefer to use a bound on the forward (truncation) error of the Padé approximants to the exponential that is cheap to evaluate at runtime.

Let f be a complex function analytic at o , and let $k, m \in \mathbb{N}$. The rational function $r_{km}(z) = p_{km}(z)/q_{km}(z)$ is the $[k/m]$ Padé approximant of f at o if $p_{km}(z)$ and $q_{km}(z)$ are

polynomials of degree at most k and m , respectively, the denominator is normalized so that $q_{km}(0) = 1$, and $f(z) - r_{km}(z) = O(z^{k+m+1})$.

The numerator and denominator of the $[k/m]$ Padé approximant to the exponential at 0 are [16, Thm. 5.9.1]

$$\begin{aligned} p_{km}(z) &= \sum_{j=0}^k \binom{k}{j} \frac{(k+m-j)!}{(k+m)!} z^j =: \sum_{j=0}^k \beta_j^{[k/m]} z^j, \\ q_{km}(z) &= \sum_{j=0}^m (-1)^j \binom{m}{j} \frac{(k+m-j)!}{(k+m)!} z^j =: \sum_{j=0}^m \delta_j^{[k/m]} z^j. \end{aligned} \quad (5.5)$$

In our algorithm, we will approximate e^A by means of the rational matrix function $r_{km}(A) = q_{km}(A)^{-1}p_{km}(A)$, which we evaluate by first computing $P = p_{km}(A)$ and $Q = q_{km}(A)$ and then solving a multiple right-hand side linear system in order to obtain $X := Q^{-1}P$. The computational efficiency of this method depends entirely on the evaluation scheme chosen to compute P and Q . In the literature, the customary choice is the Paterson–Stockmeyer method [40], which we now briefly recall.

Let us rewrite the polynomial $p(X) = \sum_{i=0}^k \alpha_i X^i$ as

$$p(X) = \sum_{i=0}^{\nu} B_i(X) X^{\nu i}, \quad (5.6)$$

where $\nu \leq k$ is a positive integer, $\nu = \lfloor k/\nu \rfloor$, and

$$B_i(X) = \begin{cases} \alpha_{\nu i + \nu - 1} X^{\nu-1} + \cdots + \alpha_{\nu i + 1} X + \alpha_{\nu i} I, & i = 0, \dots, \nu - 1, \\ \alpha_k X^{k - \nu \nu} + \cdots + \alpha_{\nu \nu + 1} X + \alpha_{\nu \nu} I, & i = \nu. \end{cases}$$

If we use Horner's method with (5.6), then the number of matrix multiplications required to evaluate $p(X)$ is

$$C_{\nu}^p(k) = \nu + \nu - 1 - \eta(\nu, k), \quad \eta(x, y) = \begin{cases} 1, & \text{if } x \text{ divides } y, \\ 0, & \text{otherwise,} \end{cases}$$

which is approximately minimized by taking either $\nu = \lfloor \sqrt{k} \rfloor$ or $\nu = \lceil \sqrt{k} \rceil$. Therefore evaluating $r_{km}(X)$ requires, in general,

$$C_\nu^r(k) = \nu_k + \nu_m + \left\lfloor \frac{k}{\nu_k} \right\rfloor + \left\lfloor \frac{m}{\nu_m} \right\rfloor - 2 - \eta(\nu_k, k) - \eta(\nu_m, m) \quad (5.7)$$

matrix multiplications, where ν_ℓ denotes $\sqrt{\ell}$ rounded to the nearest integer. This cost can be considerably reduced for diagonal Padé approximants (for which $k = m$) by exploiting the identity $p_m(X) = q_m(-X)$, where $p_m := p_{mm}$ and $q_m := q_{mm}$. By rewriting the numerator as

$$p_m(A) = \sum_{i=0}^m \beta_i A^i = \sum_{i=0}^{\lfloor m/2 \rfloor} \beta_{2i} A^{2i} + A \sum_{i=0}^{\lfloor m/2 - 1 \rfloor} \beta_{2i+1} A^{2i} =: U_e + AV =: U_e + U_o,$$

where U_o and U_e are the sums of the monomials with even and odd powers, respectively, we obtain that $q_m(A) = U_e - U_o$. By using ν stages of the Paterson–Stockmeyer method on A^2 , computing U_e and U_o requires one matrix product to form A^2 and $\nu - 1$ matrix multiplications to compute the first ν powers of A^2 ; evaluating U_e and V require $\lfloor \lfloor m/2 \rfloor / \nu \rfloor - \eta(\nu, \lfloor m/2 \rfloor)$ and $\lfloor \lfloor (m-1)/2 \rfloor / \nu \rfloor - \eta(\nu, \lfloor (m-1)/2 \rfloor)$ matrix multiplications, respectively; and computing U_o requires one additional multiplication by A . Therefore evaluating both $p_m(A)$ and $q_m(A)$ requires

$$C_\nu^e(m) = \nu + 1 + \left\lfloor \frac{\lfloor m/2 \rfloor}{\nu} \right\rfloor + \left\lfloor \frac{\lfloor (m-1)/2 \rfloor}{\nu} \right\rfloor - \eta(\nu, \lfloor m/2 \rfloor) - \eta(\nu, \lfloor (m-1)/2 \rfloor)$$

matrix multiplications, and it can be shown that $C_\nu^e(m)$ is approximately minimized by taking either $\nu = \lfloor \sqrt{m-1/2} \rfloor$ or $\nu = \lceil \sqrt{m-1/2} \rceil$. For m between 1 and 21, we have that $\min \{C_{\lfloor \sqrt{m} \rfloor}(m), C_{\lceil \sqrt{m} \rceil}(m)\} = \pi_m$, where the π_m are tabulated in [20, Table 10.3].

In principle, when designing an algorithm based on Padé approximation, one could use approximants of any order but, for any given cost, it is worth considering only the approximant that will deliver the most accurate result. By definition, this will be that of highest order, thus if evaluating the approximant of order m requires $C(m)$ matrix multiplications, an algorithm will typically examine only approximants of *optimal* order $m'(\zeta) = \max\{m : C(m) = \zeta\}$, for some $\zeta \in \mathbb{N}$. For truncated Taylor

series and diagonal Padé approximants to the exponential, the sequences of optimal orders are [13, eqs. (2.7) and (4.6)]

$$a_i^p = \left\lfloor \frac{(i+2)^2}{4} \right\rfloor, \quad i \in \mathbb{N}, \quad (5.8)$$

and

$$\begin{aligned} a_0^e &= 1, \quad a_1^e = 2, \\ a_i^e &= 2 \left\lfloor \frac{i-1}{4} \right\rfloor \left(i - 3 \left\lfloor \frac{i-1}{4} \right\rfloor \right) + 1, \quad i \in \mathbb{N} \setminus \{0, 1\}, \end{aligned} \quad (5.9)$$

respectively. Note that, for diagonal Padé approximants to the exponential, all optimal orders but a_1^e are odd. For a thorough discussion of the effect of rounding errors on the evaluation of matrix polynomials using the scheme (5.6) see [20, sect. 4.2].

5.2.1 Forward error

Now we present a new upper bound on the norm of the forward error of $r_{km}(A)$ as an approximation to e^A which in section 5.3, will play a central role in the design of a scaling and squaring algorithm for computing the matrix exponential in arbitrary precision. The leading term of the truncation error of the $[k/m]$ Padé approximant is known [16, Thm. 5.9.1], since $e^z - r_{km}(z) = c_{km}^1 z^{k+m+1} + O(z^{k+m+2})$, where

$$c_{km}^1 = (-1)^m \frac{k!m!}{(k+m)!(k+m+1)!}. \quad (5.10)$$

We begin by obtaining all the terms in the series expansion of $q_{km}(z)e^z - p_{km}(z)$.

Lemma 5.1. *Let $r_{km}(z) = p_{km}(z)/q_{km}(z)$ be the $[k/m]$ Padé approximant to e^z at 0. Then for all $z \in \mathbb{C}$,*

$$q_{km}(z)e^z - p_{km}(z) = \sum_{i=1}^{\infty} c_{k,m}^\ell z^{k+m+i}, \quad c_{k,m}^\ell = \frac{(-1)^m k!}{(k+m)!} \frac{(m+i-1)!}{(i-1)!(k+m+i)!}. \quad (5.11)$$

Proof. By equating the coefficients of z^{k+m+i} on the left- and right-hand side of the first equation in (5.11), we obtain that $c_{k,m}^\ell$ is the sum, for j from 0 to m , of the

j th coefficient of $q_{km}(z)$ multiplied by the $(k + m + i - j)$ th coefficient of the series expansion of e^z :

$$c_{k,m}^\ell = \sum_{j=0}^m \delta_j^{[k/m]} \frac{1}{(k + m + i - j)!} - 0 = \frac{1}{(k + m)!} \sum_{j=0}^m (-1)^j \binom{m}{j} \frac{(k + m - j)!}{(k + m + i - j)!}.$$

We prove (5.11) by induction on m . For $m = 1$ we have

$$\begin{aligned} c_{k,m}^\ell &= \frac{1}{(k + 1)!} \left(\binom{1}{0} \frac{(k + 1)!}{(k + 1 + i)!} - \binom{1}{1} \frac{k!}{(k + i)!} \right) \\ &= \frac{k!}{(k + 1)!(k + 1 + i)!} (k + 1 - k - 1 - i) = \frac{(-1)^1 k! i!}{(k + 1)!(k + i)!(i - 1)!}. \end{aligned}$$

By exploiting the identity $\binom{a+1}{b} = \binom{a}{b} + \binom{a}{b-1}$, for the inductive step we have

$$\begin{aligned} c_{k,m+1}^\ell &= \frac{1}{(k + m + 1)!} \sum_{j=0}^{m+1} (-1)^j \binom{m+1}{j} \frac{(k + m + 1 - j)!}{(k + m + 1 + i - j)!} \\ &= \frac{1}{(k + m + 1)!} \left(\frac{(k + m + 1)!}{(k + m + 1 + i)!} + \sum_{j=1}^m (-1)^j \binom{m}{j} \frac{(k + m + 1 - j)!}{(k + m + 1 + i - j)!} \right. \\ &\quad \left. + \sum_{j=1}^m (-1)^j \binom{m}{j-1} \frac{(k + m + 1 - j)!}{(k + m + 1 + i - j)!} + (-1)^{m+1} \frac{k!}{(k + i)!} \right) \\ &= \frac{1}{(k + m + 1)!} \sum_{j=0}^m (-1)^j \binom{m}{j} \left(\frac{((k + 1) + m - j)!}{((k + 1) + m + i - j)!} - \frac{(k + m - j)!}{(k + m + i - j)!} \right) \\ &= \frac{1}{(k + m + 1)!} \left(\frac{(-1)^m (k + 1)! (m + i - 1)!}{(i - 1)! (m + k + i + 1)!} - \frac{(-1)^m k! (m + i - 1)!}{(i - 1)! (m + k + i)!} \right) \\ &= \frac{(-1)^m k! (m + i - 1)!}{(k + m + 1)! (i - 1)! (m + k + i + 1)!} (k + 1 - m - k - i - 1) \\ &= \frac{(-1)^{m+1} k! ((m + 1) + i - 1)!}{(k + (m + 1))! (i - 1)! ((m + 1) + k + i)!}, \end{aligned}$$

which concludes the proof. \square

This result can be exploited to bound the truncation error of $r_{km}(A)$. We will use the result in [2, Thm. 4.2(a)]: if $f(x) = \sum_{i=\ell}^{\infty} c_i x^i$ and c_i has the same sign for all $i \geq \ell$, then for any X such that

$$\alpha_d(X) := \min\{\|X^d\|^{1/d}, \|X^{d+1}\|^{1/(d+1)}\}, \quad d(d-1) \leq \ell, \quad (5.12)$$

is less than the radius of convergence of the series, we have

$$\|f(X)\| \leq \sum_{i=\ell}^{\infty} |c_i| \alpha_d(X)^i = \left| \sum_{i=\ell}^{\infty} c_i \alpha_d(X)^i \right| = |f(\alpha_d(X))|. \quad (5.13)$$

An alternative definition of $\alpha_d(X)$ has been recently proposed for the computation of the wave-kernel matrix functions [39]. This more refined strategy requires the computation of $\|A^{d_i}\|_1$ for all d_i, d_j such that $\gcd(d_i, d_j) = 1$ and $d_i d_j - d_i - d_j < k + m$. The cost of finding all such pairs is difficult to determine, but it must be at least $O((k + m) \log(k + m))$ operations, as there are at least $O((k + m)^2)$ pairs to test and Euclid's algorithm for finding the greatest common divisor of two integers $a, b \in \mathbb{N}$ such that $a < b$ requires $2 \log_2 a + 1$ operations in the worst case. Moreover, even if all the pairs to be tested were known, the cost of evaluating the bound would increase with k and m , and as both can be potentially large when resorting to high precision, we prefer to use the cheaper bound given by (5.12). However, all the results in this section can be modified by replacing $\alpha_d(X)$ with

$$\alpha^{[k/m]}(X) := \min_{\substack{\gcd(a,b) \\ ab-a-b < k+m}} \max \left\{ \|X^a\|^{1/a}, \|X^b\|^{1/b} \right\}.$$

For the truncation error of the $[k/m]$ Padé approximant to the matrix exponential, we would like to obtain a bound of the form

$$\|e^X - r_{km}(X)\| \leq |e^{\alpha_d(X)} - r_{km}(\alpha_d(X))|, \quad (5.14)$$

which would be true if all the nonzero terms in the series expansion at 0 of $e^z - r_{km}(z)$ had the same sign. By Lemma 5.1, this would be true if $q_{km}(z)^{-1}$ had a power series expansion with all coefficients of the same sign. This applies to Taylor approximants $t_k := r_{k0}$, since $q_{km}(z)^{-1} = 1$, and by (5.13) we can derive the bound

$$\|e^X - t_k(X)\| = \left\| \sum_{i=k+1}^{\infty} \frac{1}{i!} X^i \right\| \leq \left\| \sum_{i=k+1}^{\infty} \frac{\alpha_d(X)^i}{i!} \right\| = |e^{\alpha_d(X)} - t_k(\alpha_d(X))|.$$

For all other even values of m that we have checked, this is not the case. For example, the series expansion for the reciprocal of the denominator of the $[2/2]$ Padé approximant is

$$\frac{1}{q_{22}(z)} = 1 + \frac{z}{2} + \frac{z^2}{6} + \frac{z^3}{24} + \frac{z^4}{144} - \frac{z^6}{1728} + O(z^7).$$

However, for the algorithm we are designing we are interested only in bounding the forward error of diagonal approximants of optimal degree, and from (5.9) we know that most optimal degrees are, in fact, odd. Experimental evidence suggests that, in this case, the coefficients of the series expansion are indeed one-signed.

Conjecture 5.2. *Let $k, m \in \mathbb{N}$. If m is odd, then all the coefficients of the series expansion of $q_{km}(z)^{-1}$ at 0 are positive.*

In order to prove this conjecture, we attempted to derive an explicit expression for the coefficients of the Maclaurin expansion of $q_{km}(z)^{-1}$. First, we tried to decompose this function into the product of simpler terms whose Taylor expansion at 0 had one-signed coefficients, but we were unable to find a suitable factorization of $q_{km}(z)$. Next, by interpreting $q_{km}(z)^{-1}$ as the composition of z^{-1} and $q_{km}(z)$, we considered using the well-known Faà di Bruno's formula [11], [12] to compute higher-order derivatives of this function. By exploiting well-known results [29], we obtained several equivalent expressions for the quantities we were examining, but none of them led us to a proof of Conjecture 5.2.

If the latter were true, then we could use the bound (5.14) for all diagonal approximants of degree a_i^e in (5.9) for $i \in \mathbb{N} \setminus \{1\}$, but since we do not have a proof of the conjecture we will bound the truncation error of r_{km} for any k and m . We will use the next result, which combines Lemma 5.1 and (5.13).

Corollary 5.3 (bound on the truncation error of Padé approximants). *Let us denote the $[k/m]$ Padé approximant to e^z at 0 by $r_{km} = p_{km}/q_{km}$. Then for $X \in \mathbb{C}^{n \times n}$ and any positive integer d such that $d(d-1) \leq k+m+1$,*

$$\|e^X - r_{km}(X)\| \leq \|q_{km}(X)^{-1}\| \left| q_{km}(\alpha_d(X))e^{\alpha_d(X)} - p_{km}(\alpha_d(X)) \right| \quad (5.15)$$

Proof. By (5.13), we have

$$\begin{aligned} \|e^X - r_{km}(X)\| &= \|q_{km}(X)^{-1}(q_{km}(X)e^X - p_{km}(X))\| \\ &\leq \|q_{km}(X)^{-1}\| \|q_{km}(X)e^X - p_{km}(X)\| \\ &\leq \|q_{km}(X)^{-1}\| \|q_{km}(\alpha_d(X))e^{\alpha_d(X)} - p_{km}(\alpha_d(X))\|, \end{aligned}$$

where for the last inequality we used the fact that the coefficients of the series expansion of $q_{km}(z)e^z - p_{km}(z)$ all have the same sign, by Lemma 5.1. \square

Since the norm of $q_{km}^{-1}(X)$ does not depend on $\alpha_d(X)$, the bound in (5.15) is nondecreasing in $\alpha_d(X)$ and therefore is minimized by choosing for d the value

$$d^* = \arg \min_{1 \leq d \leq d^{[k/m]}} \alpha_d(X), \quad (5.16)$$

where

$$d^{[k/m]} = \max\{d \in \mathbb{N} : d(d-1) \leq k+m+1\} = \left\lfloor \frac{1 + \sqrt{5 + 4(k+m+1)}}{2} \right\rfloor. \quad (5.17)$$

Depending on the size of k and m , this choice might require the estimation of $\alpha_d(X)$ for too many values of d , and thus be unpractical. On the other hand, it has been observed [2] that the sequence $(\alpha_d(X))_{d \in \mathbb{N}}$ is typically roughly decreasing, so it is reasonable to use the considerably cheaper approximation $\alpha_{d^{[k/m]}}(X)$. In our algorithm, we adopt an intermediate approach that has the same cost as the computation of $\alpha_{d^{[k/m]}}(X)$, but improves on it by reusing previously computed quantities. We discuss this in detail in section 5.3.

5.3 A MULTIPRECISION ALGORITHM

In this section we develop a novel scaling and squaring method for computing the matrix exponential in arbitrary precision floating point arithmetic. Our algorithm differs from traditional scaling and squaring approaches, such as those of Al-Mohy

and Higham [2] and Caliari and Zivcovich [7], in several respects. First, it relies on a bound on the forward error rather than on the backward error of the Padé approximants to the matrix exponential and avoids the use of any precomputed precision-dependent constants by evaluating at runtime the bound (5.15) for some choice of d . Moreover, unlike scaling and squaring algorithms for double precision based on diagonal Padé approximants [2], [19], [20, Alg. 10.20], [21], which use approximants of order at most 13 and a nonzero scaling parameter only if the approximant of highest degree is expected not to deliver either a truncation error smaller than u or an accurate evaluation of $r_{13}(A)$, our algorithm blends the two phases together and tries to determine both parameters at the same time.

Our arbitrary precision scaling and squaring algorithm for the computation of e^A is given in Algorithm 5.1. Besides the matrix $A \in \mathbb{C}^{n \times n}$, the algorithm accepts several additional input arguments.

- The arbitrary precision floating point parameter $u \in \mathbb{R}^+$ specifies the unit round-off of the working precision of the algorithm.
- The Boolean parameter `use_taylor` specifies the kind of Padé approximants the algorithm will use: truncated Taylor series if set to **true**, diagonal Padé approximants otherwise.
- The parameter $u_{bnd} \in \mathbb{R}^+$ specifies the unit roundoff of the precision used to evaluate $\|q_{m_i}(2^{-s}A)^{-1}\|_1$ in (5.18) below. The value of u_{bnd} is ignored if `use_taylor` is set to **true**.
- The vector $\mathfrak{m} \in \mathbb{N}^N$, sorted in ascending order, specifies what orders of Padé approximants the algorithm can consider. The algorithm will select i between 1 and N , and then evaluate either the truncated Taylor series of order \mathfrak{m}_i or the $\lfloor \mathfrak{m}_i/\mathfrak{m}_i \rfloor$ Padé approximant, depending on the value of `use_taylor`.
- The nonnegative integer s_{\max} specifies the maximum number of binary powerings the algorithm is allowed to compute during the squaring stage, or, equiva-

Algorithm 5.1: Scaling and squaring algorithm for the matrix exponential.

Given $A \in \mathbb{C}^{n \times n}$, this algorithm computes an approximation to e^A in floating point arithmetic with unit roundoff u using a scaling and squaring method based upon Padé approximants. The pseudocode of `EVALBOUND` and `EVALPADE` is given in Fragment 5.3, that of `EVALBOUNDTAYL` and `EVALPADETAYL` in Fragment 5.5, and that of `RECOMPDIAGS` in Fragment 5.2.

```

1  $\mathcal{A}_0 \leftarrow I$ 
2 if use_taylor then
3    $\text{EVALBOUND} \leftarrow \text{EVALBOUNDTAYL}$ 
4    $\text{EVALPADE} \leftarrow \text{EVALPADETAYL}$ 
5    $\mathcal{A}_1 \leftarrow A$ 
6 else
7    $\text{EVALBOUND} \leftarrow \text{EVALBOUNDDIAG}$ 
8    $\text{EVALPADE} \leftarrow \text{EVALPADEDIAG}$ 
9    $\mathcal{A}_1 \leftarrow A^2$ 
10  $s \leftarrow 0$ 
11  $i \leftarrow 0$ 
12  $\gamma \leftarrow [-\infty, -\infty, \dots]$ 
13  $\alpha_{\min} \leftarrow \infty$ 
14  $\delta_{old} \leftarrow \infty$ 
15  $[\delta, \psi, \kappa_A] \leftarrow \text{EVALBOUND}(A, m_i, s)$ 
16 while  $\delta \geq u\psi$  and  $s < s_{\max}$  and  $i \leq N$  do
17   if  $\kappa_A \geq \zeta(u)$  or  $\delta_{old} < \delta^2$  then
18      $s \leftarrow s + 1$ 
19   else
20      $i \leftarrow i + 1$ 
21    $\delta_{old} \leftarrow \delta$ 
22    $[\delta, \psi, \kappa_A] \leftarrow \text{EVALBOUND}(A, m_i, s)$ 
23  $Y \leftarrow \text{EVALPADE}(A, m, s)$ 
24 if ISQUASIUPPERTRIANGULAR( $A$ ) then
25    $Y \leftarrow \text{RECOMPDIAGS}(2^{-s}A, Y)$ 
26 for  $t \leftarrow 1$  to  $s$  do
27    $Y \leftarrow Y^2$ 
28   if ISQUASIUPPERTRIANGULAR( $A$ ) then
29      $Y \leftarrow \text{RECOMPDIAGS}(2^{-s+t}A, Y)$ 
30 return  $X$ 

```

lently, the maximum number of times the matrix can be multiplied by $\frac{1}{2}$ during the initial scaling stage.

- The function parameter $\zeta : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ specifies a precision-dependent value that is used to predict whether the evaluation of $q_{m_i}(A)$ will be accurate or not. This parameter is not used when `use_taylor` is set to **true**.

We now discuss the outline of Algorithm 5.1. The variables \mathcal{A} , γ , and α_{\min} are assumed to be available within the following code fragments (that is, their scope is global). The Boolean variable `use_taylor` chooses between the auxiliary functions in Fragments 5.3 and 5.5, tailored to the case of diagonal Padé approximants and truncated Taylor series, respectively. Finally, we use the notation $[x, x, \dots]$, to denote a vector whose elements are all initialized to x and whose length is unimportant. We assume that very few of its entries will take a value different from the default, and thus that such a vector can be stored in a memory-efficient way.

The algorithm starts by determining a suitable order and a scaling parameter s for the scaling and squaring method. To this end, on line 10–11 it sets s and i to 0 and then increments them, trying to find a choice for which the right-hand side of (5.15), for $X = 2^{-s}A$, $k = m_i$, and $m = k$ or $m = 0$, is smaller than $u\psi(2^{-s}A)$, where $\psi(X)$ approximates $\|e^X\|_1$. As long as this condition is not satisfied, two heuristics are used to decide which parameter is more convenient to change. One approach is aimed at keeping the evaluation of the Padé approximant as accurate as possible, by taking into account the conditioning of q_{m_i} ; being specific to diagonal approximants this is discussed in section 5.3.1. On the other hand, we noticed that when $\alpha_d(2^{-s}A) \gg 1$, the bound (5.15) can sometimes decrease exceedingly slowly as m increases, leading to the use of an approximant of degree much larger than needed, which in turn causes loss of accuracy and unnecessary computation. We found that monitoring the rate at which our bound on the truncation error of the approximant decreases provides an effective strategy to prevent this from happening. In particular, we increment s when the bound on the truncation error does not decrease at least quadratically, that is, when $\delta_{old} < \delta^2$, where δ_{old} and δ are the values of the error bound at the previous and current iteration of the while loop on line 16 of Algorithm 5.1, respectively.

Fragment 5.2: Recomputation of the diagonals.

```

1 function RECOMPDIAGS( $A \in \mathbb{C}^{n \times n}, X \in \mathbb{C}^{n \times n}$ )
   $\triangleright$  Compute main diagonal and first upper-diagonal of  $X \approx e^A$  from  $A$ .
2 for  $i = 1$  to  $n$  do
3   if  $i = n - 1$  or  $i \leq n - 2$  and  $a_{i+2,i+1} = 0$  then
4     if  $a_{i+1,i} = 0$  then
5       | Recompute  $x_{i,i}, x_{i,i+1}, x_{i+1,i+1}$  using [20, Eq. (10.42)].
6     else
7       | Recompute  $x_{i,i}, x_{i,i+1}, x_{i+1,i}, x_{i+1,i+1}$  using [2, Eq. (2.2)].
8      $i \leftarrow i + 1$ 
9   else
10    |  $x_{i,i} \leftarrow e^{a_{i,i}}$ 

```

As soon as a combination of scaling parameter and Padé approximant order is found, the algorithm computes Y by evaluating the Padé approximant (diagonal or Taylor) of order m_i at $2^{-s}A$, and finally computes $e^A \approx Y^{2^s}$, by applying s steps of binary powering to Y . If A is upper quasi-triangular, then in order to improve the accuracy of the final result, the function RECOMPDIAGS in Fragment 5.2 is used to recompute the diagonal and first upperdiagonal of the intermediate matrices from the elements of A , as recommended by Al-Mohy and Higham [2].

In the next two sections, we discuss how the functions EVALBOUND and EVALPADE can be implemented efficiently for diagonal Padé approximants and truncated Taylor series.

5.3.1 Diagonal Padé approximants

When `use_taylor` is set to **false**, that is, when diagonal Padé approximants are being considered, the condition that needs to be tested on lines 15 and 22 of Algorithm 5.1 is, for some d such that $d(d-1) < 2m_i + 1$,

$$\left\| q_{m_i}(2^{-s}A)^{-1} \right\|_1 \left| q_{m_i}(\alpha_d(2^{-s}A))e^{\alpha_d(2^{-s}A)} - p_{m_i}(\alpha_d(2^{-s}A)) \right| < u\psi(2^{-s}A), \quad (5.18)$$

As discussed in section 5.2, the choice of $\alpha_d(X)$ that would guarantee the best bound is $\alpha_{d^*}^{[\mathfrak{m}_i/\mathfrak{m}_i]}(X)$, for d^* in (5.16), but this value can become impractical to compute, even for Padé approximants of relatively low degree. Taking $\alpha_{d^{[\mathfrak{m}_i/\mathfrak{m}_i]}}(X)$, where $d^{[\mathfrak{m}_i/\mathfrak{m}_i]}$ is defined in (5.17), on the other hand, is appealing because this estimate requires the evaluation of $\|A^d\|_1^{1/d}$ for at most two values of d independently of the value of \mathfrak{m}_i , and is often not far from the best choice, since the sequence $(\|A^d\|_1^{1/d})_{d \in \mathbb{N}}$ is typically roughly decreasing [2].

However, it is sometimes possible to obtain a better bound at almost no extra cost, by reusing quantities computed during previous steps of the algorithm. Observe that, since $\|(2^{-s}A)^d\|_1^{1/d} = 2^{-s}\|A^d\|_1^{1/d}$ and thus $\alpha_d(2^{-s}A) = 2^{-s}\alpha_d(A)$, it is enough to estimate the norm of powers of A and then scale their value as required. Moreover, since the algorithm considers the approximants in nondecreasing order of cost, the value of $d^{[\mathfrak{m}_i/\mathfrak{m}_i]}$ is nondecreasing in i . Therefore, in (5.18) we can replace $\alpha_d(2^{-2}A)$ by $2^{-s}\alpha_{\min}$, where α_{\min} is a variable that keeps track of the smallest value of $\alpha_{d^{[\mathfrak{m}_i/\mathfrak{m}_i]}}(X)$ computed so far, and is updated only when a new value $\alpha_{d^{[\mathfrak{m}_j/\mathfrak{m}_j]}}(X) < \alpha_{\min}$ is found for some $j > i$.

Since only the order of magnitude of α_{\min} is actually needed, we estimate $\|A^d\|_1$ by running in precision u_{bnd} the 1-norm estimation algorithm of Higham and Tisseur [26]. This method repeatedly computes the action of A on a tall and skinny matrix without explicitly forming any powers of A , and thus requires only $O(n^2)$ flops. In the pseudocode, the 1-norm estimation is performed by the function `NORMEST1`, whose only input is a function that computes the product AX given the matrix $X \in \mathbb{C}^{n \times t}$. In order to keep the notation as succinct as possible, anonymous functions are specified using a lambda notation, and $\lambda x.f(x)$ denotes a function that replaces all the occurrences of x in the body of f with the value of its input argument.

By storing the values of $\|A^d\|_1$ in the global array γ , the 1-norm of each power of A is estimated at most once. Further computational savings can be achieved by computing some carefully chosen powers of A within the algorithm and using them to evaluate of the action of powers of A on a vector, as we will discuss later.

As long as the bound (5.18) is not satisfied, the algorithm can decide to either increment the scaling factor or increase the order of the Padé approximant, since either choice will reduce the truncation error of the approximation. Both options, however, may have an adverse effect on the numerical behavior of the algorithm, since taking a Padé approximant of higher degree may significantly increase the conditioning of the coefficient of the linear system to be solved, thus jeopardizing the accurate evaluation of the approximant, whereas increasing s will increase the number of matrix multiplications that will occur during the squaring phase of the algorithm, which is the most sensitive to rounding errors, as shown by [20, Thm. 10.21].

We solve this dilemma by means of a heuristic that prevents the 1-norm condition number of $q_{k_i m_i}(2^{-s}A)$ from getting too large. In particular, if our estimate $\kappa_A = \text{NORMEST1}(\lambda x \cdot q_m(2^{-s}A)^{-1}x) / \text{NORMEST1}(\lambda x \cdot q_m(2^{-s}A)x)$ is larger than a constant $\zeta(u)$ that depends on the unit roundoff u , we update the scaling parameter and leave the order of the Padé approximant unchanged. Otherwise, we increment i and take an approximant of higher order chosen according to the elements in \mathfrak{m} . In practice, we set $\zeta(u) := u^{-1/8}$. For IEEE double precision, this choice gives $\zeta(u) = 2^{53/8} \approx 98.70$, which agrees (within a factor of 1.4) with the largest condition number allowed by Al-Mohy and Higham [2, Table 3.1] for double precision.

Within our algorithm, we can exploit the evaluation scheme discussed in section 5.2 to reduce the computational cost of the evaluation not only of $r_{\mathfrak{m}_i}(2^{-s}A)$, but also of the term $\|q_{\mathfrak{m}_i}(2^{-2}A)\|_1$ appearing in the bound (5.18).

Since Algorithm 5.1 considers Padé approximants of increasing cost, for diagonal Padé approximants we have that $\mathfrak{m}_i < \mathfrak{m}_j$ for $i < j$. Hence, whenever in Algorithm 5.1 the bound (5.18) is evaluated for the approximant of order \mathfrak{m}_i , we are guaranteed that on line 23 $r_{\mathfrak{m}_j}(2^{-s}A)$ will be evaluated for some $j \geq i$. Since numerator and denominator of the approximant are evaluated by means of the Paterson–Stockmeyer method, we know that at least the first $\nu = \lceil \sqrt{\mathfrak{m}_i} \rceil$ powers of $2^{-s}A$ will be needed, and since scaling a matrix requires only $O(n^2)$ flops, it is worth it to compute immediately the first ν powers of A , and subsequently use them to speed up the estimation of $\|A^d\|_1^{1/d}$, $\|q_{\mathfrak{m}_i}^{-1}(2^{-s}A)\|_1$, and $\|e^A\|_1$.

Fragment 5.3: Auxiliary functions for diagonal Padé approximants r_m .

```

1 function EVALBOUNDDIAG( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, s \in \mathbb{N}$ )
  ▷ Check (5.18) for  $r_m$  and estimate  $\kappa_1(q_m(A))$ .
2  $\alpha_{\min} \leftarrow \text{OPTALPHADIAG}(A, m, \alpha_{\min})$ 
3  $[U_e, U_o] \leftarrow \text{EVALPADEDIAGAUx}(A, m, s, u_{\text{bnd}})$ 
4 Set working precision to  $u_{\text{bnd}}$ .
5  $[L, U] \leftarrow \text{LU}(U_e - U_o)$ 
6  $\eta \leftarrow \text{NORMEST1}(\lambda x. (U^{-1}(L^{-1}x))$ 
7 Set working precision to  $u$ .
8  $\delta \leftarrow \eta |q_m(2^{-s}\alpha_{\min})e^{2^{-s}\alpha_{\min}} - p_m(2^{-s}\alpha_{\min})|$ 
9  $\psi \leftarrow \text{NORMEST1}(\lambda x. (U^{-1}(L^{-1}(2U_o x)) + x))$ 
10  $\kappa_A \leftarrow \eta \text{NORMEST1}(\lambda x. (U_e - U_o)x)$ 
11 return  $\delta, \psi, \kappa_A$ 

```

```

12 function EVALPADEDIAG( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, s \in \mathbb{N}$ )
  ▷ Evaluate  $r_m(2^{-s}A)$ .
13  $[U_e, U_o] \leftarrow \text{EVALPADEDIAGAUx}(A, m, s, u)$ 
14  $[L, U] \leftarrow \text{LU}(U_e - U_o)$ 
15 return  $U^{-1}(L^{-1}(2U_o + I))$ 

```

```

16 function EVALPADEDIAGAUx( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, s \in \mathbb{N}, u \in \mathbb{R}^+$ )
  ▷ Evaluate components of  $p_{mm}(A)$  and  $q_{mm}(A)$ .
17 Set working precision to  $u$ .
18  $\beta_e \leftarrow \left[ \frac{m!(2m-2i)!}{(2m)!(m-2i)!(2i)!} \right]_{i=0}^{\lfloor m/2 \rfloor} \quad \beta_o \leftarrow \left[ \frac{m!(2m-2i-1)!}{(2m)!(m-2i-1)!(2i+1)!} \right]_{i=0}^{\lfloor m/2-1 \rfloor}$ 
19 return  $\text{EVALPOLYPS}(2s, \beta_e, \lceil \sqrt{m} \rceil), (2^{-s}A) \text{EVALPOLYPS}(2s, \beta_o, \lceil \sqrt{m} \rceil)$ 

```

```

20 function OPTALPHADIAG( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, \alpha_{\min} \in \mathbb{R}^+$ )
  ▷ Compute  $\alpha_{\min}$ .
21  $d \leftarrow \left\lfloor \frac{1+\sqrt{5+8m}}{2} \right\rfloor$ 
22 if  $\gamma_d = -\infty$  then
23    $\gamma_d \leftarrow \text{NORMEST1}(\lambda x. \text{EVALPOWVECdiag}(d, x))^{1/d}$ 
24 if  $\gamma_{d+1} = -\infty$  then
25    $\gamma_{d+1} \leftarrow \text{NORMEST1}(\lambda x. \text{EVALPOWVECdiag}(d+1, x))^{1/(d+1)}$ 
26 return  $\min\{\max\{\gamma_d, \gamma_{d+1}\}, \alpha_{\min}\}$ 

```

```

27 function EVALPOWVECdiag( $d \in \mathbb{N}, X \in \mathbb{C}^{n \times t}$ )
  ▷ Compute  $A^d X$  using elements in  $\mathcal{A}$ .
28  $\ell \leftarrow \text{length}(\mathcal{A})$ 
29 while  $d > 1$  and  $\ell > 1$  do
30   for  $i \leftarrow 1$  to  $\lfloor d/(2\ell) \rfloor$  do
31      $X \leftarrow \mathcal{A}_\ell X$ 
32    $d \leftarrow d \bmod 2\ell$ 
33    $\ell \leftarrow \min\{\ell - 1, \lfloor d/2 \rfloor + 1\}$ 
34 if  $d = 1$  then
35    $X \leftarrow AX$ 
36 return  $X$ 

```

Fragment 5.3 shows how the bound (5.18) can be evaluated efficiently for diagonal Padé approximants. In order to estimate $\|q_m^{-1}(2^{-s}A)\|_1$, the algorithm computes the matrices U_e and U_o using the Paterson–Stockmeyer method given in Fragment 5.4. This implementation stores the powers of A^2 in the global array \mathcal{A} , which is updated only when it does not already contain the first $\lceil \sqrt{m} \rceil$ powers of A^2 . Since the number of matrices stored in \mathcal{A} changes with m , we introduce a function $\text{length}(\mathcal{A})$, that returns the number of positive powers stored in \mathcal{A} . In other words, if $\text{length}(\mathcal{A}) = \ell$, then \mathcal{A} contains $\ell + 1$ matrices from $\mathcal{A}_0 = I$ to $\mathcal{A}_\ell = A^{2\ell}$.

Note that although it makes sense, from a performance point of view, to compute U_e and U_o in lower precision, the elements of \mathcal{A} must be computed at precision u in order to be reused to evaluate numerator and denominator of $r_m(2^{-s}A)$. These lower precision approximation of U_e and U_o can be used to compute a cheap approximation $\psi(e^{2^{-s}A})$ to $\|e^{2^{-s}A}\|_1$ needed in (5.18), since $q_m(X)^{-1}p_m(X) = 2(U_e - U_o)^{-1}U_o + I$ can be evaluated by means of only one multiple right-hand side system solve at precision u_{bnd} .

In addition, the elements of \mathcal{A} can be used to reduce the computational cost of estimating the 1-norm of powers of A . In order to estimate $\|A^d\|_1$, NORMEST1 computes repeatedly $Y := A^d X$, where $X \in \mathbb{C}^{n \times t}$, with $t \ll n$. If the matrix multiplications are performed from right to left, evaluating Y requires $2dtn^2$ flops, but if some of the powers of A are available, the factor d can be reduced to as little as $\log_2 d$. We illustrate the strategy to perform this cost reduction in the function EVALPOWVECDIAG, which evaluates $A^d X$ using the powers of A^2 stored in \mathcal{A} . We use the two variables \tilde{d} and ℓ , initialized to d and $\text{length}(\mathcal{A})$, respectively, to keep track of the state of the computation. The function repeatedly multiplies X by $\mathcal{A}_\ell = A^{2\ell}$ for $t = \lfloor \tilde{d}/(2\ell) \rfloor$ times, that is, until \tilde{d} becomes smaller than 2ℓ . At this point, the algorithm updates \tilde{d} and ℓ , setting the former to the number of matrix multiplications left to perform, $\tilde{d} - t$, and the latter to the largest integer smaller than the new value of \tilde{d} . Since \mathcal{A} contains powers of A^2 rather than A , an additional multiplication by A is necessary for odd d .

Fragment 5.4: Modified Paterson–Stockmeyer algorithm.

```

1 function EVALPOLYPS( $s \in \mathbb{N}, \beta \in \mathbb{C}^t, v \in \mathbb{N}$ )
  ▷ Evaluate  $\sum_{\ell=0}^t \beta_\ell (2^{-s} A)^\ell$  using elements of  $\mathcal{A}$ .
2  $\ell \leftarrow \text{length}(\mathcal{A})$ 
3  $v \leftarrow \lfloor t/v \rfloor$ 
4 for  $i \leftarrow \ell + 1$  to  $v$  do
5    $\mathcal{A}_i \leftarrow \mathcal{A}_{i-1} \mathcal{A}_1$ 
6  $Y \leftarrow \sum_{j=0}^{m-vv} \beta_{vv+j} 2^{-sj} \mathcal{A}_j$ 
7 for  $i \leftarrow v - 1$  down to 0 do
8    $Y \leftarrow Y 2^{-sv} \mathcal{A}_s + \sum_{j=0}^{s-1} \beta_{vi+j} 2^{-sj} \mathcal{A}_j$ 
9 return  $Y$ 

```

This algorithm requires $(\min(C_{\lfloor \sqrt{m_i} \rfloor}(\mathfrak{m}_i), C_{\lceil \sqrt{m_i} \rceil}(\mathfrak{m}_i)) + s)n^3$ flops in precision u , for evaluating $r_{\mathfrak{m}_i}(A)$ and performing the final squaring phase, and $(2\sqrt{m_i} + \frac{2}{3})in^3$ flops in precision u_{bnd} for evaluating and factorizing $q_{\mathfrak{m}_i}(2^{-s}A)$, in order to check whether the bound (5.18) is satisfied.

5.3.2 Taylor approximants

Truncated Taylor series are appealing Padé approximants to use in conjunction with bound (5.15), as the property that $q_{k0}(x) = 1$ enables us to eliminate the computation of $q_{\mathfrak{m}_i}(2^{-s}A)$, the most expensive term to evaluate in (5.18), and thus obtain substantial computational savings. Even though for truncated Taylor series there is no need to evaluate the approximant when evaluating the bound, the function EVALBOUNDTAYL updates the array \mathcal{A} , which in this case stores powers of A rather than A^2 . In fact, these powers can be used to reduce the cost of estimating $\|A^d\|_1$ as well as $\|e^A\|_1$. The elements of \mathcal{A} are estimated by means of NORMEST1, and the action of the powers of A on a vector is computed by means of the function EVALPOWVECTAYL in Fragment 5.5, which uses the elements in \mathcal{A} analogously to EVALPOWVECDIAG.

For ψ in the bound (5.18) one can use a lower bound on the 1-norm of e^A . The inequality $\|e^A\|_1 \geq e^{-\|A\|_1}$ [20, Thm. 10.10] can be exploited at no extra cost, but

Fragment 5.5: Auxiliary functions for truncated Taylor series t_m .

```

1 function EVALBOUNDTAYL( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, s \in \mathbb{N}$ )
   $\triangleright$  Check (5.18) for  $t_m$ .
2 for  $i \leftarrow \text{length}(\mathcal{A}) + 1$  to  $\lceil \sqrt{m} \rceil$  do
3    $\mathcal{A}_i \leftarrow \mathcal{A}_{i-1} A_1$ 
4  $\alpha_{\min} \leftarrow \text{OPTALPHATAYL}(A, m, \alpha_{\min})$ 
5  $\delta \leftarrow |e^{2^{-s}\alpha_{\min}} - p_m(2^{-s}\alpha_{\min})|$ 
6  $\psi \leftarrow \text{ESTIMATENORMEXP}(s)$ 
7 return  $\delta, \psi, 1$ 

```

```

8 function EVALPADETAYL( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, s \in \mathbb{N}$ )
   $\triangleright$  Evaluate  $t_m(2^{-s}A)$ .
9  $\beta \leftarrow \left[ \frac{1}{i!} \right]_{i=0}^m$ 
10 return EVALPOLYPS( $s, \beta, \sqrt{m}$ )

```

```

11 function OPTALPHATAYL( $A \in \mathbb{C}^{n \times n}, m \in \mathbb{N}, \alpha_{\min} \in \mathbb{R}^+$ )
   $\triangleright$  Compute  $\alpha_{\min}$ .
12  $d \leftarrow \left\lfloor \frac{1+\sqrt{5+4m}}{2} \right\rfloor$ 
13 if  $\gamma_d = -\infty$  then
14    $\gamma_d \leftarrow \text{NORMEST1}(\lambda x. \text{EVALPOWVECTAYL}(d, x))^{1/d}$ 
15 if  $\gamma_{d+1} = -\infty$  then
16    $\gamma_{d+1} \leftarrow \text{NORMEST1}(\lambda x. \text{EVALPOWVECTAYL}(d+1, x))^{1/(d+1)}$ 
17 return  $\min\{\max\{\gamma_d, \gamma_{d+1}\}, \alpha_{\min}\}$ 

```

```

18 function EVALPOWVECTAYL( $d \in \mathbb{N}, X \in \mathbb{C}^{n \times t}$ )
   $\triangleright$  Compute  $A^d X$  using elements in  $\mathcal{A}$ .
19  $\ell \leftarrow \text{length}(\mathcal{A})$ 
20 while  $d > 0$  do
21   for  $i \leftarrow 1$  to  $\lfloor p/\ell \rfloor$  do
22      $X \leftarrow \mathcal{A}_\ell X$ 
23      $d \leftarrow d \bmod \ell$ 
24      $\ell \leftarrow \min\{\ell - 1, d\}$ 
25 return  $X$ 

```

```

26 function ESTIMATENORMEXP( $s \in \mathbb{N}$ )
   $\triangleright$  Estimate  $\|e^A\|_1$  using elements in  $\mathcal{A}$ .
27  $Z \leftarrow \sum_{i=0}^{\text{length}(\mathcal{A})} \frac{\mathcal{A}_i}{2^{si} i!}$ 
28 return  $\text{NORMEST1}(\lambda x. Zx)$ 

```

being typically not very sharp can potentially lead to unnecessary computation. Van Loan [45] suggests the bound

$$\|e^A\|_1 \geq e^{\lambda^*}, \quad \lambda^* = \max_{\lambda \in \sigma(A)} \operatorname{Re} \lambda,$$

which is typically tighter than the previous bound, and always so when $\lambda^* > 0$. Estimating λ^* , however, requires either the eigendecomposition of A , or the solution of a family of shifted linear systems [35], and both solutions might be unpractical in that they require $O(n^3)$ flops for dense matrices. A practical estimate that can be computed with only $O(n^2)$ extra cost is provided by the function `ESTIMATENORMEXP` in Fragment 5.5, which relies on the approximation

$$e^{2^{-s}A} \approx \sum_{i=0}^{\ell} \frac{(2^{-s}A)^i}{i!} = \sum_{i=0}^{\ell} \frac{2^{-si} \mathcal{A}_i}{i!} =: \tilde{\zeta}_{\ell}^s, \quad \ell = \operatorname{length}(\mathcal{A}).$$

If only the elements of \mathcal{A} already computed on line 2-3 of `EVALBOUNDTAYL` are used, computing this estimate requires only $2\ell n^2$ flops. Additional savings can be gained by noting that $\tilde{\zeta}_{\ell+1}^s$ can be obtained from $\tilde{\zeta}_{\ell}^s$ with only one matrix scaling and one matrix sum, and the full cost of $2\ell n^2$ flops need not be paid as long as s does not change.

Overall, this algorithm requires $(2\sqrt{m_i} + s)n^3$ floating-point operations.

5.3.3 Schur–Padé variants

If A is normal ($A^*A = AA^*$) and a multiprecision implementation of the QR algorithm is available, diagonalization in higher precision is another approach for computing e^A . More generally, a (real) Schur decomposition can be used: $A = QTQ^*$, where $T, Q \in \mathbb{C}^{n \times n}$ are, respectively, upper triangular and unitary if A has complex entries and upper quasi-triangular and orthogonal if A has real entries. Then $e^A = Qe^TQ^*$. In our experiments, we consider a Schur–Padé approach that computes the Schur decomposition of the input matrix and exploits Algorithm 5.1 to compute the exponential of its triangular factor.

Overall, this algorithm requires $\left(28 + \left(\min\{C_{\lfloor\sqrt{m_i}\rfloor}(m_i), C_{\lceil\sqrt{m_i}\rceil}(m_i)\} + s\right)/3\right)n^3$ and $(28 + (2\sqrt{m_i} + s)/3)n^3$ flops for diagonal Padé approximants and truncated Taylor series, respectively.

5.4 NUMERICAL EXPERIMENTS

We now test the algorithm derived in section 5.3 and compare it with two existing codes for computing the matrix exponential in arbitrary precision floating point arithmetic. We consider two test sets: \mathcal{H} , which contains 35 Hermitian matrices, and \mathcal{N} , which consists of 97 non-Hermitian matrices. These matrices, of size ranging between 2 and 1000, are taken from the literature of the matrix exponential, from a collection of benchmark problems for the burnup equations [30], [46], and from the MATLAB gallery function. The experiments were performed using the 64-bit (glna64) version of MATLAB 9.5 (R2018b Update 3) on a machine equipped with an Intel I5-3570 processor running at 3.40GHz and with 8GB of RAM. The code uses the Multiprecision Computing Toolbox (version 4.4.7.12739) [38], which provides the class `mp` to represent arbitrary precision floating-point numbers and overloads all the MATLAB functions we need in our implementations. We note that this toolbox allows the user to specify the number of *decimal* digits of working precision, but not the number of bits in the fraction of its binary representation, thus, in this section, whenever we refer to d (decimal) digits of precision, we mean that the working precision is set using the command `mp.Digits(d)`. The MATLAB code that runs the tests in this section is available on GitHub.¹

In our experiments, we compare the following codes.

- `expm`, the built-in function `expm` of MATLAB, which implements the algorithm of Al-Mohy and Higham [2], and is intended for double precision only.
- `exp_mct`, the `expm` function provided by the Multiprecision Computing Toolbox.

¹ <https://github.com/mfasi/mpexpm>.

- `exp_otf`, the algorithm by Caliari and Zivcovich [7], a shifted scaling and squaring method based on truncated Taylor series. The matrix is shifted by $\text{tr}(A)/n$, and (5.2) is replaced by

$$e^A = \left(e^{(2^s + 2^t)^{-1}A} \right)^{2^s + 2^t}, \quad s \in \mathbb{N}, \quad t \in \mathbb{N} \cup \{-\infty\}.$$

The order of the approximant is chosen by estimating at runtime a bound on the backward error of the approximant in exact arithmetic.

- `exp_d`, an implementation of Algorithm 5.1 with `use_taylor = false`.
- `exp_t`, an implementation of Algorithm 5.1 with `use_taylor = true`.
- `exp_sp_d`, an implementation of the Schur–Padé approach discussed in section 5.3.3 using Algorithm 5.1, with `use_taylor = false`, for the triangular Schur factor.
- `exp_sp_t`, an implementation of the Schur–Padé approach discussed in section 5.3.3 using Algorithm 5.1, with `use_taylor = true`, for the triangular Schur factor.

In our implementations of Algorithm 5.1, we set $u_{\text{bnd}} = 2^{-53}$, $\varepsilon = u$, $s_{\text{max}} = 100$, and the entries of \mathfrak{m} to the elements of (5.8) smaller than 1000 and those of (5.9) smaller than 400, for truncated Taylor series and diagonal Padé approximant, respectively.

We do not include the function `expm` provided by the Symbolic Math Toolbox in our tests since, for precision higher than double, it appears to be extending the precision internally, using a number of extra digits that increases with the working precision. As a result, the forward error of the algorithm is typically several orders of magnitude smaller than machine epsilon, when computing with 20 or more digits, but tends to be larger than the unit roundoff u , for $u \approx 10^{-20}$ or larger. We note that the accuracy drops for matrices that are nondiagonalizable, singular, or have ill-conditioned eigenvectors. Moreover, the Symbolic Math Toolbox implementation is rather slow on moderately large matrices ($n \gtrsim 50$, say), which makes this code unsuitable for extended testing.

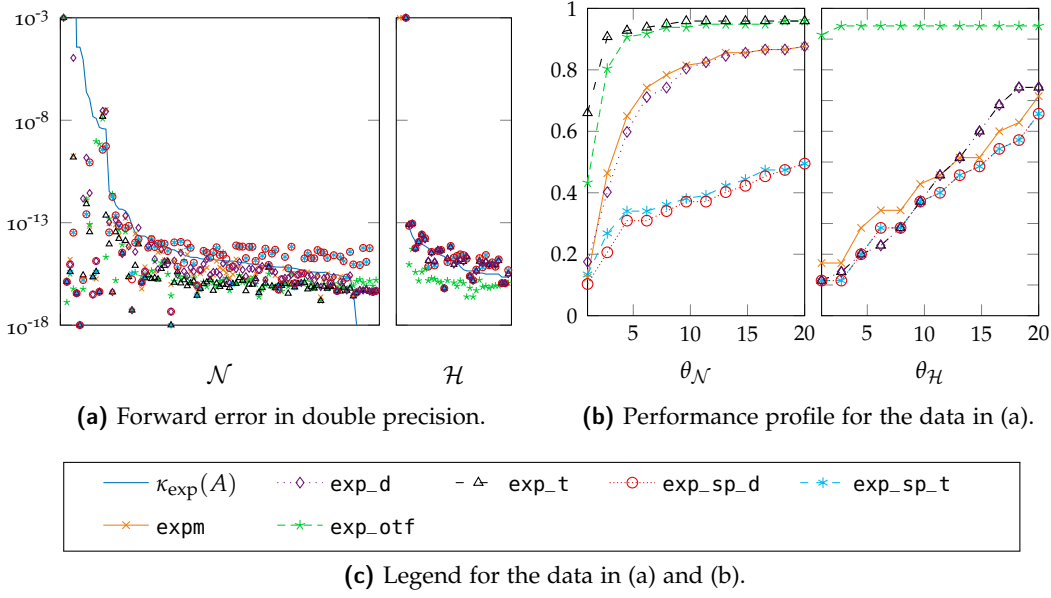


Figure 5.1: Left: forward error of expm and exp_sp_d, exp_sp_t, exp_d, and exp_t running in IEEE double precision arithmetic on the matrices in \mathcal{N} and \mathcal{H} , ordered by decreasing value of $\kappa_{\text{exp}}(A)$. Right: performance profile for the matrices in \mathcal{N} .

In our tests, we assess the accuracy of the solution \tilde{X} computed running with d digits of precision by means of the relative forward error $\|X - \tilde{X}\|_1 / \|X\|_1$, where X is a reference solution computed using exp_mct with $2d$ significant digits. Since the magnitude of forward errors depends not only on the working precision but also on the conditioning of the problem, in our plots we compare the forward error of the algorithms with $\kappa_{\text{exp}}(A)u$, where $\kappa_{\text{exp}}(A)$ is the 1-norm condition number [20, Chap. 4] of the matrix exponential of A (see (5.4)). We estimate it in double precision using the funm_condest1 function provided by the Matrix Function Toolbox [18] on expm.

When plotting the forward error, we choose the limits of the y-axis in order to show the area where most of the data points lie, and move the outliers to the closest edge of the box containing the plot. In several cases, we present our experimental results with the aid of performance profiles [10], and adopt the technique of Dingle and Higham [9] to rescale values smaller than u .

5.4.1 Comparison with `expm` in double precision

In this experiment, we compare the performance of `exp_d`, `exp_t`, `exp_sp_d`, and `exp_sp_t` running in IEEE double precision, with that of `expm` and `exp_otf`. The purpose of this experiment is to verify that the new algorithms are broadly competitive with `expm`; since `expm` is optimized for double precision, we do not expect them to be as efficient.

Figure 5.1a compares the forward error of the algorithms on the matrices in our test sets, sorted by decreasing condition number. The performance profile in Figure 5.1b presents the results on a by-algorithm rather than by-matrix basis: for a given method, the height of the line at $\theta_{\mathcal{D}} = \theta_0$ represents the fraction of matrices in \mathcal{D} for which the relative forward error is within a factor θ_0 of the error of the algorithm that delivers the most accurate result for that matrix.

For Hermitian matrices, the error plot clearly shows that while `exp_otf`, `exp_t`, and `exp_d` all provide forward error well below $\kappa_{\text{exp}}(A)u$, and the corresponding performance profiles indicate that `exp_otf` is consistently the most accurate on that test set. The performance of `expm` is the same as that of `exp_sp_d` because both implementations reduce to the evaluation of the scalar exponential at the eigenvalues of A when A is Hermitian.

For the matrices in \mathcal{N} , the errors of `expm`, `exp_otf`, `exp_d`, and `exp_t` are approximately bounded by $\kappa_{\text{exp}}(A)u$, with the algorithms based on truncated Taylor series being overall more accurate than those based on diagonal Padé approximants. The algorithms based on the Schur decomposition of A tend to give somewhat larger errors, with the result that the performance profile curves are the least favorable.

5.4.2 Behavior in higher precision

Now we investigate the accuracy of our algorithm in higher precision and compare it with `exp_mct`, the built-in function of the Multiprecision Computing Toolbox, and `exp_otf`. The left column of Figure 5.2 compares the quantity $\kappa_{\text{exp}}(A)u$ with the

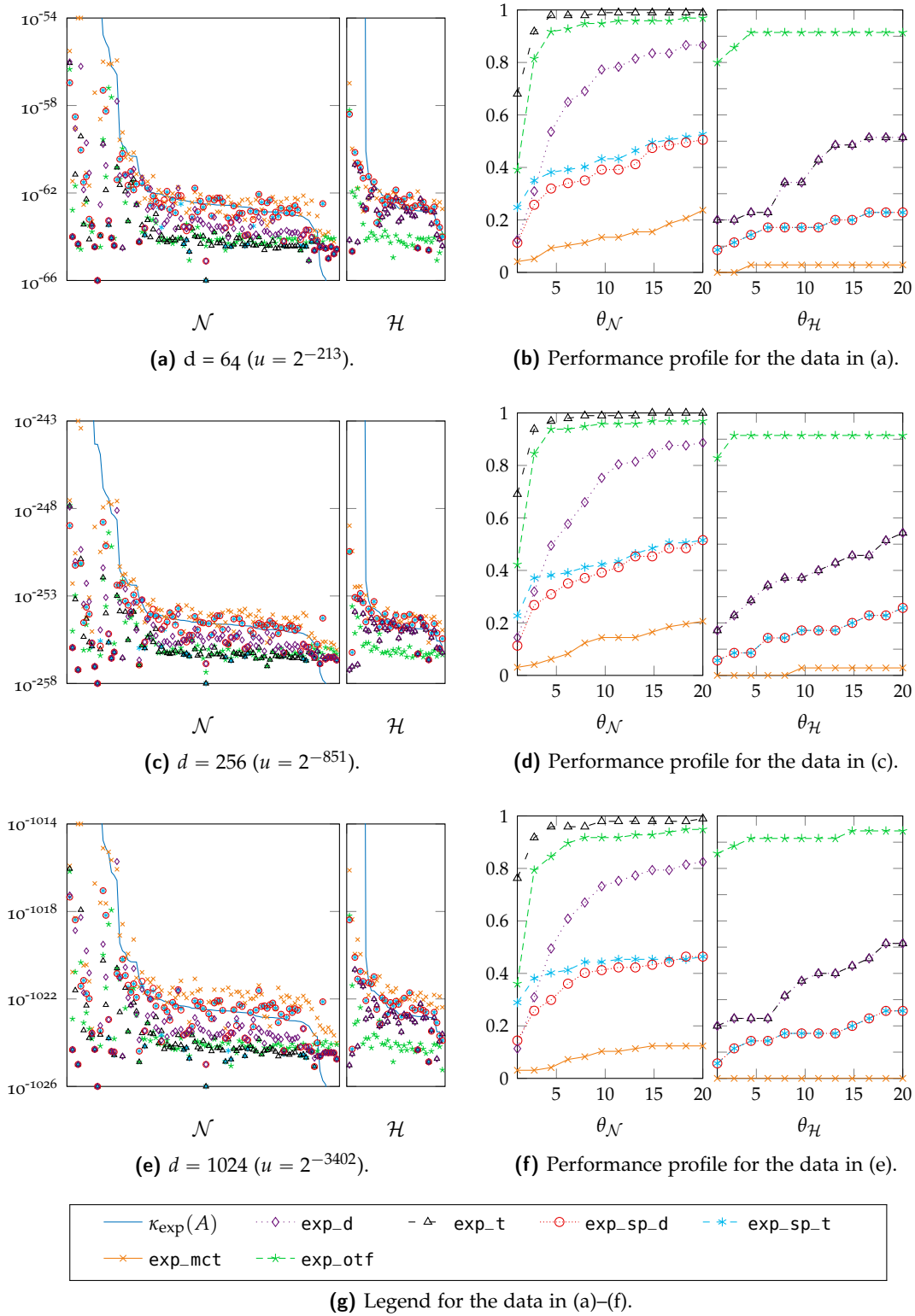


Figure 5.2: Left: forward error of the methods on the matrices in the test sets.
Right: Corresponding performance profiles for the matrices in \mathcal{N} .

forward errors of `exp_mct`, `exp_otf`, `exp_sp_d`, `exp_d`, `exp_sp_t`, and `exp_t` running with about 64, 256, and 1024 decimal significant digits on the matrices in our test sets sorted by decreasing condition number $\kappa_{\text{exp}}(A)$. The right-hand column presents the data for the matrices in \mathcal{N} by means of performance profiles.

The results show that, as for double precision, transformation-free algorithms tend to produce a more accurate approximation of e^A than those based on the Schur decomposition of A . The code `exp_mct` typically delivers the least accurate results, with a forward error usually larger than $\kappa_{\text{exp}}(A)u$.

On the Hermitian test matrices, `exp_otf` is typically the most accurate algorithm, having a surprising ability to produce errors much less than $\kappa_{\text{exp}}(A)u$. On the set of non-Hermitian matrices, `exp_otf` and `exp_t` are the most accurate algorithms, with `exp_t` having the superior performance profile. On this test set, the least accurate of our versions of Algorithm 5.1 is `exp_sp_t`, closely followed by `exp_sp_d`; the forward error of both, despite being typically smaller than that of `exp_mct`, is often slightly larger than $\kappa_{\text{exp}}(A)u$, especially for the better conditioned of our test matrices. Finally, `exp_d` performs better than Schur-based variants, but is slightly less accurate than `exp_otf` and `exp_t` on most of the matrices in this test set. We remark, however, that its forward error is typically smaller than $\kappa_{\text{exp}}(A)u$ when that of the other two Taylor-based algorithms is.

We note that the tests of `exp_otf` in [7] have precision target 10^{-25} or larger, which is between double and quadruple precision. This experiment shows that `exp_otf` maintains its good accuracy up to much higher precisions.

5.4.3 Code profiling

In Table 5.1, we compare the execution time of our MATLAB implementations of `exp_t` and `exp_d`, running in quadruple precision ($d = 34$ and unit roundoff $u = 2^{-113}$), on the matrices

```
A = 1000 * triu(ones(n),1);
```

Table 5.1: Execution time breakdown of `exp_t` and `exp_d`, run in quadruple precision on three matrices of increasing size. The table reports, for each algorithm, the number of squarings (M_{sqr}), the number of matrix multiplications needed to evaluate the Padé approximant (M_{eval}), the total execution time in seconds (T_{tot}), and the percentage of time spent evaluating the scalar bound (T_{bnd}), evaluating the Padé approximant (T_{eval}), and performing the final squaring step (T_{sqr}).

	n	exp_t						exp_d					
		M_{sqr}	M_{eval}	T_{bnd}	T_{eval}	T_{sqr}	T_{tot}	M_{sqr}	M_{eval}	T_{bnd}	T_{eval}	T_{sqr}	T_{tot}
A	10	5	13	20%	39%	41%	0.1	7	8	60%	12%	28%	0.2
	20	6	13	13%	31%	56%	0.1	8	8	41%	14%	45%	0.2
	50	7	15	7%	26%	67%	0.3	9	9	37%	17%	46%	0.6
	100	8	15	4%	29%	67%	0.8	10	9	35%	23%	42%	1.6
	200	9	15	4%	38%	58%	2.7	11	9	42%	29%	29%	6.4
	500	10	16	3%	45%	51%	27.5	12	9	39%	33%	28%	59.2
	1000	11	16	3%	45%	52%	197.1	12	10	33%	38%	29%	365.3
B	10	1	11	24%	61%	15%	0.0	1	9	69%	24%	8%	0.1
	20	2	12	14%	58%	28%	0.1	2	9	54%	27%	18%	0.1
	50	3	13	8%	44%	48%	0.2	3	10	45%	28%	27%	0.4
	100	4	13	5%	34%	61%	0.4	5	9	41%	28%	31%	1.1
	200	5	13	4%	33%	63%	1.3	6	9	46%	31%	24%	4.4
	500	6	14	3%	30%	67%	11.2	6	10	39%	39%	22%	37.6
	1000	7	14	2%	24%	74%	69.0	7	10	35%	39%	26%	238.7
C	10	2	11	35%	64%	1%	0.0	1	10	76%	24%	0%	0.1
	20	2	12	23%	76%	1%	0.1	2	9	69%	30%	1%	0.1
	50	3	12	16%	75%	9%	0.1	2	10	53%	45%	2%	0.3
	100	3	13	8%	78%	14%	0.5	2	11	40%	56%	3%	1.4
	200	3	14	6%	79%	15%	3.6	3	10	34%	58%	8%	6.5
	500	4	13	5%	73%	21%	52.4	4	9	21%	63%	16%	69.5
	1000	4	14	5%	74%	21%	434.0	5	9	16%	62%	22%	505.2

```
B = zeros(n); B(n+1:n+1:n^2) = 1:n-1; % Upper bidiagonal.
```

```
C = gallery('lotkin', n);
```

where n ranges between 10 and 1000. The first two matrices are strictly upper triangular, thus singular, but the 1-norm of A is much larger than that of B . The full matrix C is nonsingular. For each matrix, we report the overall execution time in seconds of the two implementations (T_{tot}), specifying how much time is spent evaluating the scalar bound (T_{bnd}), evaluating the approximant on the input matrix (T_{eval}), and performing the final squaring stage (T_{sqr}).

For `exp_d`, the evaluation of the forward bound on the truncation error of the Padé approximant is typically the most expensive operation for small matrices, and even when the size of the matrix increases the cost of this operation remains nonnegligible, as the estimation of the quantity $\|(q_{km}(A))^{-1}\|_1$, which appears in the bound (5.18), has cubic dependence on the size of the input matrix. For `exp_t`, on the other hand,

T_{bnd} depends only quadratically on n , and tends to become relatively small for matrices of size larger than 100.

5.5 CONCLUSIONS

State-of-the-art scaling and squaring algorithms for computing the matrix exponential typically rely on a backward error analysis in order to scale the matrix and then select an appropriate Padé approximant to meet a given accuracy threshold. The result of this analysis is a small set of precision-dependent constants that are hard to compute but can be easily stored for subsequent use. This approach is not viable in multiprecision environments, where the working precision is known only at run time and is an input argument of the algorithm rather than a property of the underlying floating-point number system. For truncated Taylor series, it is possible to estimate on-the-fly a bound on the backward error of the approximation [7], but this technique relies on a conjecture and does not readily generalize to other Padé approximants.

We have developed a new algorithm (Algorithm 5.1) based on Padé approximation for computing the matrix exponential in arbitrary precision. In particular, we derived a new representation for the truncation error of a general $[k/m]$ Padé approximant and showed how it can be used to compute practical error bounds for truncated Taylor series and diagonal Padé approximants. In the first case, the bound is cheap to compute, requiring only $O(n^2)$ flops. For diagonal Padé approximants the new bound requires $O(n^3)$ low-precision flops, but if Conjecture 5.2 turns out to be true then this cost will reduce to $O(n^2)$ flops.

According to our experimental results, Algorithm 5.1 in transformation-free form using truncated Taylor series is the best option for computing the matrix exponential in precisions higher than double. In particular, the algorithm is the most accurate on non-Hermitian matrices. For Hermitian matrices, it is natural to compute e^A via a spectral decomposition (as does the MATLAB function `expm`), but an interesting finding is that on our Hermitian test matrices this approach (to which `exp_sp_d` and

`exp_sp_t` effectively reduce) is less accurate than Algorithm 5.1 and the algorithm of [7].

When developing the algorithm and testing it experimentally we focused on precisions higher than double. We believe that a different approach is needed to address the computation of the matrix exponential in low precision arithmetic, such as IEEE half precision. Indeed, due to its very limited range this number format is prone to underflow and overflow, which makes accuracy and robustness difficult to achieve. How to handle these challenges will be the subject of future work.

ACKNOWLEDGEMENTS

We thank the editor and the anonymous referees for their comments that helped us improve the presentation of the paper.

BIBLIOGRAPHY

- [1] A. H. AL-MOHY, *A truncated Taylor series algorithm for computing the action of trigonometric and hyperbolic matrix functions*, SIAM J. Sci. Comput., 40 (2018), pp. A1696–A1713.
- [2] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989.
- [3] —, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169.
- [4] A. H. AL-MOHY, N. J. HIGHAM, AND S. D. RELTON, *New algorithms for computing the matrix sine and cosine separately or simultaneously*, SIAM J. Sci. Comput., 37 (2015), pp. A456–A487.

- [5] M. APRAHAMIAN AND N. J. HIGHAM, *Matrix inverse trigonometric and inverse hyperbolic functions: Theory and algorithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1453–1477.
- [6] J. BEZANSON, A. EDELMAN, S. KARPINSKI, AND V. B. SHAH, *Julia: A fresh approach to numerical computing*, SIAM Rev., 59 (2017), pp. 65–98.
- [7] M. CALIARI AND F. ZIVCOVICH, *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*, J. Comput. Appl. Math, 346 (2019), pp. 532–548.
- [8] E. DEFEZ, J. SASTRE, J. IBÁÑEZ, AND J. PEINADO, *Solving engineering models using hyperbolic matrix functions*, Appl. Math. Model., 40 (2016), pp. 2837–2844.
- [9] N. J. DINGLE AND N. J. HIGHAM, *Reducing the influence of tiny normwise relative errors on performance profiles*, ACM Trans. Math. Software, 39 (2013), pp. 24:1–24:11.
- [10] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Programming, 91 (2002), pp. 201–213.
- [11] F. FAÀ DE BRUNO, *Note sur une nouvelle formule de calcul différentiel*, Quarterly J. Pure Appl. Math, 1 (1857), p. 12.
- [12] F. FAÀ DI BRUNO, *Sullo sviluppo delle funzioni*, Ann. Sci. Mat. Fis., 6 (1855), pp. 479–480.
- [13] M. FASI, *Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions*, Linear Algebra Appl., 574 (2019), p. 182–200.
- [14] M. FASI AND N. J. HIGHAM, *Multiprecision algorithms for computing the matrix logarithm*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 472–491.
- [15] M. FASI, N. J. HIGHAM, AND B. IANNAZZO, *An algorithm for the matrix Lambert W function*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 669–685.
- [16] W. GAUTSCHI, *Numerical Analysis*, Birkhäuser, 2011.

- [17] G. I. HARGREAVES AND N. J. HIGHAM, *Efficient algorithms for the matrix cosine and sine*, Numer. Algorithms, 40 (2005), pp. 383–400.
- [18] N. J. HIGHAM, *The Matrix Function Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mfttoolbox>.
- [19] —, *The scaling and squaring method for the matrix exponential revisited*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 1179–1193.
- [20] —, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [21] —, *The scaling and squaring method for the matrix exponential revisited*, SIAM Rev., 51 (2009), pp. 747–764.
- [22] N. J. HIGHAM AND P. KANDOLF, *Computing the action of trigonometric and hyperbolic matrix functions*, SIAM J. Sci. Comput., 39 (2017), pp. A613–A627.
- [23] N. J. HIGHAM AND L. LIN, *A Schur–Padé algorithm for fractional powers of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1056–1078.
- [24] —, *An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360.
- [25] N. J. HIGHAM AND M. I. SMITH, *Computing the matrix cosine*, Numer. Algorithms, 34 (2003), pp. 13–26.
- [26] N. J. HIGHAM AND F. TISSEUR, *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201.
- [27] F. JOHANSSON, *Arb: Efficient arbitrary-precision midpoint-radius interval arithmetic*, IEEE Trans. Comput., 66 (2017), p. 1281–1292.
- [28] F. JOHANSSON ET AL., *Mpmath: A Python library for arbitrary-precision floating-point arithmetic*. <http://mpmath.org>.

- [29] W. P. JOHNSON, *The curious history of Faà di Bruno's formula*, Amer. Math. Monthly, 109 (2002), pp. 217–234.
- [30] D. LAGO AND F. RAHNEMA, *Development of a set of benchmark problems to verify numerical methods for solving burnup equations*, Ann. Nucl. Energy, 99 (2017), pp. 266–271.
- [31] E. N. LAGUERRE, *Le calcul des systèmes linéaires, extrait d'une lettre adressé à M. Hermite*, in *Oeuvres de Laguerre*, C. Hermite, H. Poincaré, and E. Rouché, eds., vol. 1, Gauthier-Villars, Paris, 1898, pp. 221–267. The article is dated 1867 and is “Extrait du Journal de l'École Polytechnique, LXII^e Cahier”.
- [32] J. D. LAWSON, *Generalized Runge–Kutta processes for stable systems with large lipschitz constants*, SIAM J. Numer. Anal., 4 (1967), pp. 372–380.
- [33] *Maple*. Waterloo Maple Inc., Waterloo, Ontario, Canada. <http://www.maplesoft.com>.
- [34] *Mathematica*. Wolfram Research, Inc., Champaign, IL, USA. <http://www.wolfram.com>.
- [35] K. MEERBERGEN, A. SPENCE, AND D. ROOSE, *Shift-invert and Cayley transforms for detection of rightmost eigenvalues of nonsymmetric matrices*, BIT, 34 (1994), pp. 409–423.
- [36] C. B. MOLER AND C. F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Rev., 20 (1978), pp. 801–836.
- [37] ———, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.
- [38] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. <http://www.advanpix.com>.
- [39] P. NADUKANDI AND N. J. HIGHAM, *Computing the wave-kernel matrix functions*, SIAM J. Sci. Comput., 40 (2018), pp. A4060–A4082.
- [40] M. S. PATERSON AND L. J. STOCKMEYER, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66.

- [41] M. PUSA, *Rational approximations to the matrix exponential in burnup calculations*, Nuclear Science and Engineering, 169 (2011), p. 155–167.
- [42] THE SAGE DEVELOPERS, *Sage Mathematics Software*. <http://www.sagemath.org>.
- [43] J. SASTRE, J. IBÁÑEZ, P. RUIZ, AND E. DEFEZ, *Accurate and efficient matrix exponential computation*, Internat. J. Comput. Math., 91 (2013), p. 97–112.
- [44] *Symbolic Math Toolbox*. The MathWorks, Inc., Natick, MA, USA. <http://www.mathworks.co.uk/products/symbolic/>.
- [45] C. F. VAN LOAN, *The sensitivity of the matrix exponential*, SIAM J. Numer. Anal., 14 (1977), pp. 971–981.
- [46] S. ZHAO, *Matrix exponential approximation for burnup equation*, M.Sc. thesis, The University of Manchester, 2017.
- [47] P. ZIMMERMANN, A. CASAMAYOU, N. COHEN, G. CONNAN, T. DUMONT, L. FOUSSE, F. MALTEY, M. MEULIEN, M. MEZZAROBBA, C. PERNET, N. M. THIÉRY, E. BRAY, J. CREMONA, M. FORETS, A. GHITZA, AND H. THOMAS, *Computational Mathematics with SageMath*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2018.

6

MULTIPRECISION ALGORITHMS FOR
COMPUTING THE MATRIX LOGARITHM

Abstract. Two algorithms are developed for computing the matrix logarithm in floating point arithmetic of any specified precision. The backward error-based approach used in the state of the art inverse scaling and squaring algorithms does not conveniently extend to a multiprecision environment, so instead we choose algorithmic parameters based on a forward error bound. We derive a new forward error bound for Padé approximants that for highly nonnormal matrices can be much smaller than the classical bound of Kenney and Laub. One of our algorithms exploits a Schur decomposition while the other is transformation-free and uses only the computational kernels of matrix multiplication and the solution of multiple right-hand side linear systems. For double precision computations the algorithms are competitive with the state of the art algorithm of Al-Mohy, Higham, and Relton implemented in `logm` in MATLAB. They are intended for computing environments providing multiprecision floating point arithmetic, such as Julia, MATLAB via the Symbolic Math Toolbox or the Multiprecision Computing Toolbox, or Python with the `mpmath` or `SymPy` packages. We show experimentally that the algorithms behave in a forward stable manner over a wide range of precisions, unlike existing alternatives.

Keywords: multiprecision arithmetic, matrix logarithm, principal logarithm, inverse scaling and squaring method, Fréchet derivative, Padé approximation, Taylor approximation, forward error analysis, MATLAB, `logm`.

2010 MSC: 15A60, 65F30, 65F60.

6.1 INTRODUCTION

Let $A \in \mathbb{C}^{n \times n}$ be nonsingular with no nonpositive real eigenvalues. Any matrix $X \in \mathbb{C}^{n \times n}$ satisfying the matrix equation

$$X = e^A \tag{6.1}$$

is a matrix logarithm of A . This equation has infinitely many solutions, but in applications one is typically interested in the principal matrix logarithm, denoted by $\log A$, which is the unique matrix X whose eigenvalues have imaginary part strictly between $-\pi$ and π . This choice is the most natural in that it guarantees that if the matrix is real then so is its logarithm and that if the matrix has positive real spectrum then so does its logarithm.

More generally, the unique matrix satisfying (6.1) having spectrum in the complex strip

$$L_k = \{z \in \mathbb{C} \mid (k-1)\pi < \operatorname{Im} z < (k+1)\pi\}, \quad k \in \mathbb{Z},$$

is called the k th branch of the matrix logarithm, and is denoted by $\log_k A$. The choice $k = 0$ yields the principal logarithm $\log A$. From a computational viewpoint, being able to approximate $\log A$ is enough to determine the value of $\log_k A$ for any $k \in \mathbb{Z}$, in view of the identity $\log_k A = \log A + 2k\pi iI$.

The aim of this work is to develop an algorithm for $\log A$ that is valid for floating point arithmetic of any given precision. A new algorithm is needed because the state of the art algorithm of Al-Mohy, Higham, and Relton [3], [4], which is implemented in the MATLAB function `logm`, is designed specifically for IEEE double precision arithmetic. Indeed most available software for the matrix logarithm has this limitation of being precision-specific [32]. Applications of the new algorithm will be in both low and high precision contexts. For example, both the matrix logarithm [36] and low precision computations (perhaps 32-bit single precision, or 16-bit half precision) [15],

[26], have been recently used in machine learning, and a combination of the two is potentially of interest.

The need for high precision arises in several contexts. For instance, in order to estimate the forward error of a double precision algorithm for the matrix logarithm a reference solution computed at quadruple or higher precision is usually needed. Estimating the backward error of a double precision algorithm for the matrix exponential also requires the ability to evaluate $\log A$ at high precision. Let $X = e^A$ and let \tilde{X} be a solution computed by a double precision algorithm for the matrix exponential. If the spectrum of A lies inside L_k , so that $A = \log_k X$, then the backward error of \tilde{X} is naturally defined as the matrix ΔA such that

$$\log_k \tilde{X} = A + \Delta A, \quad (6.2)$$

because then $\tilde{X} = e^{A+\Delta A}$, and the normwise relative backward error is

$$\frac{\|\Delta A\|}{\|A\|} = \|\log \tilde{X} - A\|/\|A\|.$$

A multiprecision algorithm for the matrix logarithm is needed in a variety of languages and libraries that attempt to provide multiprecision implementations of a wide range of functions with both scalar and matrix arguments. The Julia language [8] and Python's SymPy [44], [49] and mpmath [39] libraries currently lack such an algorithm, and we will show that the algorithms proposed here improve upon those in version 7.1 of the Symbolic Toolbox for MATLAB [48] and version 4.3.2 of the Multiprecision Computing Toolbox [45].

The algorithm of Al-Mohy, Higham, and Relton used by `logm` is the culmination of a line of inverse scaling and squaring algorithms that originates with Kenney and Laub [40] for matrices and goes back to Briggs [11] in the 17th century in the scalar case. In essence, the algorithm performs three steps. Initially, it takes square roots of A , s of them, say, until the spectrum of $A^{1/2^s} - I$ is within the unit disk, which is the largest disk centered at the origin in which the principal branch of $\log(1+x)$ is analytic and its Padé approximants are therefore well defined. Then it selects a Padé

approximant $r_{km}(x) := p_{km}(x)/q_{km}(x)$ to $\log(1+x)$ of suitable degree $[k/m]$, evaluates the rational matrix function $r_{km}(X) = p_{km}(X) q_{km}(X)^{-1}$ at $X = A^{1/2^s} - I$, and finally reverts the square roots to form the approximation $\log A \approx 2^s r_{km}(A^{1/2^s} - I)$. The algorithm is based on a backward error analysis and uses pre-computed constants that specify how small a normwise measure of $A^{1/2^s} - I$ must be in order for a given diagonal Padé approximant r_{mm} to deliver a backward stable evaluation in IEEE double precision arithmetic. These constants require a mix of symbolic and high precision computation and it is not practical to compute them during the execution of the algorithm for different precisions. Therefore in this work we turn to forward error bounds, as were used in earlier work [14], [40].

Kenney and Laub [41] showed that for $\|X\| < 1$ and any subordinate matrix norm,

$$\|\log(I - X) - r_{km}^-(X)\| \leq |\log(1 - \|X\|) - r_{km}^-(\|X\|)|, \quad (6.3)$$

where $r_{km}^-(x)$ is the $[k/m]$ approximant to $\log(1-x)$. In subsequent literature, the equivalent bound

$$\|\log(I + X) - r_{km}(X)\| \leq |\log(1 + \|X\|) - r_{km}(\|X\|)| \quad (6.4)$$

has been preferred [30], [31, sec. 11.4]. Both upper bounds can be evaluated at negligible cost, so they provide a way to choose the Padé degrees k and m . We will derive and exploit a new version of the latter bound that it is phrased in terms of the quantities

$$\alpha_p(X) = \max(\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)}), \quad (6.5)$$

for suitable p , instead of $\|X\|$. Since $\alpha_p(X)$ is no larger than $\|X\|$, and can be much smaller when X is highly nonnormal, the new bound leads to a more efficient algorithm.

Since in higher precision our new algorithm may need a considerable number of square roots, it can happen that $\log(I + X)$ has very small norm and thus that an absolute error bound is not sufficient to guarantee that the algorithm will deliver a

result with small relative error. For this reason, unlike in some previous algorithms we will use a relative error bound containing an inexpensive estimate of $\|\log(I + X)\|$.

It is well known [41, Thm. 6] that for X with nonnegative entries and $k + m$ fixed the diagonal Padé approximant ($k = m$) minimizes the error $\|\log(I - X) - r_{km}^-(X)\|$ and the cost in flops of evaluating $r_{km}(X)$ is roughly constant. Therefore diagonal approximants $r_m := r_{mm}$ have been favoured. However, the special case of the Taylor approximant $t_m := r_{m0}$ merits consideration here, as its evaluation requires only matrix multiplications, which in practice are faster than multiple right-hand side solves. Throughout the paper we write f_m to denote either the Padé approximant r_m or the Taylor approximant t_m .

In addition to the matrix logarithm itself, we are also interested in evaluating its Fréchet derivative. Being able to evaluate the Fréchet derivative and its adjoint allows us to estimate the condition number $\kappa_{\log}(A)$ of the matrix logarithm, which in turn gives an estimate of the accuracy of the computed logarithm.

We use the term “multiprecision arithmetic” to mean arithmetic supporting multiple, and usually arbitrary, precisions. These precisions can be lower or higher than the single and double precisions that are supported by the IEEE standard [37] and usually available in hardware. We note that the 2008 revision of the IEEE standard [38] also supports a quadruple precision floating point format and, for storage only, a half precision format.

We begin the paper by summarizing in section 6.2 available multiprecision computing environments. In section 6.3 we derive a new forward bound for the error of Padé approximation of a class of hypergeometric functions, which yields a bound sharper than (6.3) and (6.4) for highly nonnormal matrices. In section 6.4 we describe a new Schur–Padé algorithm for computing the matrix logarithm and its Fréchet derivative at any given precision. Section 6.5 explores a Schur-free version of the algorithm. Numerical examples are presented in section 6.6 and concluding remarks are given in section 6.7.

Finally, we introduce some notation. The unit roundoff of the floating point arithmetic is denoted by u . For $A \in \mathbb{C}^{n \times n}$, the spectral radius of a matrix is denoted by

$\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}$. We recall that the Frechét derivative of a matrix function $f : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ at $A \in \mathbb{C}^{n \times n}$ is the linear map $D_f(A) : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{n \times n}$ that satisfies

$$f(A + E) = f(A) + D_f(A)[E] + o(\|E\|).$$

The relative condition number of the matrix function f at A is defined as

$$\kappa_f(A) = \lim_{\varepsilon \rightarrow 0} \sup_{\|E\| \leq \varepsilon \|A\|} \frac{\|f(A + E) - f(A)\|}{\varepsilon \|f(A)\|},$$

and is explicitly given by the formula [31, Thm. 3.1]

$$\kappa_f(A) = \frac{\|D_f(A)\| \|A\|}{\|f(A)\|}.$$

6.2 SUPPORT FOR MULTIPLE PRECISION ARITHMETIC

A wide range of software supporting multiprecision floating point arithmetic is available. Multiprecision capabilities are a built-in feature of Maple [42] and Mathematica [43] as well as the open-source PARI/GP [46] and Sage [47] computer algebra systems, and are available in MATLAB through the Symbolic Math Toolbox [48] and the Multiprecision Computing Toolbox [45]. The programming language Julia [8] supports multiprecision floating point numbers by means of the built-in data type `BigFloat`, while for other languages third-party libraries are available: `mpmath` [39] and `SymPy` [44], [49], for Python; the GNU MP Library [25] and the GNU MPFR Library [21] for C; the BOOST libraries [10] for C++; and the ARPREC library [5] for C++ and Fortran. The GNU MPFR Library is used in some of the software mentioned above, and interfaces to it for several programming languages are available¹.

¹ See <http://www.mpfr.org>.

6.3 APPROXIMATION OF HYPERGEOMETRIC FUNCTIONS

We recall that the rational function $r_{km} = p_{km}/q_{km}$ is a $[k/m]$ Padé approximant of f if p_{km} and q_{km} are polynomials of degree at most k and m , respectively, $q_{km}(0) = 1$, and $f(x) - r_{km}(x) = O(x^{k+m+1})$. In order to obtain the required error bounds for Padé approximants to the logarithm we consider more generally Padé approximants to the hypergeometric function

$${}_2F_1(a, 1, c, x) = \sum_{i=0}^{\infty} \frac{(a)_i}{(c)_i} x^i,$$

where $(a)_i = a(a+1) \cdots (a+i-1)$ is the Pochhammer symbol for the rising factorial, a and c are real numbers and x is complex with $|x| < 1$. Such Padé approximants have been well studied [6, sec. 2.3].

By combining the analysis of Kenney and Laub [41] with a result of Al-Mohy and Higham [2] we obtain the following stronger version of the error bound [41, Cor. 4]. Recall that α_p is defined in (6.5).

Theorem 6.1. *Let $X \in \mathbb{C}^{n \times n}$ be such that $\rho(X) < 1$. Let \tilde{r}_{km} be the $[k/m]$ Padé approximant to ${}_2F_1(a, 1, c, y)$. If $m \leq k+1$ and $0 < a < c$ then*

$$\|{}_2F_1(a, 1, c, X) - \tilde{r}_{km}(X)\| \leq \|{}_2F_1(a, 1, c, \alpha_p(X)) - \tilde{r}_{km}(\alpha_p(X))\|, \quad (6.6)$$

for any p satisfying $p(p-1) \leq k+m+1$.

Proof. Kenney and Laub [41, Thm. 5] show that if $m \leq k+1$, $|y| < 1$, and $0 < a < c$ then

$${}_2F_1(a, 1, c, y) - \tilde{r}_{km}(y) = \frac{q_{km}(1)}{q_{km}(y)} \sum_{i=k+m+1}^{\infty} \frac{(a)_i(i-k-m)_m}{(c)_i(i+a-m)_m} y^i, \quad (6.7)$$

where q_{km} is the denominator of \tilde{r}_{km} , a polynomial of degree m . By [41, Cor. 1] the zeros of q_{km} are simple and lie on $(1, \infty)$, and since $q_{km}(0) = 1$ it follows that for $|y| < 1$,

$$q_{km}^{-1}(y) = \sum_{j=0}^{\infty} d_j y^j,$$

with $d_i > 0$ for all i . Since $a, c > 0$ and $i > k + m$, the coefficients of the series in (6.7) are positive and so (6.7) can be rewritten as

$${}_2F_1(a, 1, c, y) - \tilde{r}_{km}(y) = \sum_{i=k+m+1}^{\infty} \psi_i y^i, \quad (6.8)$$

where $\text{sign}(\psi_i) = \text{sign}(q_{km}(1))$ for all i . Therefore, applying [2, Thm. 4.2(a)] gives

$$\begin{aligned} \|{}_2F_1(a, 1, c, X) - \tilde{r}_{km}(X)\| &\leq \sum_{i=k+m+1}^{\infty} |\psi_i| \alpha_p(X)^i \\ &= |{}_2F_1(a, 1, c, \alpha_p(X)) - \tilde{r}_{km}(\alpha_p(X))|, \end{aligned}$$

for $p(p-1) \leq m+k+1$. \square

For the matrix logarithm, we have

$$\frac{\log(1+x)}{x} = {}_2F_1(1, 1, 2, -x), \quad (6.9)$$

and thus the $[k/m]$ Padé approximant $r_{km}(x)$ to $\log(1+x)$ and the $[k/m]$ Padé approximant $\tilde{r}_{km}(x)$ to ${}_2F_1(1, 1, 2, x)$ are related by

$$\frac{r_{km}(x)}{x} = \tilde{r}_{k-1,m}(-x). \quad (6.10)$$

The next result is an analog of [34, Thm. 2.2], which applies to the function $(1-x)^t$ for $t \in [-1, 1]$ rather than $\log(1+x)$.

Corollary 6.2. *Let $X \in \mathbb{C}^{n \times n}$ be such that $\rho(X) < 1$ and $\alpha_p(X) < 1$, and let r_{km} be the $[k/m]$ Padé approximant to $\log(1+x)$. Then for $m \leq k$, and p such that $p(p-1) \leq k+m+1$, we have*

$$\|\log(I+X) - r_{km}(X)\| \leq |\log(1 - \alpha_p(X)) - r_{km}(-\alpha_p(X))|. \quad (6.11)$$

Proof. From (6.8), (6.9), and (6.10), with $a = 1$, $c = 2$, and $x = -y$ we have

$$-y^{-1}(\log(1-y) - r_{km}(-y)) = {}_2F_1(1, 1, 2, y) - \tilde{r}_{k-1,m}(y) = \sum_{i=k+m}^{\infty} \psi_i y^i,$$

that is,

$$\log(1-y) - r_{km}(-y) = - \sum_{i=k+m}^{\infty} \psi_i y^{i+1}.$$

We know from the proof of Theorem 6.1 that the ψ_i are one-signed, and so we deduce that

$$\begin{aligned} \|\log(I-X) - r_{km}(-X)\| &\leq \left| \sum_{i=k+m}^{\infty} \psi_i \alpha_p(X)^{i+1} \right| \\ &= |\log(1 - \alpha_p(X)) - r_{km}(-\alpha_p(X))|. \end{aligned}$$

Since $\alpha_p(-X) = \alpha_p(X)$, we obtain the bound (6.11) on replacing X by $-X$. \square

From the condition $p(p-1) \leq k+m+1$ of the corollary we see that (6.11) holds for any $p \in \mathcal{I}_{[k/m]}$, where

$$\mathcal{I}_{[k/m]} = \left\{ n \in \mathbb{N} : 1 \leq n \leq \frac{(1 + \sqrt{5 + 4(k+m)})}{2} \right\}. \quad (6.12)$$

Since the bound (6.11) is decreasing in $\alpha_p(X)$, the smallest bound is obtained for

$$p^* = \arg \min \{ \alpha_p(X) : p \in \mathcal{I}_{[k/m]} \}. \quad (6.13)$$

In practice we will approximate p^* rather than compute it exactly, as discussed in section 6.4.

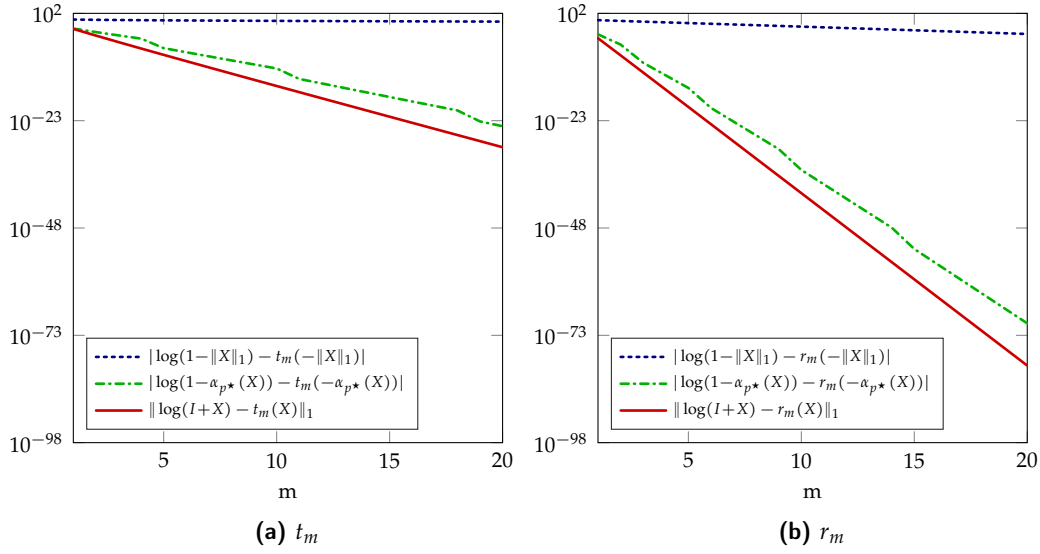


Figure 6.1: Comparison of the bounds (6.4) and (6.11) for A in (6.14) and $1 \leq m \leq 20$, with p^* given by (6.13).

We compare in Figure 6.1 the bounds (6.4) and (6.11) for the diagonal Padé approximant r_m and the Taylor approximant t_m , for m between 1 and 20, with the fairly nonnormal matrix

$$A = \begin{bmatrix} 0.01 & 0.95 \\ 0 & 0.04 \end{bmatrix}. \quad (6.14)$$

We see that for both t_m and r_m the new bound can be many orders of magnitude smaller than (6.4) and it is a much better estimate of the actual error.

6.4 SCHUR-PADÉ ALGORITHM

In this section and the next we develop two new algorithms for computing the matrix logarithm in arbitrary precision floating point arithmetic, one using the Schur decomposition and one transformation-free. The algorithms build on the inverse scaling and squaring algorithms in [3], [4], [14], [31, Algs 11.9, 11.10]. They combine

features from these algorithms in a novel way that yields algorithms that take the unit roundoff as an input parameter and need no pre-computed constants.

We approximate the Fréchet derivative of the logarithm by the Fréchet derivative of the logarithm’s Padé approximant. We will not give an error bound for this approximation because, as noted in [4], it is problematic to obtain error bounds for it that are expressed in terms of the $\alpha_p(A)$. However our main intended use of the Fréchet derivative is for condition number estimation, which does not require accurate derivatives, and the same approximation was found to perform well at double precision in [4].

Our precision-independent algorithm for the matrix logarithm and its Fréchet derivative is given in Algorithm 6.1. Instructions with an underlined line number are to be executed only when the Fréchet derivative of the matrix logarithm is required.

The algorithm begins by computing the Schur decomposition $A = QTQ^*$. In lines 9–10 it repeatedly takes square roots of T until the spectrum of $T - I$ is within the unit disk centered at 0. Although the requirement $\rho(X) = \rho(T - I) < 1$ in Corollary 6.2 is now satisfied, there is no guarantee that the relative forward error $\|\log T - f_{m_{\max}}(T - I)\|_1 / \|\log T\|_1$, where m_{\max} is the maximum degree of approximant allowed and f_m denotes r_m or t_m , is less than the unit roundoff u . This is especially true for Taylor approximation, which could need hundreds of terms to achieve a bound on the forward error smaller than u .

Hence in lines 11–12 the algorithm keeps taking square roots until

$$|\log(1 - \alpha_p(T - I)) - f_{m_{\max}}(-\alpha_p(T - I))| < u\psi(T), \quad (6.15)$$

where $\psi(T)$ is an estimate of the 1-norm of $\log T$ and p is chosen as described below. Approximating $\log T$ by the first term of the Taylor series, $\psi(T) = \|T - I\|_1$, provides an estimate accurate enough for all matrices and levels of precision considered in our numerical experiments.

Algorithm 6.1: Schur–Padé matrix logarithm with Fréchet derivative.

Given $A \in \mathbb{C}^{n \times n}$ with no eigenvalues on \mathbb{R}^- this algorithm computes $X = \log A$, and optionally the Fréchet derivative $Y \approx D_{\log}(A)[E]$, in floating point arithmetic with unit roundoff u using inverse scaling and squaring with Padé approximation. s_{\max} is the maximum allowed number of square roots and m_{\max} the maximum allowed approximant degree. The logical parameter `use_taylor` determines whether a diagonal Padé approximant or a Taylor approximant is to be used. $\psi(X)$ provides an approximation to $\|\log X\|_1$.

```

1  Compute the complex Schur decomposition  $A = QTQ^*$ 
2  if use_taylor then
3    |  $f = t$  and  $\zeta(m)$  is defined as in (6.20).
4  else
5    |  $f = r$ ,  $\zeta(m) \leftarrow m - 2$ 
6   $E \leftarrow Q^*EQ$ 
7   $T_0 \leftarrow T$ 
8   $s \leftarrow 0$ 
9  while  $\max_{1 \leq i \leq n} (|\sqrt{t_{ii}} - 1|) > 1$  and  $s < s_{\max}$  do
10   |  $[T, E, s] \leftarrow \text{SQRTM}(T, E, s)$ 
11  while  $|\log(1 - \tilde{\alpha}_{m_{\max}}^f(T - I)) - f_{m_{\max}}(-\tilde{\alpha}_{m_{\max}}^f(T - I))| \geq u\psi(T)$  and  $s < s_{\max}$  do
12   |  $[T, E, s] \leftarrow \text{SQRTM}(T, E, s)$ 
13   $\tilde{m} \leftarrow \min\{m \leq m_{\max} : |\log(1 - \tilde{\alpha}_m^f(T - I)) - f_m(-\tilde{\alpha}_m^f(T - I))| < u\psi(T)\}$ 
14  while  $|\log(1 - \tilde{\alpha}_{\zeta(\tilde{m})}^f(T - I)/2) - f_{\zeta(\tilde{m})}(-\tilde{\alpha}_{\zeta(\tilde{m})}^f(T - I)/2)| < u\psi(T)$  and  $s < s_{\max}$  do
15   |  $[T, E, s] \leftarrow \text{SQRTM}(T, E, s)$ 
16   |  $\tilde{m} \leftarrow \min\{m \leq \tilde{m} : |\log(1 - \tilde{\alpha}_m^f(T - I)) - f_m(-\tilde{\alpha}_m^f(T - I))| < u\psi(T)\}$ 
17   $\text{diag}(T, 1) \leftarrow \text{diag}(T_0^{1/2^s}, 1)$  using [33, eq. (5.6)].
18   $\text{diag}(T) \leftarrow \text{diag}(T_0)^{1/2^s} - I$  using [1, Alg. 2].
19   $X \leftarrow 2^s f_{\tilde{m}}(T)$ 
20   $\text{diag}(T) \leftarrow \log(\text{diag}(T_0))$ 
21  Update the superdiagonal of  $T$  using [31, eq. (11.28)].
22   $X \leftarrow QXQ^*$ 
23   $Y \leftarrow 2^s QL_{f_m}(T, E)Q^*$  using [4, eq. (2.3)].

24  function  $\text{SQRTM}(T \in \mathbb{C}^{n \times n}, E \in \mathbb{C}^{n \times n}, s \in \mathbb{N})$ 
    ▷ Compute square root of  $T$  and update algorithmic parameters.
25   $T \leftarrow T^{1/2}$  using [9, Alg. 6.3].
26   $E \leftarrow X$ , where  $X$  is the solution of  $TX + XT = E$ .
27  return  $T, E, s + 1$ 

```

Note that $\alpha_p(\|X\|_1)$ can be estimated efficiently, without explicitly forming any powers of X using the block 1-norm estimation algorithm of Higham and Tisseur [35], which requires only $O(n^2)$ flops. Since only an estimate of $\|X^p\|^{1/p}$ is computed there is no need to use high precision for this sub-problem, so we carry out this computation in double precision (or single precision if the requested precision is lower than double), in order to exploit the floating point hardware. When the matrix dimension is small and the working precision is not too high the cost of estimating $\alpha_p(T - I)$ can nevertheless be non-negligible. Rather than computing α_p for the optimal value p^* in (6.13), we compute it only for the largest possible p . Some justification for this choice comes from the fact that despite a sometimes considerably nonmonotonic behaviour, the sequence $\{\alpha_p(X)\}_{p \in \mathbb{N}}$ is typically roughly decreasing [2]. We denote the α value corresponding to the diagonal Padé approximant r_m by

$$\tilde{\alpha}_m^r(X) = \alpha_p(X), \quad p = \left\lfloor (1 + \sqrt{5 + 8m})/2 \right\rfloor,$$

and that corresponding to the truncated Taylor series t_m by

$$\tilde{\alpha}_m^t(X) = \alpha_p(X), \quad p = \left\lfloor (1 + \sqrt{5 + 4m})/2 \right\rfloor.$$

Thus (6.15) is used in the form

$$|\log(1 - \tilde{\alpha}_{m_{\max}}^f(T - I)) - f_{m_{\max}}(-\tilde{\alpha}_{m_{\max}}^f(T - I))| < u\psi(T). \quad (6.16)$$

We now discuss the cost of the algorithm, beginning with the case of diagonal Padé approximants. Higham [30] considered several ways of evaluating the rational function $r_m(X)$. The partial fraction form

$$r_m(A) = \sum_{j=1}^m \gamma_j^{(m)} (I + \delta_j^{(m)} A)^{-1} A, \quad (6.17)$$

where $\gamma_j^{(m)}$ and $\delta_j^{(m)}$ are the weights and nodes, respectively, of the m -point Gauss–Legendre quadrature rule on $[0, 1]$, was found to provide the best balance between efficiency and numerical stability, and it also has the advantage of allowing parallel

evaluation. For a triangular matrix of size n , the evaluation of (6.17) requires $mn^3/3$ flops and the computation of each of the s matrix square roots costs $n^3/3$ flops. Thus, when diagonal approximants are used, the algorithm requires $25n^3$ flops for the computation of the Schur decomposition, $\chi_r(s, m) := (s + m) n^3/3$ for the inverse scaling and squaring phase, and $3n^3$ flops to recover the solution.

Since $\chi_r(s, m) = \chi_r(s + 1, m - 1)$, an additional square root will save computational effort only if the degree of the approximant decreases by at least 2, in which case the overall reduction in cost is at least $n^3/3$ flops. In view of the approximation [3]

$$\alpha_p(A^{1/2^{s+1}} - I) \approx \frac{\alpha_p(A^{1/2^s} - I)}{2}, \quad (6.18)$$

an additional square root is taken only if

$$|\log(1 - \tilde{\alpha}_{\zeta(\tilde{m})}^f(T - I)/2) - f_{\zeta(\tilde{m})}(-\tilde{\alpha}_{\zeta(\tilde{m})}^f(T - I)/2)| < u\psi(T), \quad (6.19)$$

with $f \equiv r$ and $\zeta(\tilde{m}) = \tilde{m} - 2$, where \tilde{m} is the current degree of the approximant (defined in line 13 of Algorithm 6.1).

Turning to Taylor series approximants, in order to evaluate the truncated Taylor series the algorithm uses the Paterson–Stockmeyer method, which among the four methods for polynomial evaluation considered in [31, Thm. 4.5] is the one that minimizes the number of matrix-matrix multiplications while satisfying a forward error bound of the same form as for the other methods. The computational cost of computing s square roots and evaluating a Taylor approximant of degree m is approximately $\chi_t(s, m) = (s + 2\sqrt{m}) n^3/3$ flops. Simple algebraic manipulations show that

$$\chi_t(s, m) = \chi_t\left(s + 1, \left(\sqrt{m} - \frac{1}{2}\right)^2\right),$$

and thus an additional square root is taken only if (6.19) holds with $f \equiv t$ and

$$\zeta(\tilde{m}) = \left\lceil \left(\sqrt{\tilde{m}} - \frac{1}{2}\right)^2 \right\rceil - 1, \quad (6.20)$$

which guarantees that $\zeta(\tilde{m}) < (\sqrt{\tilde{m}} - \frac{1}{2})^2$. Since the cost function χ_t is monotonic in both arguments, the reduction in the number of flops will be at least

$$\chi_t(s, \tilde{m}) - \chi_t(s+1, \zeta(\tilde{m})) = \frac{2n^3}{3} \left(\sqrt{\tilde{m}} - \left(\left\lceil \left(\sqrt{\tilde{m}} - \frac{1}{2} \right)^2 \right\rceil - 1 \right)^{1/2} - \frac{1}{2} \right), \quad (6.21)$$

which depends only on \tilde{m} . Unlike in the Padé case, we cannot put a useful lower bound on the decrease in flops resulting from an extra square root.

For both types of approximant we can perform a binary search to find the smallest $m^* \in [1, \tilde{m}]$ such that

$$|\log(1 - \tilde{\alpha}_{m^*}^f(T - I)) - f_{m^*}(-\tilde{\alpha}_{m^*}^f(T - I))| < u\psi(T), \quad (6.22)$$

which requires the estimation of $\|X^p\|_1^{1/p}$ for no more than $2 \log_2 \tilde{m} - 1$ values of p .

If the Frechét derivative is needed then each time a square root is taken the algorithm solves an additional Sylvester equation, as detailed in [31, sec. 11.8]. Once the optimal values for s and \tilde{m} have been found, in order to increase the accuracy the algorithm recomputes the first superdiagonal of T from T_0 , making use of the identity [33, eq. (5.6)], and then the main diagonal of $T - I$, by applying [1, Alg. 2] to the diagonal of T_0 . The algorithm can be easily adapted to compute the adjoint of the Frechét derivative of A in the direction E , by replacing the increment E by E^* and returning Y^* [4].

Algorithm 6.1 can be extended to compute an estimate of the 1-norm condition number of the matrix logarithm, using the same approach as in [4, Alg. 4.1]. Since in this case the Frechét derivative of the matrix logarithm of A and its adjoint need to be computed in several directions, but neither s nor \tilde{m} depend on E , the algorithm can be modified to store the matrix T after each square root is taken, and then use it to solve several Sylvester cascades for different matrices E . If η is the number of bits required to store a single entry of the matrix, then this modification increases the memory requirement of the algorithm by about $\eta(s-1)n^2/2$ bits if the upper triangular pattern is exploited.

6.5 TRANSFORMATION-FREE ALGORITHM

Multiprecision computing environments often provide just a few linear algebra kernels. For example, version 7.1 of the Symbolic Math Toolbox [48] does not support the Schur decomposition in its variable precision arithmetic (VPA). In this section we therefore present a version of Algorithm 6.1 that does not require the computation of the Schur decomposition and builds solely on level 3 BLAS operations and multiple right-hand side system solves.

The algorithm, whose pseudocode is given in Algorithm 6.2, builds on the transformation-free algorithms [3, Alg. 5.2], [14, Alg. 7.1], and again makes use of the improved forward error bound (6.11).

The algorithm starts by taking enough square roots to guarantee that the Padé or Taylor approximants will produce a relative forward error below the unit roundoff threshold. Since in this case the matrix is not triangular, to compute square roots the algorithm employs the scaled Denman–Beavers iteration (in product form) [31, eq. (6.29)], whose computational cost depends on the number of iterations and is thus not known a priori. However, it has been observed [31, sec. 11.5.2] that in practice up to ten iterations are typically required for the first few square roots, but just four or five are enough in the later stages. Since the cost of one iteration is $4n^3$ flops, it is customary to consider that the computation of a square root requires $16n^3$ flops. On the other hand, evaluating the diagonal Padé approximant in partial fraction form requires $8mn^3/3$ if the matrix is not upper triangular. The cost of the algorithm is $\chi_r(s, m)$ flops, where

$$\chi_r(s, m) = \left(\frac{8m}{3} + 16s \right) n^3,$$

and it can be readily seen that $\chi_r(s, m) = \chi_r(s+1, m-6)$, and thus an additional square root is taken if (6.19) holds for $f \equiv r$, $T = A$, and $\zeta(\tilde{m}) = \tilde{m} - 7$. Using the Paterson–Stockmeyer scheme to evaluate the truncated Taylor expansion, we get the asymptotic cost $\chi_t(s, m) = 4(\sqrt{m} + 4s)n^3$. Since the cost function χ_t satisfies

Algorithm 6.2: Transformation-free matrix logarithm with Fréchet derivative.

Given $A \in \mathbb{C}^{n \times n}$ with no eigenvalues on \mathbb{R}^- this algorithm computes $X = \log A$, in floating point arithmetic with unit roundoff u using inverse scaling and squaring with Padé approximation. s_{\max} is the maximum allowed number of square roots and m_{\max} the maximum allowed approximant degree. The logical parameter `use_taylor` determines whether a diagonal Padé approximation or a Taylor approximation is to be used. $\psi(X)$ provides an approximation to $\|\log X\|_1$.

```

1  if use_taylor then
2    |  $f = t$  and  $\zeta(m)$  is defined as in (6.23).
3  else
4    |  $f = r, \zeta(m) \leftarrow m - 7$ 
5   $s \leftarrow 0$ 
6   $Z \leftarrow A - I$ 
7   $P \leftarrow I$ 
8  while  $|\log(1 - \tilde{\alpha}_m^f(A - I)) - f_m(-\tilde{\alpha}_m^f(A - I))| \geq u\psi(A)$  and  $s < s_{\max}$  or  $\|A\|_1 > 1$  do
9    |  $[A, P, s] \leftarrow \text{SQRTMDB}(A, P, s)$ 
10  $\tilde{m} \leftarrow \min\{m \leq m_{\max} : |\log(1 - \tilde{\alpha}_m^f(A - I)) - f_m(-\tilde{\alpha}_m^f(A - I))| < u\psi(A)\}$ 
11 while  $|\log(1 - \tilde{\alpha}_{\zeta(\tilde{m})}^f(A - I)/2) - f_{\zeta(\tilde{m})}(-\tilde{\alpha}_{\zeta(\tilde{m})}^f(A - I)/2)| < u\psi(A)$  and  $s < s_{\max}$ 
12   do
13     |  $[A, P, s] \leftarrow \text{SQRTMDB}(A, P, s)$ 
14     |  $\tilde{m} \leftarrow \min\{m \leq \tilde{m} : |\log(1 - \tilde{\alpha}_m^f(A - I)) - f_m(-\tilde{\alpha}_m^f(A - I))| < u\psi(A)\}$ 
15 if  $s < 2$  then
16   |  $Y \leftarrow A - I$ 
17 else
18   |  $Y \leftarrow ZP^{-1}$ 
19  $X \leftarrow 2^s f_{\tilde{m}}(Y)$ 
20 return  $X$ 

21 function SQRTMDB( $A \in \mathbb{C}^{n \times n}, P \in \mathbb{C}^{n \times n}, s \in \mathbb{N}$ )
22    $\triangleright$  Compute square root of  $A$  and update algorithmic parameters.
23    $A \leftarrow A^{1/2}$  using the iteration in [31, eq. (6.29)].
24   if  $s > 1$  then
25     |  $P \leftarrow P(A + I)$ 
26   else
27     | return
28    $A, P, s$ 

```

$\chi_t(s, m) = \chi_t(s + 1, (\sqrt{m} - 4)^2)$, an additional square root will be worthwhile if (6.19) holds for $f \equiv t$, $T = A$, and

$$\zeta(\tilde{m}) = \left\lceil (\sqrt{\tilde{m}} - 4)^2 \right\rceil - 1, \quad (6.23)$$

which guarantees a reduction in the number of flops of at least

$$\chi_t(s, \tilde{m}) - \chi_t(s + 1, \zeta(\tilde{m})) = 4n^3 \left(\sqrt{\tilde{m}} - \left(\left\lceil (\sqrt{\tilde{m}} - 4)^2 \right\rceil - 1 \right)^{1/2} - 4 \right). \quad (6.24)$$

To reduce the chances of numerical cancellation in the computation of $A^{1/2^s} - I$, the matrix form of [1, Alg. 2] is used, as in [3].

Note that Algorithm 6.2 is not suitable for the computation of the Fréchet derivative of the matrix logarithm, nor for the estimation of its condition number. Standard methods for the solution of Sylvester equations [7], [23] start by computing the Schur decomposition of one or both the coefficients of the matrix equation, and are thus not suitable for a framework where a multiprecision implementation of the QR algorithm is not available. The alternative of converting the Sylvester equation to an $n^2 \times n^2$ structured linear system $Ax = b$ and solving by Gaussian elimination has too high a computational cost to be useful in this context.

6.6 NUMERICAL EXPERIMENTS

In this section we describe numerical tests with the new multiprecision algorithms for the matrix logarithm. All the experiments were performed using the 64-bit version of MATLAB 2017b on a machine equipped with an Intel I5-5287U processor running at 2.90GHz. The collection of MATLAB scripts and functions that generate the test matrices and produce the figures and tables in this section is available on GitHub,² and the MATLAB functions implementing Algorithm 6.1 and Algorithm 6.2 can also be retrieved on MATLAB Central.³ For the underlying computations the implemen-

² <https://github.com/mfasi/mplogm>.

³ <https://uk.mathworks.com/matlabcentral/fileexchange/63841>.

tations exploit the overloaded methods from the Multiprecision Computing Toolbox (version 4.3.2.12168) [45] to run in different precisions. The precision is specified in terms of the number of decimal digits.

We test the following algorithms.

- `logm_mct`: the (overloaded) `logm` function from the Multiprecision Computing Toolbox, which implements a blocked version of the Schur–Parlett algorithm [16]. After computing the complex Schur decomposition, a blocking of the matrix is computed according to [16], [31, sec. 9.3]. For each diagonal block T_{ii} of the triangular Schur factor the algorithm repeatedly takes the square root until the spectrum of $T_{ii} - I$ lies within the unit ball centered at 1. Then it chooses the degree of the (diagonal) Padé approximant as the smallest m so that the bound in (6.3) is less than the unit roundoff. Since this strategy does not optimize the balance between the number of square roots and the Padé degree it tends to choose higher degrees than our algorithm. The off-diagonal blocks are obtained via the block Schur–Parlett recurrence.
- `logm_agm`: an algorithm for the computation of the matrix logarithm based on the arithmetic-geometric mean iteration. In particular, we use [12, Thm. 8.2], which gives the approximation $\log A \approx \log(4/\varepsilon)I - (\pi/2) \text{AGM}(\varepsilon A)^{-1}$, where $\varepsilon = \sqrt{u}/\|A\|_F$ and $\text{AGM}(A)$ is the arithmetic-geometric mean iteration, which we compute by means of the stable double recursion [12, eqs. (5.2), (5.3)] with stopping criterion $\|P_k - I\|_F \leq u\|A\|_F$. We do not implement the optimization in [12, sec. 7], because it is precision dependent and aimed at speed rather than accuracy.
- `logm_pade`: the version of Algorithm 6.1 employing diagonal Padé approximants and relative error bounds, i.e. $\psi(X) = \|X - I\|_1$.
- `logm_pade_abs`: the version of Algorithm 6.1 employing diagonal Padé approximants and absolute error bounds, i.e. $\psi(X) = 1$.
- `logm_tayl`: the version of Algorithm 6.1 employing truncated Taylor approximants and relative error bounds, i.e. $\psi(X) = \|X - I\|_1$.

- `logm_tayl_abs`: the version of Algorithm 6.1 employing truncated Taylor approximants and absolute error bounds, i.e. $\psi(X) = 1$.
- `logm_tfree_pade`: the transformation-free Algorithm 6.2 employing diagonal Padé approximants and relative error bounds, i.e. $\psi(X) = \|X - I\|_1$.
- `logm_tfree_tayl`: the transformation-free Algorithm 6.2 employing truncated Taylor approximants and relative error bounds, i.e. $\psi(X) = \|X - I\|_1$.
- `logm`: the built-in MATLAB function that implements the algorithms for real and complex matrices from [3], [4] and is designed for double precision only.

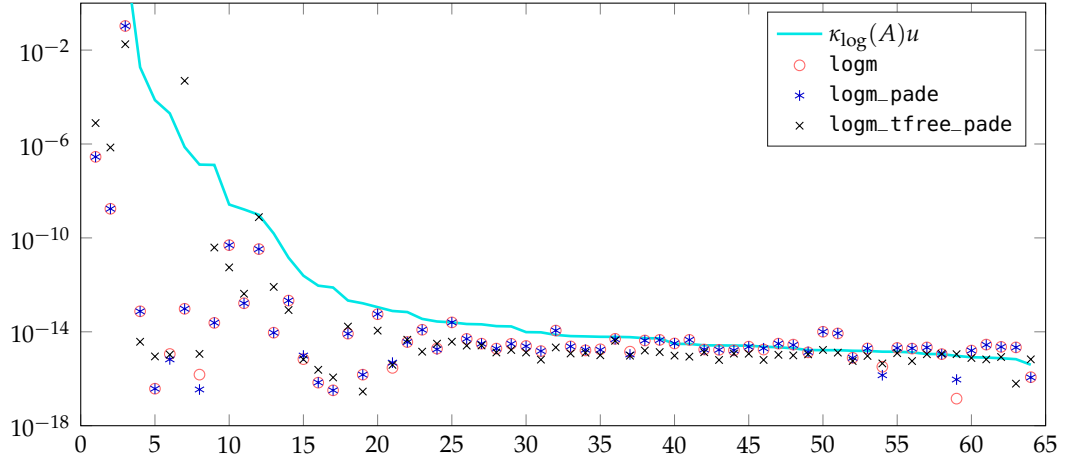
In the implementations, m_{\max} and s_{\max} are set to 200 and 100, for diagonal Padé approximants, and to 400 and 100, for truncated Taylor series, respectively.

The Gauss–Legendre nodes and weights in (6.17) are computed by means of the GaussLegendre method of the `mp` class provided by the Multiprecision Computing Toolbox. This algorithm is based on Newton root-finding [22], which computes the nodes and weights of the quadrature formula of order m in $O(m)$ flops. This is more efficient than the Golub–Welsh algorithm [24], which is based on the computation of the eigensystem of a tridiagonal matrix and costs $O(m^2)$ flops. The Schur decomposition is computed using the `schur` function of the `mp` class.

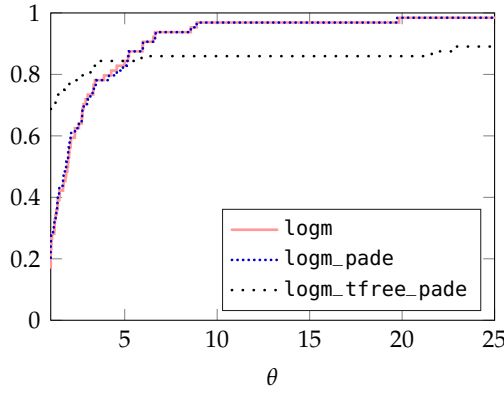
We note that version 7.1 of the Symbolic Math Toolbox provides an overloaded version of the MATLAB function `logm`, but for several of our test matrices this function either gives an error or fails to return an answer, so we exclude it from our tests.

We evaluate the forward errors $\|X - \hat{X}\|_1 / \|X\|_1$, where \hat{X} is a computed solution and $X \approx \log A$ is a reference solution computed with `logm_pade` at a precision of $8d$ decimal significant digits, where d is the number of digits used for the computation of \hat{X} .

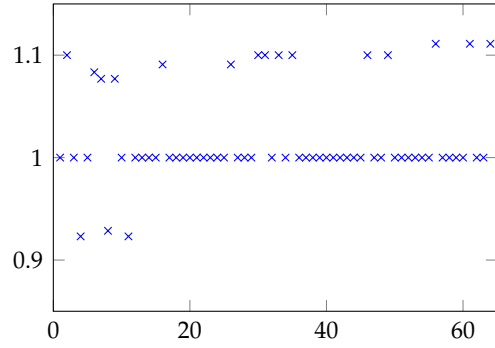
To gauge the forward stability of the algorithms we plot the quantity $\kappa_{\log}(A)u$, where $\kappa_{\log}(A)$ is the 1-norm condition number of the matrix logarithm of A estimated using `funm_condest1` from the Matrix Function Toolbox [29], with the aid of the Fréchet derivatives in Algorithm 6.1.



(a) Forward errors.



(b) Performance profile for data in (a).



(c) Computational cost.

Figure 6.2: Top: forward errors of `logm_pade`, `logm_tfree_pade`, and `logm` on the test set, all running in IEEE double precision arithmetic. Bottom left: performance profile. Bottom right: on the same test set, the number of square roots and multiple right-hand side linear system solves for `logm_pade` divided by the corresponding number for `logm`.

We use a test set of 64 matrices, of sizes ranging from 2×2 to 100×100 , including matrices from the literature of the matrix logarithm and from the MATLAB gallery function.

6.6.1 Comparison with `logm` in double precision

Our first experiment compares `logm_pade` and `logm_tfree_pade` running in IEEE double precision ($u = 2^{-53}$) with the built-in MATLAB function `logm`, in order to check that the new algorithms perform well in double precision. Figure 6.2a shows the forward errors sorted by decreasing condition number of the matrix logarithm. Fig-

ure 6.2b reports the same data in the form of a performance profile [20], which we compute with the MATLAB function `perfprof` described in [27, sec. 26.4]. Here, for each method M the height of the line at θ on the x -axis represents the fraction of matrices in the test set for which the relative forward error of M is at most θ times that of the algorithm that delivers the most accurate result. In our performance profiles we use the technique of Dingle and Higham [19] to rescale errors smaller than the unit roundoff in order to avoid abnormally small errors skewing the profiles.

The results show that `logm_pade` and `logm` produce errors bounded approximately by $\kappa_{\log}(A)u$, that is, they behave in a forward stable manner. The same is true of `logm_tfree_pade` except for one matrix, and this algorithm is most often the most accurate, while also being the least reliable, as shown by the performance profile.

Figure 6.2c shows that the computational cost of `logm_pade` can be lower or (more often) higher than that of `logm`, but overall is comparable with the state of the art. We note that `logm_pade` computes more square roots than `logm` on 23% of the matrices in our data set. On these matrices, `logm` requires between 9% and 33% (with an average of almost 22%) fewer square roots, but `logm_pade` typically compensates for this by evaluating a Padé approximant of lower degree.

As a further experiment we sought to maximize the ratios between the forward errors of `logm` and `logm_pade`, using the multidirectional search method of Dennis and Torczon [17], implemented in the `mdsmax` function in the Matrix Computation Toolbox [28]. Initializing that method with random 10×10 matrices with no positive real eigenvalues we have not been able to find a matrix for which either ratio of errors exceeds 1.4. This provides further evidence that the two algorithms deliver similar accuracy.

6.6.2 Relative and absolute error

Now we show the importance of using a relative error bound as opposed to an absolute bound, as was used in earlier algorithms intended for double precision [13], [14], [18], [40]. Figure 6.3 reports how the relative forward error to unit roundoff

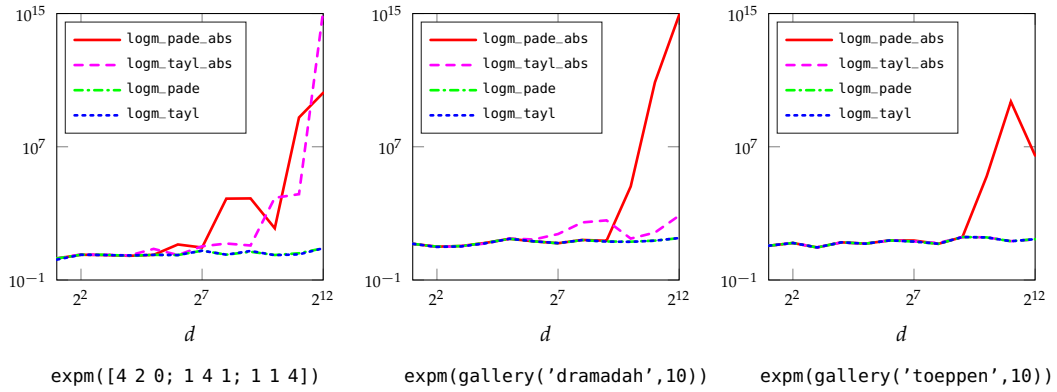


Figure 6.3: Forward error divided by unit roundoff $u = 2^{\lceil \log_2(10^{-d}) \rceil}$, where the number of decimal significant digits d is shown on the x -axis, for three matrices in the test set.

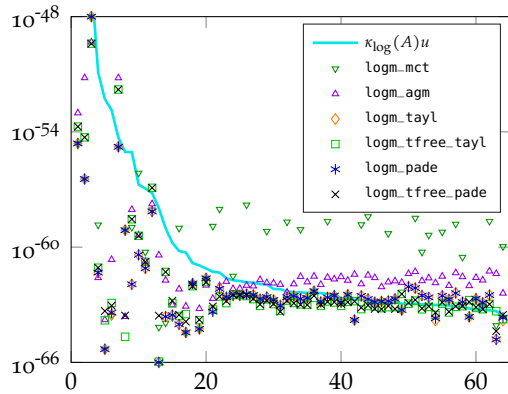
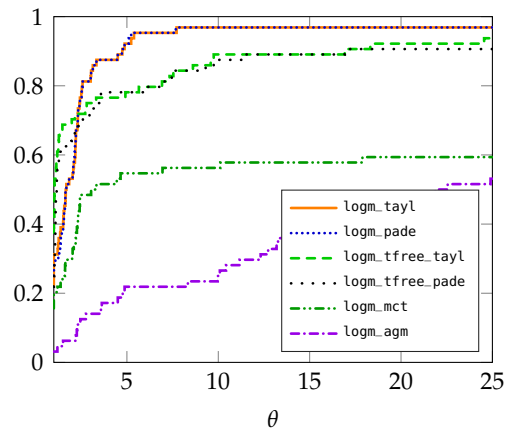
ratio varies, as the precision increases, for `logm_pade`, `logm_pade_abs`, `logm_tayl` and `logm_tayl_abs`, on three of our test matrices.

The ratio for `logm_pade` and `logm_tayl` is influenced by the conditioning of the problem, which is below 100 for these matrices, but tends to remain stable as the working precision increase. The ratio for the algorithms based on an absolute error bounds, on the other hand, grows exponentially, and we can conclude that `logm_pade_abs` and `logm_tayl_abs` are unstable.

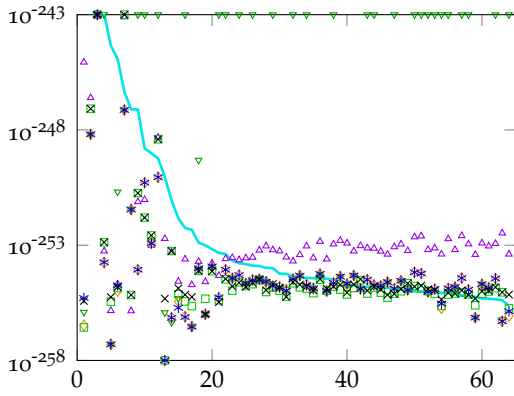
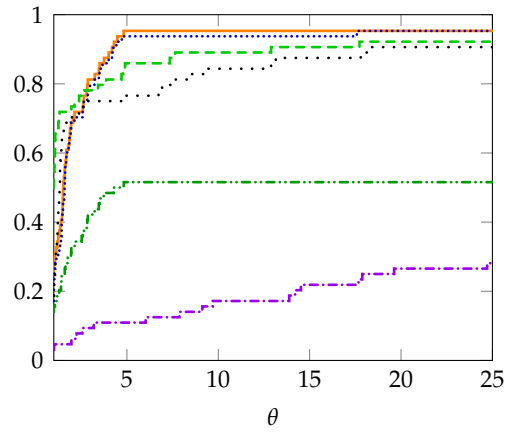
6.6.3 Experiments at higher precisions

Now we compare the accuracy of Algorithm 6.1, Algorithm 6.2, and several competing methods at four different precisions. Figure 6.4a plots, for the matrices in our test set sorted by decreasing condition number, the relative forward errors of `logm_mct`, `logm_agm`, `logm_tayl`, `logm_tfree_tayl`, `logm_pade`, and `logm_tfree_pade` against $\kappa_{\log}(A)u$. If the forward error of an algorithm falls outside the range reported in the graph, we put the corresponding marker on the nearest edge (top or bottom) of the plot. The right-hand column of Figure 6.4 reports the same data in the form of performance profiles.

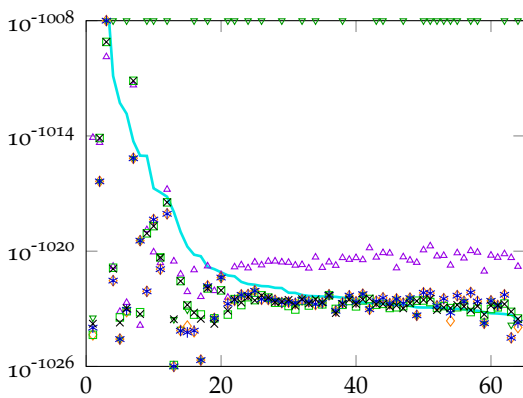
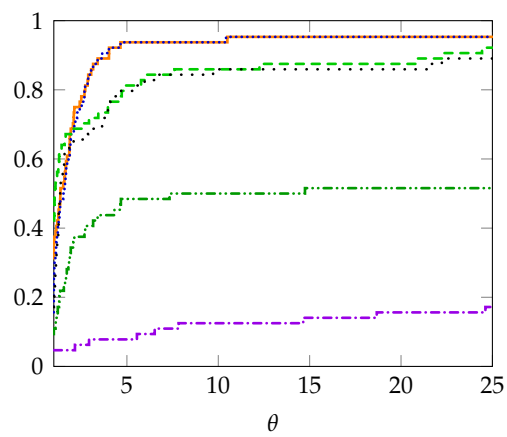
For a working precision of 16 digits (the results for which are not shown here), the six algorithms exhibit a similar behaviour. As illustrated in Figure 6.4c and 6.4e, as the precision u becomes smaller `logm_mct` loses accuracy on almost 40 percent of the

(a) $u = 2^{-213}$ (~64 digits)

(b) Performance profile for data in (a).

(c) $u = 2^{-851}$ (~256 digits)

(d) Performance profile for data in (c).

(e) $u = 2^{-426}$ (~1024 digits)

(f) Performance profile for data in (e).

Figure 6.4: Forward errors for 64, 256, and 1024 digit precisions.

Table 6.1: Execution time breakdown of `logm_tayl` and `logm_pade`, run with $u = 2^{-1701}$ on three matrices of increasing size. The table reports, for each algorithm, the number of square roots (s), the degree of the Padé approximant (m), the total execution time in seconds (T_{tot}), and the percentage of time spent computing the Schur decomposition (T_{sch}), taking the square roots (T_{sqr}), evaluating the scalar bound (T_{bnd}), and evaluating the Taylor and Padé approximants (T_{eval}).

	n	logm_tayl							logm_pade						
		s	m	T_{sch}	T_{sqr}	T_{bnd}	T_{eval}	T_{tot}	s	m	T_{sch}	T_{sqr}	T_{bnd}	T_{eval}	T_{tot}
A	10	41	40	12%	46%	4%	37%	0.6	19	37	9%	42%	19%	30%	0.6
	20	39	40	32%	35%	2%	32%	0.9	18	36	25%	26%	16%	33%	1.1
	50	41	40	50%	32%	0%	18%	8.9	18	38	53%	17%	2%	29%	8.5
	100	41	40	56%	32%	0%	12%	62.5	19	37	58%	16%	0%	25%	59.5
	200	40	40	57%	33%	0%	10%	458.0	18	38	60%	16%	0%	24%	432.4
	500	41	40	43%	45%	0%	12%	5113.7	19	37	47%	23%	0%	29%	4642.0
B	10	45	40	14%	47%	4%	35%	0.4	22	38	9%	47%	20%	24%	0.5
	20	45	40	30%	37%	1%	31%	1.0	22	39	28%	33%	5%	34%	1.1
	50	46	40	44%	39%	0%	17%	8.2	24	37	45%	23%	2%	30%	7.9
	100	46	40	47%	40%	0%	13%	57.0	24	38	50%	22%	0%	27%	54.0
	200	47	40	47%	41%	0%	11%	426.2	25	37	51%	24%	0%	25%	397.6
	500	48	40	37%	51%	0%	12%	5283.2	24	40	41%	28%	0%	31%	4784.2
C	10	46	39	21%	42%	3%	35%	0.4	24	36	11%	39%	26%	23%	0.6
	20	46	40	37%	34%	1%	27%	1.1	24	37	33%	29%	9%	29%	1.2
	50	47	40	52%	34%	0%	14%	9.6	23	40	53%	19%	1%	27%	9.3
	100	48	40	55%	34%	0%	10%	69.1	25	38	58%	19%	0%	22%	65.6
	200	49	40	55%	36%	0%	9%	511.4	26	38	58%	20%	0%	21%	481.5
	500	52	40	48%	43%	0%	9%	6797.7	28	38	52%	25%	0%	23%	6234.3

matrices, and the accuracy of the solution degrades quickly with respect to $\kappa_{\log}(A)u$. The loss of accuracy of `logm_agm` is not as severe, but it affects the entire dataset and is particularly noticeable for well-conditioned matrices.

The new algorithms show a forward stable behavior, since the forward error remains less than or only slightly larger than $\kappa_{\log}(A)u$ as the precision u becomes smaller. On our test set, Algorithm 6.1 is more accurate than Algorithm 6.2, and the performance of `logm_tayl` and `logm_pade` is almost identical while `logm_tfree_tayl` is more accurate than `logm_tfree_pade` and often provides the most accurate result on the best-conditioned of the test matrices.

6.6.4 Code profiling

Table 6.1 compares the execution times of our implementations of `logm_tayl` and `logm_pade`, profiling the four main operations performed by the algorithms: Schur

decomposition, square roots, evaluation of the bound to determine the degree of the Padé approximant to be used (which includes computation of the norm estimates used in forming the α_p), and evaluation of the approximant itself. We consider the matrices

```
A = expm(gallery('chebvand', n))
B = expm(gallery('randsvd', n))
C = expm(gallery('chow', n))
```

The unit roundoff is 2^{-1701} , which roughly gives 512 decimal significant digits.

In both cases, evaluating the scalar bound (T_{bnd}) is relatively expensive for small matrices, but its impact drops as the size of the matrices increases and it is typically negligible for matrices of size larger than 100. We can see that `logm_tayl` needs approximately twice as many square roots as `logm_pade` on these matrices, and that while the evaluation time (T_{eval}) is larger for `logm_pade` this algorithm is slightly faster in most cases.

6.7 CONCLUSIONS

The state of the art inverse scaling and squaring algorithms for the matrix logarithm and the matrix exponential, implemented in the MATLAB functions `logm` and `expm`, are tuned specifically for double or single precision arithmetic, via the use of pre-computed constants obtained from backward error bounds. This approach does not extend in any convenient way to a multiprecision computing environment. Here we have shown that by using forward error bounds we can obtain algorithms for the matrix logarithm that perform in a forward stable way across a wide range of precisions. The Schur-based algorithms, based on Algorithm 6.1, are competitive with `logm` when run in double precision and are superior to existing algorithms at higher precisions. For computing environments lacking a variable precision Schur decomposition we recommend the transformation-free Algorithm 6.2.

The algorithms rely on three innovations. First, we have derived a new sharper version of the forward error bound of Kenney and Laub [41] that can be much smaller for nonnormal matrices. Second, we have implemented the bound in the form of a relative error bound, as we found that the absolute error bounds used in some previous algorithms yield poor results at high precision due to the need for X in the approximations to $\log(I + X)$ to have a small norm. Third, we have devised a new strategy for choosing the degree of the approximants and the number of square roots. We investigated both Padé approximants and Taylor approximants and found that there is very little to choose between them in speed or accuracy.

ACKNOWLEDGMENTS

We thank the anonymous referees, whose comments helped us to improve the presentation of the paper.

BIBLIOGRAPHY

- [1] A. H. AL-MOHY, *A more accurate Briggs method for the logarithm*, Numer. Algorithms, 59 (2012), pp. 393–402.
- [2] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989.
- [3] —, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169.
- [4] A. H. AL-MOHY, N. J. HIGHAM, AND S. D. RELTON, *Computing the Fréchet derivative of the matrix logarithm and estimating the condition number*, SIAM J. Sci. Comput., 35 (2013), pp. C394–C410.

- [5] D. H. BAILEY, H. YOZO, X. S. LI, AND B. THOMPSON, *ARPREC: An arbitrary precision computation package*, tech. report, Lawrence Berkeley National Laboratory, 2002.
- [6] G. A. BAKER, JR. AND P. GRAVES-MORRIS, *Padé Approximants*, vol. 59 of Encyclopedia of Mathematics and Its Applications, Cambridge University Press, Cambridge, UK, 2nd ed., 1996.
- [7] R. H. BARTELS AND G. W. STEWART, *Algorithm 432: Solution of the matrix equation $AX + XB = C$* , Comm. ACM, 15 (1972), pp. 820–826.
- [8] J. BEZANSON, A. EDELMAN, S. KARPINSKI, AND V. B. SHAH, *Julia: A fresh approach to numerical computing*, SIAM Rev., 59 (2017), pp. 65–98.
- [9] Å. BJÖRCK AND S. HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.
- [10] BOOST C++ libraries. <http://www.boost.org>.
- [11] H. BRIGGS, *Arithmetica Logarithmica*, William Jones, London, 1624.
- [12] J. R. CARDOSO AND R. RALHA, *Matrix arithmetic-geometric mean and the computation of the logarithm*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 719–743.
- [13] J. R. CARDOSO AND F. SILVA LEITE, *Theoretical and numerical considerations about Padé approximants for the matrix logarithm*, Linear Algebra Appl., 330 (2001), pp. 31–42.
- [14] S. H. CHENG, N. J. HIGHAM, C. S. KENNEY, AND A. J. LAUB, *Approximating the logarithm of a matrix to specified accuracy*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1112–1125.
- [15] M. COURBARIAUX, Y. BENGIO, AND J.-P. DAVID, *Training deep neural networks with low precision multiplications*, 2015. ArXiv preprint 1412.7024v5.
- [16] P. I. DAVIES AND N. J. HIGHAM, *A Schur–Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 464–485.

- [17] J. E. DENNIS, JR. AND V. TORCZON, *Direct search methods on parallel machines*, SIAM J. Optim., 1 (1991), pp. 448–474.
- [18] L. DIECI, B. MORINI, A. PAPINI, AND A. PASQUALI, *On real logarithms of nearby matrices and structured matrix interpolation*, Appl. Numer. Math., 29 (1999), pp. 145–165.
- [19] N. J. DINGLE AND N. J. HIGHAM, *Reducing the influence of tiny normwise relative errors on performance profiles*, ACM Trans. Math. Software, 39 (2013), pp. 24:1–24:11.
- [20] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*, Math. Programming, 91 (2002), pp. 201–213.
- [21] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, AND P. ZIMMERMANN, *MPFR: A multiple-precision binary floating-point library with correct rounding*, ACM Trans. Math. Software, 33 (2007), pp. 13:1–13:15.
- [22] A. GLASER, X. LIU, AND V. ROKHLIN, *A fast algorithm for the calculation of the roots of special functions*, SIAM J. Sci. Comput., 29 (2007), pp. 1420–1438.
- [23] G. H. GOLUB, S. NASH, AND C. F. VAN LOAN, *A Hessenberg–Schur method for the problem $AX + XB = C$* , IEEE Trans. Automat. Control, AC-24 (1979), pp. 909–913.
- [24] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.
- [25] T. GRANLUND AND THE GMP DEVELOPMENT TEAM, *GNU MP: The GNU multiple precision arithmetic library*. <http://gmplib.org/>.
- [26] S. GUPTA, A. AGRAWAL, K. GOPALAKRISHNAN, AND P. NARAYANAN, *Deep learning with limited numerical precision*, in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37 of JMLR: Workshop and Conference Proceedings, 2015, pp. 1737–1746.
- [27] D. J. HIGHAM AND N. J. HIGHAM, *MATLAB Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, third ed., 2017.

- [28] N. J. HIGHAM, *The Matrix Computation Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mctoolbox>.
- [29] —, *The Matrix Function Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mftoolbox>.
- [30] —, *Evaluating Padé approximants of the matrix logarithm*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1126–1135.
- [31] —, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [32] N. J. HIGHAM AND E. DEADMAN, *A catalogue of software for matrix functions. Version 2.0*, MIMS EPrint 2016.3, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Jan. 2016. Updated March 2016.
- [33] N. J. HIGHAM AND L. LIN, *A Schur–Padé algorithm for fractional powers of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1056–1078.
- [34] —, *An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360.
- [35] N. J. HIGHAM AND F. TISSEUR, *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201.
- [36] W. HU, H. ZUO, O. WU, Y. CHEN, Z. ZHANG, AND D. SUTER, *Recognition of adult images, videos, and web page bags*, ACM Trans. Multimedia Comput. Commun. Appl., 78 (2011), pp. 28:1–28:24.
- [37] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*, Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.
- [38] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008 (revision of IEEE Std 754-1985)*, Institute of Electrical and Electronics Engineers, New York, 2008.

- [39] F. JOHANSSON ET AL., *Mpmath: A Python library for arbitrary-precision floating-point arithmetic*. <http://mpmath.org>.
- [40] C. S. KENNEY AND A. J. LAUB, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209.
- [41] ———, *Padé error estimates for the logarithm of a matrix*, Internat. J. Control, 50 (1989), pp. 707–730.
- [42] *Maple*. Waterloo Maple Inc., Waterloo, Ontario, Canada. <http://www.maplesoft.com>.
- [43] *Mathematica*. Wolfram Research, Inc., Champaign, IL, USA. <http://www.wolfram.com>.
- [44] A. MEURER, C. P. SMITH, M. PAPROCKI, O. ČERTIK, S. B. KIRPICHEV, M. ROCKLIN, A. KUMAR, S. IVANOV, J. K. MOORE, S. SINGH, T. RATHNAYAKE, S. VIG, B. E. GRANGER, R. P. MULLER, F. BONAZZI, H. GUPTA, S. VATS, F. JOHANSSON, F. PEDREGOSA, M. J. CURRY, A. R. TERREL, Š. ROUČKA, A. SABOO, I. FERNANDO, S. KULAL, R. CIMRMAN, AND A. SCOPATZ, *SymPy: Symbolic computing in Python*, PeerJ Computer Science, 3 (2017), p. e103.
- [45] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. <http://www.advanpix.com>.
- [46] *PARI/GP*. <http://pari.math.u-bordeaux.fr>.
- [47] THE SAGE DEVELOPERS, *Sage Mathematics Software*. <http://www.sagemath.org>.
- [48] *Symbolic Math Toolbox*. The MathWorks, Inc., Natick, MA, USA. <http://www.mathworks.co.uk/products/symbolic/>.
- [49] SYMPY DEVELOPMENT TEAM, *Sympy: Python library for symbolic mathematics*. <http://www.sympy.org>.

7

COMPUTING THE WEIGHTED GEOMETRIC MEAN OF TWO LARGE-SCALE MATRICES AND ITS INVERSE TIMES A VECTOR

Abstract. We investigate different approaches for computing the action of the weighted geometric mean of two large-scale positive definite matrices on a vector. We derive and analyze several algorithms, based on numerical quadrature and on the Krylov subspace, and compare them in terms of convergence speed and execution time. By exploiting an algebraic relation between the weighted geometric mean and its inverse, we show how these methods can be used to efficiently solve large linear systems whose coefficient matrix is a weighted geometric mean. According to our experiments, some of the algorithms proposed in both families are suitable choices for black-box implementations.

Keywords: matrix weighted geometric mean, Krylov subspace methods, Gaussian quadrature, matrix functions.

2010 MSC: 15A16, 15A22, 65D32, 47A56.

7.1 INTRODUCTION

The weighted geometric mean of parameter t of two positive numbers, say a and b , is defined as $a^{1-t}b^t$ for any $t \in [0, 1]$. This definition covers as a special case the standard geometric mean \sqrt{ab} , arising for $t = 1/2$. The extension of this concept to

positive definite matrices is not trivial, but there is large agreement that the *right* generalization, for $A, B \in \mathbb{C}^{n \times n}$ (Hermitian) positive definite and $t \in [0, 1]$, is

$$A \#_t B = A(A^{-1}B)^t = A(B^{-1}A)^{-t}, \quad (7.1)$$

which turns out to be positive definite and is called the *matrix weighted geometric mean* of A and B . The reasons behind this choice and the properties of the matrix weighted geometric mean are discussed by Bhatia [11, Chap. 4] and Lawson and Lim [41]. Relevant applications of the weighted geometric mean of two dense matrices of moderate size, along with algorithms for its computations, can be found in the survey [35].

Here we are interested in the approximation of $(A \#_t B)v$ and $(A \#_t B)^{-1}v$, where $v \in \mathbb{C}^n$ and A, B are large and sparse. These problems arise in a preconditioning techniques for some domain decomposition methods and in methods for the biharmonic equation [4], [5], [6], and in the clustering of signed complex networks [44]. The geometric mean of large-scale matrices appears also in image processing [22].

In particular, we want to avoid the explicit computation of the matrix function $A \#_t B$, which may be unduly slow or even practically infeasible, for A and B large enough. We explore two classes of methods to achieve this goal, namely numerical quadrature of certain integral representations of the matrix function Z^{-t} for $t \in (0, 1)$, and Krylov subspace methods for computing the product of a matrix function and a vector.

It is well known that the geometric mean $A \# B := A \#_{1/2} B$ [2], [3], [12], [46] (the weighted geometric mean with weight $t = 1/2$) has several nice integral representations (see [37] and the references therein). In particular, the formula

$$A \# B = \frac{2}{\pi} \int_{-1}^1 \frac{((1+z)B^{-1} + (1-z)A^{-1})^{-1}}{\sqrt{1-z^2}} dz,$$

is well suited for Gaussian quadrature with respect to the weight function $(1-z^2)^{-1/2}$, and is considered in comparison with other algorithms for $A \# B$ by Iannazzo [35]. We generalize this approach to the matrix weighted geometric mean.

Quadrature formulae are particularly attractive in the large-scale case, since they produce an approximation of the form

$$(A\#_t B)v \approx \sum_{i=0}^N w_i A(r_i A + s_i B)^{-1} Bv, \quad (7.2)$$

where the w_i 's are the weights of the quadrature and the r_i 's and the s_i 's are parameters obtained from the nodes of the quadrature. By exploiting the identity $(A\#_t B)^{-1} = B^{-1}(B\#_t A)A^{-1}$, a similar approximation for the inverse of the geometric mean, namely

$$(A\#_t B)^{-1}v \approx \sum_{i=0}^N w_i (r_i B + s_i A)^{-1}v, \quad (7.3)$$

can be easily derived. The problem is thus reduced to the solution of linear systems and the evaluation of matrix-vector products. Moreover, if r_i and s_i are positive for all i , then the matrix coefficients of these linear systems are positive definite, being convex combinations of the positive definite matrices A and B , and we say that the quadrature formula *preserves the positivity structure* of the problem.

We consider and analyze three quadrature formulae for $A\#_t B$. The first two are obtained from integral representations of the inverse of real powers [14], [23], by exploiting the fact that $A\#_t B = A(B^{-1}A)^{-t}$. The third is based on a clever conformal mapping [30], which achieves fast convergence speed but does not preserve the positivity structure of the problem for $t \neq 1/2$.

Regarding Krylov subspace methods, we adapt to our problem standard techniques for the approximation of $f(Z^{-1}Y)v$, where Z and Y are large-scale matrices. In this case, the usual way to proceed is to consider a projection of the matrix onto a small Krylov subspace and thereby reduce the original problem to a small sized one. Since $(A\#_t B)v = A(B^{-1}A)^{-t}v$, the computation of $(A\#_t B)v$ reduces to that of $(B^{-1}A)^{-t}v$, which is well suited to the aforementioned techniques. For instance, when approximating $(B^{-1}A)^{-t}v$ by means of the Arnoldi method, we get the generalized Lanczos method [45, Chap. 15], which has been considered for $(A\#_t B)v$ in previous work [4], [5]. We revise the generalized Lanczos method and then investi-

gate some more powerful Krylov subspace techniques such as the extended Krylov subspace method [21] and the rational Krylov subspace methods [48], [49], [50], with poles chosen according to the adaptive strategy by Güttel and Knizhnerman [29] or the rational Krylov fitting by Berljafa and Güttel [8]. We show that these methods, in most cases, outperform the generalized Lanczos algorithm. Prior to our work, rational Krylov methods have been considered for the computation of $(A\#B)v$, where the implementations are meant for and tested on sparse matrices of moderate size [15].

For the sake of generality, in describing the Krylov subspace techniques, we work with the more general problem $Af(A^{-1}B)v$, where A is positive definite, B is Hermitian and f is the matrix extension of a real positive function. Our implementations, tailored for the function $f(z) = z^{-t}$, are well suited to the computation of $(A\#_t B)^{-1}v$, and could, in principle, be used for any choice of the function f .

The paper is organized as follows. In the next section we give some notation and preliminary results. Quadrature methods for the weighted geometric mean are discussed in section 7.3, while section 7.4 is devoted to Krylov subspace methods. The application of these techniques to the solution of the linear system $(A\#_t B)y = v$ is discussed in section 7.5, and an experimental comparison is provided in section 7.6. In the final section, we draw the conclusions.

7.2 NOTATION AND PRELIMINARIES

Throughout the paper we denote by I_n the identity matrix of size n , omitting the size when there is no ambiguity. The set \mathbb{R}^+ will denote the positive real numbers, while $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. We will denote by $\sigma(A)$ the spectrum of the square matrix A . Throughout the paper, we consider the spectral norm $\|A\| = \max_{\|x\|_2=1} \|Ax\|_2$. For $x_1, \dots, x_n \in \mathbb{C}$, we denote by $\text{diag}(x_1, \dots, x_n)$ the $n \times n$ diagonal matrix with x_1, \dots, x_n on the main diagonal. Let $\mathcal{V} \subset \mathbb{C}^n$ be a subspace, and $A \in \mathbb{C}^{n \times n}$, by $A\mathcal{V}$ we denote the subspace $\{Av : v \in \mathcal{V}\}$.

Let $A \in \mathbb{C}^{n \times n}$ be diagonalizable with eigenvalues in $\Omega \subset \mathbb{C}$ and let $f : \Omega \rightarrow \mathbb{C}$. If $M^{-1}AM = \text{diag}(\lambda_1, \dots, \lambda_n)$, then $f(A) := M \text{diag}(f(\lambda_1), \dots, f(\lambda_n))M^{-1}$. Note that if A is Hermitian, then $f(A)$ is Hermitian as well. This definition can be extended to nondiagonalizable matrices [33, Def. 1.2], and is independent of the choice of M .

We have the *similarity invariance of matrix functions*, that is, if $f(A)$ is well defined, then $f(KAK^{-1}) = Kf(A)K^{-1}$, for any invertible K . We now give a well-known property regarding an expression commonly encountered when dealing with functions of Hermitian matrices.

Lemma 7.1. *Let $f : \mathcal{U} \rightarrow \mathbb{R}^+$, with \mathcal{U} subset of \mathbb{R} . For any $A \in \mathbb{C}^{n \times n}$ positive definite and $B \in \mathbb{C}^{n \times n}$ Hermitian, such that $\sigma(A^{-1}B) \subset \mathcal{U}$, the matrix $Af(A^{-1}B)$ is Hermitian positive definite.*

Proof. Note that $f(A^{-1}B)$ is well defined, since $A^{-1}B$ is diagonalizable with spectrum in \mathcal{U} . Because of the similarity invariance of matrix functions, we have that $Af(A^{-1}B) = A^{1/2}f(A^{-1/2}BA^{-1/2})A^{1/2}$. The matrix $A^{-1/2}BA^{-1/2}$ is Hermitian, thus $T = f(A^{-1/2}BA^{-1/2})$ is Hermitian with positive eigenvalues and the same holds for $Af(A^{-1}B)$, which is obtained from T through a congruence. \square

If A and B are positive definite, then $\sigma(A^{-1}B) \subset \mathbb{R}^+$. Thus, the previous lemma, applied to $f(z) = z^t$, with $\mathcal{U} = \mathbb{R}^+$, shows that $A\#_t B = A(A^{-1}B)^t$ is positive definite. Using other properties of matrix functions one obtains the following equivalent expressions:

$$\begin{aligned} A\#_t B &= A(A^{-1}B)^t = A(B^{-1}A)^{-t} = B(A^{-1}B)^{t-1} = B(B^{-1}A)^{1-t}, \\ &= (BA^{-1})^t A = (AB^{-1})^{-t} A = (BA^{-1})^{t-1} B = (AB^{-1})^{1-t} B. \end{aligned} \tag{7.4}$$

Another useful property of the weighted geometric mean is

$$(A\#_t B)^{-1} = B^{-1}(B\#_t A)A^{-1}, \tag{7.5}$$

which follows from an algebraic manipulation of the formulae in (7.4)

$$(A\#_t B)^{-1} = ((BA^{-1})^{t-1} B)^{-1} = B^{-1}(BA^{-1})^{1-t} AA^{-1} = B^{-1}(B\#_t A)A^{-1}.$$

7.3 QUADRATURE METHODS

In this section, we exploit the formula $A\#_t B = A(B^{-1}A)^{-t}$ to obtain three quadrature formulae for $A\#_t B$ from the corresponding quadrature formulae for the inverse real power function z^{-t} .

In the next subsection we describe and analyze two integral representations for z^{-t} and in sections 7.3.2 and 7.3.3 we discuss their application to the matrix weighted geometric mean. Finally, in section 7.3.4 we adapt an algorithm based on a conformal map transformation to the matrix weighted geometric mean.

7.3.1 Integral representations for z^{-t}

Since $A\#_t B = A(B^{-1}A)^{-t}$, useful integral representations of the matrix weighted geometric mean can be obtained from the representations of the fractional inverse power function. The function $\mathbb{C} \setminus [-\infty, 0] \ni z \rightarrow z^{-t}$ for $t \in (0, 1)$ is a Markov function [10, p. 116], which can be written as

$$z^{-t} = \frac{\sin(\pi t)}{\pi} \int_0^\infty \frac{dx}{x^t(x+z)}, \quad 0 < t < 1. \quad (7.6)$$

To rewrite this integral in a more practical form, we exploit the Cayley transform $\mathcal{C}(x) = \frac{1-x}{1+x}$, which sends the positive real numbers to the interval $(-1, 1)$.

The variable transformation $s = \mathcal{C}(x)$ gives

$$z^{-t} = \frac{2 \sin(\pi t)}{\pi} \int_{-1}^1 (1-s)^{-t} (1+s)^{t-1} \frac{ds}{(1-s) + (1+s)z}. \quad (7.7)$$

On the other hand, by applying the transformation $s = -\mathcal{C}(x^{1-t})$ to the integral in (7.6), we obtain

$$z^{-t} = \frac{2 \sin(\pi(1-t))}{\pi(1-t)} \int_{-1}^1 (1-s)^{\frac{2t-1}{1-t}} \frac{ds}{(1+s)^{\frac{1}{1-t}} + (1-s)^{\frac{1}{1-t}} z}, \quad (7.8)$$

which has been considered in a similar form in order to compute the p th root [14].

Both (7.7) and (7.8) are integrals of the form

$$\int_{-1}^1 (1-s)^\alpha (1+s)^\beta f(s) ds,$$

with $(\alpha, \beta) = (-t, t-1)$ and $(\alpha, \beta) = (\frac{2t-1}{1-t}, 0)$, respectively. These integrals, for $\alpha, \beta > -1$, can be approximated by using Gaussian quadrature with respect to the weight

$$\omega_{\alpha, \beta}(s) = (1-s)^\alpha (1+s)^\beta, \quad s \in [-1, 1]. \quad (7.9)$$

These formulae are known as the Gauss–Jacobi quadrature formulae [47, Sec. 4.8].

A nice feature of the Gauss–Jacobi quadrature applied to the integral (7.7) is that the function to be integrated with respect to the weighted measure, namely

$$f_{1,z}(s) = \frac{1}{1-s+(1+s)z}, \quad (7.10)$$

is analytic on $[-1, 1]$, for any $z \in \mathbb{C} \setminus (-\infty, 0)$, and thus the convergence of the quadrature formulae is exponential.

In particular, given a function f analytic on the interval $[-1, 1]$, for the error of the Gaussian quadrature with nodes s_i and weights w_i for $i = 0, \dots, N-1$, we have the estimate [25], [53]

$$|R_N(f)| = \left| \int_{-1}^1 f(x) \omega(x) dx - \sum_{i=0}^{N-1} w_i f(s_i) \right| \leq 4\mu_0 \frac{1}{\rho^{2N}} \left(\frac{\rho^2}{\rho^2 - 1} \right) \max_{x \in \Gamma} |f(x)|, \quad (7.11)$$

where $\mu_0 = \int_{-1}^1 \omega(x) dx$ and the curve Γ is an ellipse with foci -1 and 1 and sum of the semimajor and semiminor axes ρ , entirely enclosed (with its interior part) in the domain of analyticity of f .

When f is analytic on $[-1, 1]$, we may assume that $\rho > 1$. Hence, for any ellipse contained in the region of analyticity corresponding to ρ , the convergence of the quadrature formula is exponential with rate γ such that $1/\rho^2 < \gamma < 1$. On the other hand, for the integral (7.8), the integrand is

$$f_{2,z}(s) = \frac{1}{(1+s)^{\frac{1}{1-t}} + (1-s)^{\frac{1}{1-t}} z}, \quad (7.12)$$

which is analytic on $[-1, 1]$ for any $z \in \mathbb{C} \setminus (-\infty, 0)$ only if t is of the form $(p-1)/p$, with $p \in \mathbb{N}$. When $1/(1-t)$ is not an integer, the integrand (7.12) has two branch points at -1 and 1 , which makes the use of this second quadrature method less attractive for our purposes. Nevertheless, in some cases the Gauss–Jacobi quadrature applied to (7.8) converges faster than the same method applied to (7.7).

We analyze the convergence just for $z \in \mathbb{R}^+$, because we want to apply the formulae to diagonalizable matrices having positive real eigenvalues and, in this case, the convergence of the quadrature formulae for the matrix follows from that of the same formulae for its eigenvalues.

Convergence for the integrand $f_{1,z}(s)$. Let us start by considering the quadrature formula for $f_{1,z}(s)$, which has only one pole at $\zeta = 1/C(z)$. The function $1/C(z)$ maps the half line $(0, \infty)$ to $\overline{\mathbb{R}} \setminus [-1, 1]$, thus we are guaranteed that the pole lies outside the interval $[-1, 1]$ for any $z > 0$ and that the convergence result for analytic functions applies.

If $z \in (0, \infty)$, then it is easy to identify the smallest ellipse not contained in the domain of analyticity of $f_{1,z}(s)$ as the one passing through ζ . The real semiaxis of such an ellipse has length $|\zeta|$ and its imaginary semiaxis has length $\sqrt{\zeta^2 - 1}$, thus, the sums of its semiaxes is

$$\begin{aligned} \rho^{(1)}(z) &= |\zeta| + \sqrt{\zeta^2 - 1} = \frac{1}{|C(z)|} + \sqrt{\frac{1}{C(z)^2} - 1} \\ &= \frac{|1+z| + 2\sqrt{z}}{|1-z|} = \frac{1 + \sqrt{z}}{|1 - \sqrt{z}|} = \frac{1}{|C(\sqrt{z})|}, \end{aligned} \tag{7.13}$$

and hence a lower bound for the rate of convergence is $|C(\sqrt{z})|^2$.

Convergence for the integrand $f_{2,z}(s)$. The convergence analysis for $f_{2,z}(s)$ is more problematic, since the function lacks analyticity at 1 and -1 when $1/(1-t) \notin \mathbb{N}$. For $t = (p-1)/p$, with $p \in \mathbb{N}$, the function $f_{2,z}(s)$ is rational and its poles are given by the solutions of the equation

$$(1 + \zeta)^p + (1 - \zeta)^p z = 0,$$

which are the p distinct points

$$\zeta_\ell = -\mathcal{C}(z^{1/p} e^{\frac{1}{p}i\pi(2\ell+1)}), \quad \ell = 0, \dots, p-1. \quad (7.14)$$

Since none of them lies on the interval $[-1, 1]$, the integrand is analytic there.

In order to get the rate of convergence of the quadrature formula, we consider the sum of the semiaxes of the smallest ellipse not contained in the domain of analyticity of $f_{2,z}(s)$.

Proposition 7.2. *For any positive integer p , the smallest ellipse not contained in the domain of analyticity of $f_{2,z}(s)$ (defined in (7.12)), with $t = (p-1)/p$, passes through ζ_0 (defined in (7.14)) and the sum of its semiaxes is*

$$\rho^{(2)}(z) = \frac{1 + z^{1/p} + \sqrt{2z^{1/p}(1 - \cos(\pi/p))}}{\sqrt{1 + z^{2/p} + 2z^{1/p} \cos(\pi/p)}}. \quad (7.15)$$

Proof. We know that the poles of $f_{2,s}(z)$ are $\zeta_\ell = -\mathcal{C}(\xi_\ell)$ with $\xi_\ell = z^{\frac{1}{p}} e^{\frac{2\ell+1}{p}i\pi}$, for $\ell = 0, \dots, p-1$.

We want to find the smallest sum of the semiaxes of an ellipse not including the points $\{\zeta_\ell\}$ in its interior part, and with foci 1 and -1 . If we denote by x the length of the major semiaxis of such an ellipse, then the sum of the length of the semiaxes is $\rho = x + \sqrt{x^2 - 1}$.

We know that the sum of the distances between a point of the ellipse and the foci is twice the major semiaxis. To find the major semiaxis of the ellipse passing through ζ_ℓ we can use the fact that

$$|\zeta_\ell - 1| + |\zeta_\ell + 1| = 2x_\ell,$$

which readily gives x_ℓ and thus ρ_ℓ .

Since $\zeta_\ell = -\mathcal{C}(\xi_\ell)$, we have

$$\zeta_\ell + 1 = \frac{2\xi_\ell}{\xi_\ell + 1}, \quad \zeta_\ell - 1 = \frac{-2}{\xi_\ell + 1}, \quad x_\ell = \frac{1}{2}(|\zeta_\ell + 1| + |\zeta_\ell - 1|) = \frac{|\xi_\ell| + 1}{|\xi_\ell + 1|},$$

from which, by using $|\xi_\ell| = z^{1/p}$ and $(|\xi| + 1)^2 - |\xi + 1|^2 = 2|\xi| - 2\operatorname{re} \xi$, we get

$$\rho_\ell = x_\ell + \sqrt{x_\ell^2 - 1} = \frac{|\xi_\ell| + 1 + \sqrt{2|\xi_\ell| - 2\operatorname{re} \xi_\ell}}{|\xi_\ell + 1|} = \frac{1 + z^{1/p} + \sqrt{2z^{1/p}(1 - \cos(\vartheta_\ell))}}{\sqrt{1 + z^{2/p} + 2z^{1/p} \cos(\vartheta_\ell)}},$$

where $\vartheta_\ell = \frac{2\ell+1}{p}\pi$. Now observe that ρ_ℓ decreases as $\cos(\vartheta_\ell)$ grows, and thus that the closer ϑ_ℓ is to a multiple of 2π , the smaller is the value of ρ_ℓ . Noting that ϑ_0 is the nearest such value concludes the proof. \square

Hence, for $t = (p-1)/p$, we have a lower bound for the rate of convergence, namely $(1/\rho^{(2)}(z))^2$. For $t \neq (p-1)/p$, by lack of analyticity of the integrand, we cannot use these asymptotic results to study the convergence of the quadrature formula involving $f_{2,z}(s)$. Nonetheless, it appears that the formula converges also for values of t not of the type $(p-1)/p$.

Comparison. We can compare the bounds for the rates of convergence of the two quadrature formulae, namely $(1/\rho^{(1)}(z))^2$, with $\rho^{(1)}(z)$ defined as in (7.13); and $(1/\rho^{(2)}(z))^2$, with $\rho^{(2)}(z)$ given by (7.15), just for $t = (p-1)/p$. Since $\rho^{(1)}(1/z) = \rho^{(1)}(z)$ and $\rho^{(2)}(1/z) = \rho^{(2)}(z)$, we can restrict our attention to $z \geq 1$.

In a neighborhood of 1, the quadrature formula using $f_{1,z}(s)$ works better since $1/\rho^{(1)}(1) = 0$, while $1/\rho^{(2)}(1) > 0$.

On the other hand, as $z \rightarrow \infty$, we have

$$1 - \left(\frac{1}{\rho^{(1)}(z)}\right)^2 \approx 4z^{-\frac{1}{2}}, \quad 1 - \left(\frac{1}{\rho^{(2)}(z)}\right)^2 \approx 2\sqrt{2(1 - \cos(\pi/p))}z^{-\frac{1}{2p}}. \quad (7.16)$$

and thus the second formula works better for large values of z .

Gauss–Jacobi quadrature and Padé approximation. Quadrature on Markov functions is related to Padé approximation. In particular, applying the Gauss–Jacobi quadrature to the integral in (7.7) yields the $[N-1/N]$ Padé approximant to z^{-t} at 1. We give a short proof of this property (see also that given by Frommer, Güttel, and Schweitzer [23]).

Theorem 7.3. *The N -node Gauss–Jacobi quadrature of (7.7) coincides with the $[N - 1/N]$ Padé approximant to z^{-t} at 1.*

Proof. The Gaussian quadrature formula with N nodes, say $\mathcal{J}_N(z)$, is a rational function of z whose numerator and denominator have degree at most $N - 1$ and exactly N , respectively.

We have that $f_{1,z}^{(k)}(s) = (-1)^k k! (z - 1)^k f_{1,z}^{k+1}(s)$ for $k \geq 0$. From the latter and using standard results on the remainder of Gaussian quadrature we have that there exists $\xi = \xi(z) \in (-1, 1)$ such that

$$z^{-t} - \mathcal{J}_N(z) = \frac{2 \sin(\pi t)}{\pi} \frac{f_{1,z}^{(2N)}(\xi)}{(2N)!} \langle P_N^{(-t, 1-t)}, P_N^{(-t, 1-t)} \rangle = c_n \frac{(z - 1)^{2N}}{(z - 1)\xi + (z + 1)},$$

where $P_N^{(\alpha, \beta)}$ is the N th Jacobi polynomial, $\langle \cdot, \cdot \rangle$ is the scalar product with respect to the weight (7.9) and c_n is a constant independent of z .

As $z \rightarrow 1$ we get that $z^{-t} - \mathcal{J}_N(z) = O((z - 1)^{2N})$ and thus $\mathcal{J}_N(z)$ is the $[N - 1, N]$ Padé approximant to z^{-t} . \square

7.3.2 Integral representations of $A \#_t B$

The integral representations in section 7.3.1 for z^{-t} readily yield analogous representations for the matrix weighted geometric mean (through $A \#_t B = A(B^{-1}A)^{-t}$).

From the formula (7.7) we obtain

$$\begin{aligned} A \#_t B &= c_1 A \int_{-1}^1 (1 - s)^{-t} (1 + s)^{t-1} ((1 - s)I + (1 + s)B^{-1}A)^{-1} ds \\ &= c_1 A^{1/2} \int_{-1}^1 (1 - s)^{-t} (1 + s)^{t-1} ((1 - s)I + (1 + s)A^{1/2}B^{-1}A^{1/2})^{-1} ds \cdot A^{1/2} \\ &= c_1 A \int_{-1}^1 (1 - s)^{-t} (1 + s)^{t-1} ((1 - s)B + (1 + s)A)^{-1} B ds, \end{aligned} \quad (7.17)$$

with $c_1 = \frac{2 \sin(\pi t)}{\pi}$, and the corresponding quadrature formula on $N + 1$ nodes gives

$$A \#_t B \approx S_{N+1}^{(1)} := \frac{2 \sin(\pi t)}{\pi} \sum_{i=0}^N w_i A((1 - s_i)B + (1 + s_i)A)^{-1} B, \quad (7.18)$$

where the w_i s are the weights of the Gauss–Jacobi quadrature formula with $N + 1$ nodes and s_i s are the nodes, which belong to the interval $[-1, 1]$. Therefore, for $i = 0, \dots, N$, the matrix $(1 - s_i)B + (1 + s_i)A$ is positive definite.

On the other hand, from (7.8) we have

$$\begin{aligned} A\#_t B &= c_2 A \int_{-1}^1 (1-s)^{\frac{2t-1}{1-t}} \left((1+s)^{\frac{1}{1-t}} I + (1-s)^{\frac{1}{1-t}} B^{-1} A \right)^{-1} ds \\ &= c_2 A^{1/2} \int_{-1}^1 (1-s)^{\frac{2t-1}{1-t}} \left((1+s)^{\frac{1}{1-t}} I + (1-s)^{\frac{1}{1-t}} A^{1/2} B^{-1} A^{1/2} \right)^{-1} ds \cdot A^{1/2} \\ &= c_2 A \int_{-1}^1 (1-s)^{\frac{2t-1}{1-t}} \left((1+s)^{\frac{1}{1-t}} B + (1-s)^{\frac{1}{1-t}} A \right)^{-1} B ds, \end{aligned} \quad (7.19)$$

with $c_2 = \frac{2 \sin(\pi(1-t))}{\pi(1-t)}$, and the corresponding quadrature formula with $N + 1$ nodes gives

$$A\#_t B \approx S_{N+1}^{(2)} := \frac{2 \sin(\pi(1-t))}{\pi(1-t)} \sum_{i=0}^N w_i A \left((1+s_i)^{\frac{1}{1-t}} B + (1-s_i)^{\frac{1}{1-t}} A \right)^{-1} B. \quad (7.20)$$

Even in this case the matrices *to be inverted*, for $i = 0, \dots, N$, are positive definite.

7.3.3 Matrix convergence

In order to analyze the convergence of quadrature formulae for the matrix weighted geometric mean, we consider the convergence of the quadrature formulae for (7.7) and (7.8) when applied to a Hermitian positive definite matrix C . In this case, the functions to be integrated are

$$f_{1,C}(s) = ((1-s)I + (1+s)C)^{-1} \text{ and } f_{2,C}(s) = ((1+s)^{\frac{1}{1-t}} I + (1-s)^{\frac{1}{1-t}} C)^{-1},$$

whose domain of analyticity is the intersection of the domain of analyticity of the corresponding function applied to all the eigenvalues of C .

If $Q^*CQ = \text{diag}(\lambda_1, \dots, \lambda_n)$, with Q unitary, and the function to be integrated is analytic on $[-1, 1]$, then the error in the quadrature formulae with N nodes defined in (7.11), in the spectral norm, is

$$\|R_N(f_{k,C}(s))\| = \|\text{diag}(R_N(f_{k,\lambda_i}(s)))\| = \max_{i=1,\dots,n} \{|R_N(f_{k,\lambda_i}(s))|\}, \quad k = 1, 2,$$

and is ruled by the eigenvalue whose corresponding pole gives the smallest ellipse with foci 1 and -1 , enclosed in the domain of analyticity.

Convergence for the integrand $f_{1,C}(s)$. Let the eigenvalues of C be ordered so that $0 < \lambda_m = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{n-1} \leq \lambda_n = \lambda_M$. The infimum of the acceptable values for ρ (the ellipse parameter) is now obtained by minimizing the function $|\zeta| + \sqrt{\zeta^2 - 1}$ for $\zeta \in \sigma(C)$, where $\sigma(C)$ denotes the spectrum of C , so that the bound on the rate of convergence, in view of (7.13), is

$$\tau^{(1)}(C) = \max_{\lambda \in \sigma(C)} \frac{1}{(\rho^{(1)}(\lambda))^2} = \max_{\lambda \in \sigma(C)} |\mathcal{C}(\sqrt{\lambda})|^2 = \max\{|\mathcal{C}(\sqrt{\lambda_m})|^2, |\mathcal{C}(\sqrt{\lambda_M})|^2\},$$

since the function $|\mathcal{C}(\sqrt{\lambda})|$ is monotonically decreasing in $(0, 1)$ and monotonically increasing in $(1, \infty)$.

Since C is positive definite, its 2-norm condition number, denoted by $\kappa := \mu_2(C)$, is λ_M/λ_m . If we further assume that $\lambda_M\lambda_m = 1$, then $\kappa = \lambda_M^2 = 1/\lambda_m^2$ and since $|\mathcal{C}(\sqrt{\lambda_m})| = |\mathcal{C}(\sqrt{\lambda_M})|$, we have

$$\tau^{(1)}(C) = |\mathcal{C}(\sqrt{\lambda_M})|^2 = \mathcal{C}(\sqrt[4]{\kappa})^2.$$

Expanding $\tau^{(1)}$ as $\kappa \rightarrow \infty$, we get

$$\tau^{(1)}(C) = \left(\frac{\sqrt[4]{\kappa}-1}{\sqrt[4]{\kappa}+1}\right)^2 = \left(1 - \frac{2}{\sqrt[4]{\kappa}+1}\right)^2 \approx 1 - \frac{4}{\sqrt[4]{\kappa}} \approx \exp(-4/\sqrt[4]{\kappa}). \quad (7.21)$$

Note that the condition $\lambda_M\lambda_m = 1$ is not restrictive, since any positive definite matrix verifies it up to scaling, but can significantly accelerate the convergence of these quadrature algorithms for matrices such that $\lambda_M\lambda_m$ is far from 1.

Convergence for the integrand $f_{2,C}(s)$ and comparison. As before, for a positive definite matrix C , a bound for the rate of convergence of the matrix quadrature formula is given by the largest bound on the rate of convergence of the scalar formula applied to the eigenvalues of C .

Since the scalar convergence is complicated by the branch points at 1 and -1 and by the presence of a possibly large number of poles in certain cases, also the matrix convergence is hardly predictable.

Nevertheless, if $\lambda_M \lambda_m = 1$, then for $t = 1/2$ we can get an asymptotic estimate as $\kappa \rightarrow \infty$, which is

$$\tau^{(2)}(C) = \max_{\lambda \in \sigma(C)} \frac{1}{(\rho^{(2)}(\lambda))^2} = \left(\frac{\sqrt{\sqrt{\kappa} + 1}}{\sqrt[4]{\kappa} + 1 + \sqrt{2}\sqrt[8]{\kappa}} \right)^2 \approx 1 - \frac{2\sqrt{2}}{\sqrt[8]{\kappa}}. \quad (7.22)$$

For $t = 1/2$, it can be shown, moreover, that the Gauss–Jacobi quadrature of (7.8) is better than that of (7.7) for

$$|z| \in \mathbb{R} \setminus \left[\frac{1}{\xi}, \xi \right], \quad \xi = 2 + \sqrt{5} + 2\sqrt{2 + \sqrt{5}} \approx 8.35,$$

and this is confirmed by the results of Test 1 in section 7.6. Thus, for a positive definite matrix and for $t = 1/2$, unless the matrix is very well conditioned/preconditioned ($\kappa_2(C) \lesssim 70$), the method based on (7.19) is preferable.

Application to the weighted geometric mean. In the case of the weighted geometric mean, in view of equations (7.17) and (7.19), the functions to be integrated are $f_{1,C}(s)$ and $f_{2,C}(s)$, with $C = A^{1/2}B^{-1}A^{1/2}$, so that the previous analysis for a positive definite matrix C can be applied.

Let λ_M and λ_m be the largest and smallest eigenvalues of $A^{1/2}B^{-1}A^{1/2}$ (or of the pencil $A - \lambda B$), respectively. A *scaling* of A and/or B would change the weighted geometric mean in a simple, predictable way, since [41]

$$(\alpha A) \#_t (\beta B) = \alpha^{1-t} \beta^t (A \#_t B).$$

Thus, we may assume that $\lambda_M \lambda_m = 1$ and replace the pair (A, B) with (\hat{A}, \hat{B}) , where $\hat{A} = A/\sqrt{\lambda_M \lambda_m}$ and $\hat{B} = B$.

The quadrature formulae $S_N^{(1)}$ of (7.18) converges at least linearly to $\hat{A} \#_t \hat{B}$, and we get the following estimate

$$\|\hat{A} \#_t \hat{B} - S_N^{(1)}\| = O(e^{-4N/\sqrt[4]{\kappa}}); \quad (7.23)$$

while we have that $S_N^{(2)}$ of (7.20), for $t = 1/2$, converges at least linearly to $\hat{A} \#_{1/2} \hat{B}$, and we get the estimate

$$\|\hat{A} \#_{1/2} \hat{B} - S_N^{(2)}\| = O(e^{-2\sqrt{2}N/\sqrt[8]{\kappa}}). \quad (7.24)$$

7.3.4 An alternative quadrature formula

Another powerful quadrature formula for real matrix powers has been obtained in [30] by applying a few variable substitutions on the Cauchy formula for z^{-t} .

Without giving any further details, we report the results of interest from the original paper [30], referring the reader to it for a complete explanation. Let the function $f : \mathbb{C} \setminus (-\infty, 0] \rightarrow \mathbb{C}$ be analytic and let us assume that $(-\infty, 0)$ is a branch cut for f and that 0 is the only singularity, if any. Under these assumptions, the approximation of $f(Z)$, where Z is a real square matrix with positive eigenvalues, using a quadrature formula with N nodes is given by

$$\frac{-8K(k)Z\sqrt[4]{\lambda_m \lambda_M}}{\pi N k} \operatorname{Im} \left(\sum_{j=1}^N \frac{f(w(t_j)^2) \operatorname{cn}(t_j) \operatorname{dn}(t_j)}{w(t_j)(k^{-1} - \operatorname{sn}(t_j))^2} (w(t_j)^2 I - Z)^{-1} \right), \quad (7.25)$$

where λ_m and λ_M are the minimum and maximum of the spectrum, respectively, $k = -\mathcal{C}(\sqrt[4]{\lambda_M/\lambda_m})$, $K(\ell)$ is the complete elliptic integral associated with ℓ [30],

$$w(t) = \sqrt[4]{\lambda_m \lambda_M} \frac{k^{-1} + \operatorname{sn}(t)}{k^{-1} - \operatorname{sn}(t)}, \quad t_j = -K(k^2) + \frac{i}{2}K(1 - k^2) + \frac{2j-1}{N}K(k^2),$$

for $1 \leq j \leq N$ and $\text{cn}(\cdot)$, $\text{dn}(\cdot)$ and $\text{sn}(\cdot)$ are Jacobi elliptic functions in standard notation (see [1]). The theoretical aspects of these functions can be found in the book by Driscoll and Trefethen [20].

This method can be easily adapted for computing $Af(A^{-1}B)v$, when $A^{-1}B$ is real with positive eigenvalues, without forming explicitly A^{-1} , providing

$$\frac{-8K(k^2)\sqrt[4]{\lambda_m\lambda_M}}{\pi Nk} B \operatorname{Im} \left(\sum_{j=1}^N \frac{f(w(t_j)^2) \operatorname{cn}(t_j) \operatorname{dn}(t_j)}{w(t_j)(k^{-1} - \operatorname{sn}(t_j))^2} (w(t_j)^2 A - B)^{-1} A \right) v, \quad (7.26)$$

which does not require any matrix product or inversion if evaluated from right to left.

Using the identity $A\#_t B = A(A^{-1}B)^t$, for the matrix geometric mean of real positive definite matrices, one gets the approximation $A\#_t B \approx S_N^{(3)}$ with

$$S_N^{(3)} := \frac{-8K(k^2)\sqrt[4]{\lambda_m\lambda_M}}{\pi Nk} A \operatorname{Im} \left(\sum_{j=1}^N \frac{w(t_j)^{2t-1} \operatorname{cn}(t_j) \operatorname{dn}(t_j)}{(k^{-1} - \operatorname{sn}(t_j))^2} (w(t_j)^2 A - B)^{-1} \right) B. \quad (7.27)$$

which is of the form (7.2) with $r_i = w(t_i)^2$ and $s_i = -1$. Unfortunately, for $t \neq 1/2$, the matrices $r_i A + s_i B$ can be complex and not positive definite, for some values of i .

The quadrature formula $S_N^{(3)}$ of (7.27) converges linearly to $A\#_t B$, in particular the following estimate can be deduced from [30, Thm. 3.1]

$$\|A\#_t B - S_N^{(3)}\| = O(e^{-2\pi^2 N/(\log(\kappa)+6)}),$$

where $\kappa = \lambda_M/\lambda_m$, with λ_m and λ_M the smallest and largest eigenvalues of $A^{-1}B$, respectively. A comparison with the analogous results for the two quadrature formulae of section 7.3.2, namely (7.21) and (7.22), suggests that this formula can converge much faster when λ_M/λ_m becomes very large and this is confirmed by the experiments in section 7.6.

7.4 KRYLOV SUBSPACE METHODS

In this section we address the problem of approximating $(A\#_t B)v = A(A^{-1}B)^t v$, using methods based on Krylov subspaces. The approach is similar to the one used in the well-developed problem of approximating $f(C)v$, where C is a large and sparse matrix (see, for instance, [24, sect. 3] or [32, Chap. 13]). However, the fact that $C = A^{-1}B$, with A and B positive definite, requires certain additional subtleties, such as the convenience of orthogonalizing with respect to a non-Euclidean scalar product. We will refer to the resulting methods as generalized Krylov methods.

We will describe first the generalized Lanczos method in section 7.4.1, then the generalized Extended Krylov method in section 7.4.2 and finally the generalized rational Krylov methods in section 7.4.3. Some convergence issues are addressed in section 7.4.4.

The algorithms are presented for the more general problem $Af(A^{-1}B)v$, where $f : \mathcal{U} \rightarrow \mathbb{R}^+$, with \mathcal{U} an open subset of \mathbb{R} , the matrix A is positive definite and B is Hermitian, with $\sigma(A^{-1}B) \subset \mathcal{U}$.

7.4.1 Generalized Arnoldi and Lanczos methods

Let $A, M \in \mathbb{C}^{n \times n}$ be positive definite and let $B \in \mathbb{C}^{n \times n}$ be Hermitian. The generalized Arnoldi method generates a sequence of M -orthonormal vectors $\{v_k\}_{k=1}^n$ and a sequence of upper Hessenberg matrices $\{H_k\}_{k=1}^n$ with $H_k \in \mathbb{C}^{k \times k}$, such that the columns of $V_k := [v_1 | \dots | v_k] \in \mathbb{C}^{n \times k}$ span an M -orthonormal basis of the Krylov subspace

$$\mathcal{K}_k(A^{-1}B, v) = \text{span}\{v, (A^{-1}B)v, \dots, (A^{-1}B)^{k-1}v\}, \quad (7.28)$$

where $v_1 = v/\|v\|_M$ and the elements of H_k , defined by $h_{ij} = v_i^* M A^{-1} B v_j$, turn out to be the coefficients of the Gram–Schmidt orthogonalization process [27, sect. 9.4.1], with respect to the scalar product defined by M . The algorithm has a breakdown when, for some $j \leq n$, we have $v_j \in \text{span}\{v_1, \dots, v_{j-1}\}$.

If no breakdown occurs, the matrices V_n and H_n produced by the algorithm satisfy $V_n^* M V_n = I_n$, $B V_n = A V_n H_n$ and, for $k < n$,

$$B V_k = A V_k H_k + h_{k+1,k} A v_{k+1} e_k^*, \quad (7.29)$$

where e_k is the last column of $I_k \in \mathbb{C}^{k \times k}$.

It is well known [33, Chap. 13] that equation (7.29) can be readily exploited to compute an approximation of $f(A^{-1}B)v$. If $Q V_k = V_k U$, where $Q, U \in \mathbb{C}^{n \times n}$ and $V \in \mathbb{C}^{n \times k}$, then, it can be proved that $f(Q) V_k = V_k f(U)$. Thus, by imposing that $B V_k \approx A V_k H_k$, we can write

$$f(A^{-1}B) V_k \approx V_k f(H_k),$$

and by observing that $v = v_1 \|v\|_M = V_k e_1 \|v\|_M$, we obtain that

$$A f(A^{-1}B) v = A f(A^{-1}B) V_k e_1 \|v\|_M \approx A V_k f(H_k) e_1 \|v\|_M, \quad (7.30)$$

a relation that is useful, in practice, only when the approximation is good for k much smaller than n .

We discuss now the options for the matrix defining the inner product used in the Arnoldi process. Following the recommendation of Parlett [45, Chap. 15], Arioli and Loghin [5] develop an algorithm to approximate $(A \#_t B) v$ using $M = A$. It is immediate to see that, in this case, H_k is tridiagonal, in being both upper Hessenberg and Hermitian, since $H_k = V_k^* B V_k$. Thus, the generalized Arnoldi process becomes a generalized Lanczos algorithm, which is superior for two main reasons. On the one hand, the computation of each v_k requires a fixed number of arithmetic operations, which considerably decreases the overall execution time of the algorithm, on the other hand, the evaluation of $f(H_k)$ becomes easier and can be accurately performed by diagonalization, since H_k is normal.

If B is positive definite, then the generalized method for (A, B) admits a minor variation: we can use the Arnoldi process to construct a basis of $\mathcal{K}_k(A^{-1}B, v)$ of (7.28)

which is B -orthonormal. In this case, we get $BV_n = AV_nH$ with $V_n^*BV_n = I_n$ and the matrices $H_k = V_k^*BA^{-1}BV_k$ turn out to be tridiagonal.

In principle, any scalar product associated to a positive definite matrix M could be used in the Arnoldi process to construct a basis of $\mathcal{K}_k(A^{-1}B, v)$, and the sequence of upper Hessenberg matrices H_k . However, if we want H_k to be tridiagonal, we must restrict the choice for M as in the following.

Proposition 7.4. *Let $A, M \in \mathbb{C}^{n \times n}$ be positive definite and $B \in \mathbb{C}^{n \times n}$ be Hermitian, and assume that the Arnoldi process applied to $A^{-1}B$ with starting vector v and orthogonalization with respect to the scalar product induced by M can be applied with no breakdown. Then for $k = 1, \dots, n$, the Hessenberg matrix H_k is Hermitian (and thus tridiagonal) if and only if $MA^{-1}B = BA^{-1}M$.*

Proof. From $H_k = V_k^*MA^{-1}BV_k$, we get that $H_k = H_k^*$ for each k , if and only if $MA^{-1}B = BA^{-1}M$. \square

The previous result shows that, for the problem $Af(A^{-1}B)v$, the customary orthogonalization procedure, that corresponds to the choice $M = I$, can cause loss of structure since H_k is nonsymmetric if A and B do not commute.

7.4.2 Generalized Extended Krylov subspace method

The standard extended Krylov methods [21], [51] can be easily generalized to build an M -orthonormal basis of the extended Krylov subspace

$$\mathcal{E}_k(A^{-1}B, v) = \text{span}\{v, A^{-1}Bv, B^{-1}Av, (A^{-1}B)^2v, \dots, (B^{-1}A)^{\frac{k}{2}-1}v, (A^{-1}B)^{\frac{k}{2}}v\},$$

if k is even and

$$\mathcal{E}_k(A^{-1}B, v) = \text{span}\{v, A^{-1}Bv, B^{-1}Av, (A^{-1}B)^2v, \dots, (A^{-1}B)^{\frac{k-1}{2}}v, (B^{-1}A)^{\frac{k-1}{2}}v\},$$

if k is odd.

As it is the case for the standard Arnoldi algorithm, the extended Krylov algorithm generates a sequence of M -orthonormal vectors $\{v_k\}_{k=1}^n$ and a sequence of Hessenberg matrices with an additional subdiagonal $\{H_k\}_{k=1}^n$ with $H_k \in \mathbb{C}^{k \times k}$. We stress that, in this case, H_k does not contain the orthogonalization coefficients of the Gram–Schmidt process applied to the set $\{v_1, \dots, v_k\}$. The interplay between orthogonalization coefficients and H_k , for the extended Krylov subspace methods, are discussed by Simoncini [51] and Jagels and Reichel [38], [39].

If we define $V_k = [v_1 | \dots | v_k]$ as the M -orthonormal basis of $\mathcal{E}_k(A^{-1}B, v)$, then the matrices produced by the algorithm, if no breakdown occurs, verify $BV_n = AV_n H_n$ and $V_n^* M V_n = I_n$, while for k even and $k < n$

$$BV_k = AV_k H_k + A[v_{k+1} | v_{k+2}] \tilde{H} E_k, \quad (7.31)$$

where $H_k \in \mathbb{C}^{k \times k}$, $\tilde{H} = [v_{k+1} | v_{k+2}]^* M A^{-1} B [v_{k-1} | v_k] \in \mathbb{C}^{2 \times 2}$, $E_k \in \mathbb{C}^{2 \times k}$ contains the last two rows of the identity matrix I_k and $V_k \in \mathbb{C}^{n \times k}$ is the M -orthonormal basis of the extended Krylov subspace at step k .

As in the previous section, we can conclude that $H_k = V_k^* M A^{-1} B V_k$ and thus that Proposition 7.4 remains valid for the extended method. The choice $M = A$, is again the most natural. Moreover, for any $k \leq n$ the function $Af(A^{-1}B)v$ can be approximated by means of

$$Af(A^{-1}B)v \approx AV_k f(H_k) e_1 \|v\|_M, \quad (7.32)$$

where V_k and H_k are the matrices produced by the extended algorithm.

We wish to point out that the Arnoldi decomposition (7.31) is specific to the basis computation approach that adds two vectors at each step [51]. Using the approach of [38, 39], one would have to add to $AV_k H_k$ only one non-zero column rather than two.

7.4.3 Generalized rational Krylov subspace methods

The rational Arnoldi algorithm [48], [50] can be adapted to our problem. Starting with a vector v , a positive definite matrix M , and poles $\xi_1, \dots, \xi_k \in \mathbb{C} \cup \{\infty\}$ such that $\xi_i \notin \sigma(A^{-1}B) \cup \{0\}$, we can construct a basis of the rational Krylov subspaces (we set $1/\infty = 0$)

$$\mathcal{Q}_k(A^{-1}B, v) := \prod_{j=1}^{k-1} \left(I_n - \frac{1}{\xi_j} A^{-1}B \right)^{-1} \text{span}\{v, A^{-1}Bv, \dots, (A^{-1}B)^{k-1}v\},$$

by considering $v_1 = v/\|v\|_M$ and then M -orthogonalizing the vector

$$w_j = (A - B/\xi_j)^{-1}Bv_j,$$

with respect to v_1, \dots, v_j , obtaining

$$h_{ij} = w_j^* M v_i, \quad \widetilde{w}_j = w_j - \sum_{i=1}^j h_{ij} v_i, \quad h_{j+1,j} = \|\widetilde{w}_j\|_M, \quad v_{j+1} = \widetilde{w}_j / h_{j+1,j}.$$

Notice that a breakdown can occur if $\widetilde{w}_j = 0$, that is, $w_j \in \text{span}\{v_1, \dots, v_j\}$.

In this way, if no breakdown occurs, we get the rational Arnoldi decomposition

$$BV_k(I_k + H_k D_k) + \frac{h_{k+1,k}}{\xi_k} Bv_{k+1}e_k^* = AV_k H_k + h_{k+1,k} A v_{k+1} e_k^*, \quad (7.33)$$

where $D_k = \text{diag}(1/\xi_1, \dots, 1/\xi_k)$, H_k is the matrix containing the entries h_{ij} and $V_k = [v_1 | \dots | v_k]$ is an M -orthogonal basis of $\mathcal{Q}_k(A^{-1}B, v)$. Note that we do not allow 0 to be a pole just for ease of exposition; it is possible to build a rational Arnoldi decomposition with a pole at 0 , by using a slightly different definition [8, sect. 3].

If the last pole is at infinity, then (7.33) simplifies to

$$BV_k(I_k + H_k D_k) \approx AV_k H_k$$

and we get the approximation

$$Af(A^{-1}B)v \approx AV_k f(H_k(I_k + H_k D_k)^{-1})e_1 \|v\|_M. \quad (7.34)$$

Notice that in this case $H_k(I_k + H_k D_k)^{-1} = V_k^* M A^{-1} B V_k$, which is Hermitian if M commutes with $A^{-1}B$. Thus, the argument of f is normal and the evaluation can be done by diagonalization.

The Krylov subspaces described in section 7.4.1 and section 7.4.2 are in fact rational Krylov subspaces where the poles are chosen to be ∞ or 0 and ∞ , respectively. In order to achieve a convergence rate faster than that of the previous two algorithms, the choice of poles is crucial, but there is no general recipe. In section 7.6 we use two black-box heuristics which are well-suited to the problem $f(A)b$.

7.4.4 Convergence of Krylov methods

Despite being rather impractical from a computational point of view, the identity $Af(A^{-1}B)v = A^{1/2}f(A^{-1/2}BA^{-1/2})A^{1/2}v$ turns out to be very useful in the analysis of the convergence of the Krylov methods, as we will see.

By exploiting the generalized Arnoldi method, we get an approximation of the form (compare (7.30))

$$Af(A^{-1}B)v \approx AV_k f(H_k)e_1 \|v\|_M =: f_k, \quad (7.35)$$

where V_k is an M -orthogonal basis of the Krylov subspace $\mathcal{K}_k(A^{-1}B, v)$ defined in (7.28) and $H_k = V_k^* M A^{-1} B V_k$.

On the other hand, we can pick a positive definite matrix \widetilde{M} and apply the generalized Lanczos method to compute a matrix $W_k \in \mathbb{C}^{n \times k}$, with \widetilde{M} -orthogonal columns and span the Krylov subspace $\mathcal{K}_k(A^{-1/2}BA^{-1/2}, A^{1/2}v)$, obtaining the approximation

$$Af(A^{-1}B)v = A^{1/2}f(A^{-1/2}BA^{-1/2})A^{1/2}v \approx A^{1/2}W_k f(\widetilde{H}_k)e_1 \|A^{1/2}v\|_{\widetilde{M}} =: g_k, \quad (7.36)$$

with $\tilde{H}_k = W_k^* \tilde{M} A^{-1/2} B A^{-1/2} W_k$.

We will prove that these two approximations are equal for a suitable choice of W_k and \tilde{M} .

Proposition 7.5. *Let $A, B, M \in \mathbb{C}^{n \times n}$ be positive definite and let $v \in \mathbb{C}^n$. If the columns of $V_k \in \mathbb{C}^{n \times k}$ span an M -orthogonal basis of $\mathcal{K}_k(A^{-1}B, v)$, then those of $W_k := A^{1/2}V_k$ span an \tilde{M} -orthogonal basis of $\mathcal{K}_k(A^{-1/2}BA^{-1/2}, A^{1/2}v)$, with $\tilde{M} = A^{-1/2}MA^{-1/2}$ and f_k and g_k , defined in (7.35) and (7.36), respectively, are such that $f_k = g_k$.*

Proof. First, we observe that the columns of W_k are \tilde{M} -orthogonal, since

$$W_k^* \tilde{M} W_k = V_k^* A^{1/2} \tilde{M} A^{1/2} V_k = V_k^* M V_k = I.$$

and that it is a basis of $\mathcal{K}_k(A^{-1/2}BA^{-1/2}, A^{1/2}v)$, since for $\ell = 0, \dots, k-1$ we have that $A^{1/2}(A^{-1}B)^\ell v = (A^{-1/2}BA^{-1/2})^\ell (A^{1/2}v)$. By direct inspection, we can see that $\|A^{1/2}v\|_{\tilde{M}}^2 = v^* A^{1/2} \tilde{M} A^{1/2} v = v^* M v = \|v\|_M^2$ and

$$\tilde{H}_k = W_k^* \tilde{M} A^{-1/2} B A^{-1/2} W_k = V_k^* A^{1/2} \tilde{M} A^{1/2} A^{-1} B V_k = V_k^* M A^{-1} B V_k = H_k,$$

from which we obtain

$$g_k = A^{1/2} W_k f(\tilde{H}_k) e_1 \|A^{1/2}v\|_{\tilde{M}} = A V_k f(H_k) e_1 \|v\|_M = f_k. \quad \square$$

Observe that for $M = A$, we have $\tilde{M} = I$, which gives yet another reason for making this choice.

The previous equivalence is true also for rational Krylov subspaces (and in particular, for extended Krylov subspaces), because approximating $(A \#_t B)v$ in the space $\mathcal{Q}_k(A^{-1}B, v)$ is equivalent to constructing a sequence of approximations to the same quantity in $\mathcal{Q}_k(A^{-1/2}BA^{-1/2}, A^{1/2}v)$.

The equivalence of approximations allows one to estimate the convergence of Krylov methods using the convergence results for functions of positive definite matrices, which are simpler than those for general matrices.

For instance, if \tilde{f}_k is the approximation of $(A\#_t B)v$ in the extended Krylov subspace $\mathcal{E}_k(A^{-1}B, v)$, using the error bound from [40], we obtain

$$\|(A\#_t B)v - \tilde{f}_k\| = O(e^{-2k/\sqrt[4]{\kappa}}),$$

where κ is the condition number of $A^{-1/2}BA^{-1/2}$.

7.5 COMPUTING $(A\#_t B)^{-1}v$

The methods for computing the product of the weighted geometric mean times a vector, described in the previous sections, can be easily adapted for reducing the linear system

$$(A\#_t B)^{-1}v,$$

to the solution of a certain number of simpler linear systems.

Since $(A\#_t B)^{-1} = B^{-1}(B\#_t A)A^{-1}$, the quadrature formulae of section 7.3 can still be applied. From (7.18) we get the approximation

$$(A\#_t B)^{-1} \approx \frac{2 \sin(\pi t)}{\pi} \sum_{i=0}^N w_i ((1 - s_i)B + (1 + s_i)A)^{-1},$$

from (7.20) the approximation

$$(A\#_t B)^{-1} \approx \frac{2 \sin(\pi t)}{\pi} \sum_{i=0}^N w_i ((1 - s_i)^{\frac{1}{1-t}} A + (1 + s_i)^{\frac{1}{1-t}} B)^{-1},$$

and from (7.27) the approximation

$$(A\#_t B)^{-1} \approx \frac{-8K(k^2)\sqrt[4]{\lambda_m \lambda_M}}{\pi N k} \operatorname{Im} \left(\sum_{j=1}^N \frac{w(t_j)^{2t-1} \operatorname{cn}(t_j) \operatorname{dn}(t_j)}{(k^{-1} - \operatorname{sn}(t_j))^2} (w(t_j)^2 B - A)^{-1} \right),$$

when both A and B are real. The three quadrature formulae have exactly the same convergence properties as the respective formulae for $A\#_t B$.

Regarding the Krylov methods of section 7.4, we can exploit the identity

$$(A\#_t B)^{-1} = (A(A^{-1}B)^t)^{-1} = (A^{-1}B)^{-t}A^{-1},$$

reducing the computation of $(A\#_t B)^{-1}v$ to that of $(A^{-1}B)^{-t}(A^{-1}v)$, which can be performed by first computing $w = A^{-1}v$ and then approximating $(A^{-1}B)^{-t}w$ with any of the Krylov subspace methods described in section 7.4.

7.6 NUMERICAL TESTS

By means of numerical experiments, we illustrate the behavior of the methods presented in the paper for the computation of $(A\#_t B)v$ and $(A\#_t B)^{-1}v$, where A and B are medium- to large-scale matrices.

The tests were performed using MATLAB R2017a (9.2) on a machine equipped with an Intel i5-3570 Processor running at 3.40GHz and 8GB of dedicated RAM.

We compare the following methods:

1. The generalized Arnoldi algorithm [45, sect. 15.11] (Poly);
2. The extended Krylov subspace method [21] (Extended);
3. A rational Krylov subspace method, with poles chosen according to the adaptive strategy of Güttel and Knizhnermann [28] (RatAdapt);
4. A rational Krylov subspace method, where the choice of the poles is based on the solution of the best rational approximation of an auxiliary problem [8] (RatFit);
5. The quadrature formula (7.18) (Quad1);
6. The quadrature formula (7.20) (Quad2);
7. The quadrature formula (7.27) (Elliptic).

Krylov subspace methods. Our implementations of the Krylov subspace method are based on the modified Gram–Schmidt procedure with reorthogonalization [26]. When approximating $Af(A^{-1}B)v$, we can decide to use either the projection of $A^{-1}B$ onto the Krylov subspace or the matrix containing the orthonormalization coefficients used in the Gram–Schmidt process. When the Krylov subspace is enlarged, the projection does not have to be computed from scratch, but can be updated cheaply by exploiting a suitable recurrence. This choice still leads to a larger computational cost, due to one or more additional matrix-vector products and/or linear system solves per step, but guarantees that the projected matrix is symmetric positive definite. The matrix obtained by the orthogonalization procedure, on the other hand, is numerically not Hermitian, and it is not Hermitian when rational Arnoldi is used as described in section 7.4.3.

In our implementations of Poly and Extended, we trade off maintaining the structure of the problem for efficiency, and use the orthonormalization coefficients to build the reduced matrix. In this case the fractional power of a nonnormal matrix can be computed by spectral decomposition or by using algorithms for the real power of dense matrices [36], [34] (all these algorithms require $O(\ell^3)$ ops for a matrix of size ℓ). We stress that, in our tests, this choice did not reduce the accuracy of the final result, and only marginally affected the computational cost.

Rational Krylov methods, however, produce a pair of matrices from the orthonormalization coefficients, and it is not obvious how to combine them in order to obtain an approximation of $Af(A^{-1}B)v$. For that reason we resort to the slightly more expensive projections in RatAdapt and RatFit.

For the rational Krylov methods, the poles are chosen according to either the adaptive strategy by Güttel and Knizhnerman [28] or the function `rkfit` from the `rktoolbox` [7], based on an algorithm by Berljafa and Güttel [8], [9]. In our implementation, we get the poles by running `rkfit` on a surrogate problem of size 800 whose setup requires a rough estimate of the extrema of the spectrum of $A^{-1}B$.

Table 7.1: Comparison of the methods used in the numerical experiments in terms of knowledge of the spectrum of $A^{-1}B$ or $B^{-1}A$ (spectrum), type of linear systems to be solved (shifted systems, positive definite or not, or systems with the same left hand side), and possibility to increase the number of nodes/enlarge the Krylov subspace (update) exploiting the previous computation without starting from scratch.

Method	Spectrum	Systems	Update
Poly	no	same lhs	yes
Extended	no	same lhs	yes
RatAdapt	no	shifted pd	yes
RatFit	yes	shifted pd	yes
Quad1	yes	shifted pd	no
Quad2	yes	shifted pd	no
Elliptic ($t = 1/2$)	yes	shifted pd	no
Elliptic ($t \neq 1/2$)	yes	shifted	no

As a stopping criterion for the Krylov subspace methods, we use the estimate [40]

$$\frac{\|u - u_m\|}{\|u_m\|} \approx \frac{\delta_{m+j}}{1 - \delta_{m+j}},$$

where $\|\cdot\|$ is the spectral norm, $u = (A^{-1}B)^{-t}v$, u_m is the approximation at step m and δ_{m+j} is the norm of the relative difference between the approximation at the step m and $m+j$, i.e. $\|u_m - u_{m+j}\|/\|u_m\|$ where j is usually small and is set to 4 in our experiments.

Quadrature methods. For quadrature methods related to the Gauss–Jacobi quadrature, namely (7.18) and (7.20), the nodes and the weights are generated using the function `jacpts` of `Chebfun` [19], based on an algorithm by Hale and Townsend [31], which requires $O(N)$ operations to compute N nodes and weights of the quadrature. The scaling technique described at the end of section 7.3.3 is used to accelerate the convergence.

For Quad2 we use the quadrature formula (7.20) when $t > 1/2$, and if $t \leq 1/2$ we exploit the identity $A\#_t B = B\#_{1-t} A$ to reduce it to the former case.

In view of the remark at the end of section 7.3.3, the convergence in the matrix case is exactly predicted by the scalar convergence on the extreme eigenvalues. Thus, the number of nodes needed by Quad1 and Quad2 to get the required approximation is

estimated by applying its scalar counterpart, with a variable number of nodes and weights, to the extreme eigenvalues of the matrix $B^{-1}A$. These scalar problems are much easier and marginally affect the total computational cost of the algorithms, when dealing with large matrices.

Regarding the method described in section 7.3.4, we adapt the implementation given by Hale, Higham, and Trefethen [30], which uses the routines `ellipkjc` and `ellipkcp` from Driscoll's *Schwarz–Christoffel Toolbox* [17], [18]. In this case, the number of nodes is estimated by applying the same method to a 2×2 matrix whose eigenvalues are the extreme eigenvalues of $A^{-1}B$. Since in all our tests we consider only real matrices, the method of section 7.3.4, which is designed for real problems only, can always be applied.

Linear systems and extreme eigenvalues. In both Krylov subspace methods and quadrature methods, the problem is reduced to the solution of linear systems which are solved by the MATLAB sparse linear solver, exploiting the band and the positive definite structure. The linear systems to be solved by the method `Elliptic` are not guaranteed to be positive definite for $t \neq 1/2$ and this may considerably increase the overall time required by the algorithm.

Finally, the extreme eigenvalues of $A^{-1}B$ (or $B^{-1}A$), when needed, are approximated with two significant digits by calling the function `eigs` of MATLAB, with the pair (B, A) (or (A, B)) as argument. In Table 7.1 we give a synoptic comparison of the key features of the methods.

Test 1. In section 7.3, we considered two Gauss–Jacobi quadrature formulae for z^{-t} , one based on (7.7), implemented by `Quad1` and one based on (7.8), implemented by `Quad2`. We derived a bound on the rate of convergence of both formulae: $|\mathcal{C}(\sqrt{z})|^2$ with $\mathcal{C}(x) = \frac{1-x}{1+x}$ for `Quad1`, and $(1/\rho^{(2)}(z))^2$ with $\rho^{(2)}$ as in (7.15) for `Quad2`. The latter is valid just for $t = 1/2$.

We compare the experimental rate of convergence, which is the median of the error reduction over a certain number of steps, with the predicted rate of convergence. The results, for $t = 1/2$, are drawn in Figure 7.1. As one can see, the first quadrature

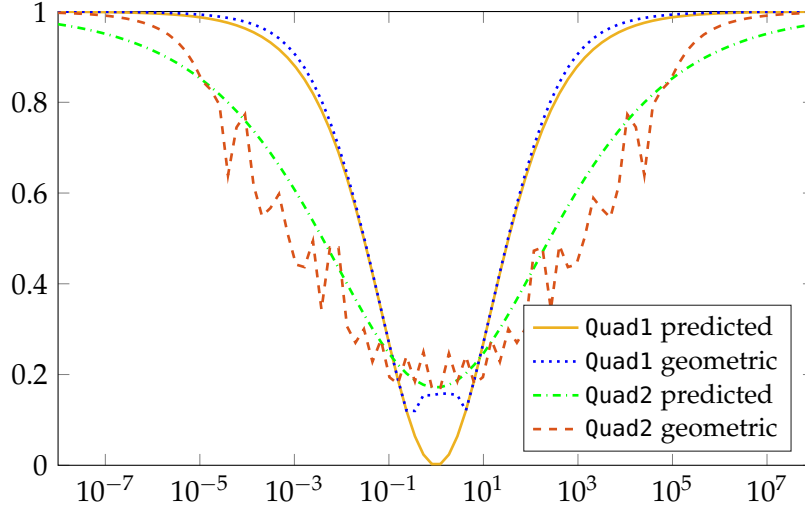


Figure 7.1: Comparison of the parameters of convergence (on the y -axis) of the two Gaussian quadrature formulae for $z^{-1/2}$ (on the semilogarithmic x -axis).

formula is more accurate for values of $|z|$ close, in magnitude, to 1, while the second gives better results for values of $|z|$ far from 1.

If we consider a positive definite matrix A scaled so that $\lambda_M \lambda_m = 1$ (where λ_M and λ_m are the extreme eigenvalues of A), then the first formula seems to be more convenient for well conditioned matrices, say with $\lambda_M/\lambda_m \lesssim 70$.

For $t \neq 1/2$ the bound for Quad1 is still valid, as confirmed by numerical experiments not reported here, while the bound for Quad2 is less predictive, and does not give any information for $t \neq 1/2$. Nevertheless, the asymptotic expansion (7.16) suggests a better convergence for Quad2 for $t = (p-1)/p$ and the quadrature formula shows an acceptable convergence rate even for values of t such that the integrand is not analytic, provided that $t \geq 1/2$. By using the formula $A \#_t B = B \#_{1-t} A$ we can achieve similar convergence properties also for $t < 1/2$.

Test 2. Since the convergence of most of the methods depends on the conditioning of the matrix $A^{1/2} B^{-1} A^{1/2}$ (that is λ_M/λ_m , where λ_M and λ_m are the largest and the smallest, respectively, eigenvalues of the matrix), we generate two matrices A and B such that $A^{-1} B$ (and thus $A^{1/2} B^{-1} A^{1/2}$) has prescribed eigenvalues.

We consider matrices of size 1000, so that a reference value for $w = (A \#_t B)v$ can be computed by means of a reliable algorithm for the dense case, namely the Cholesky–

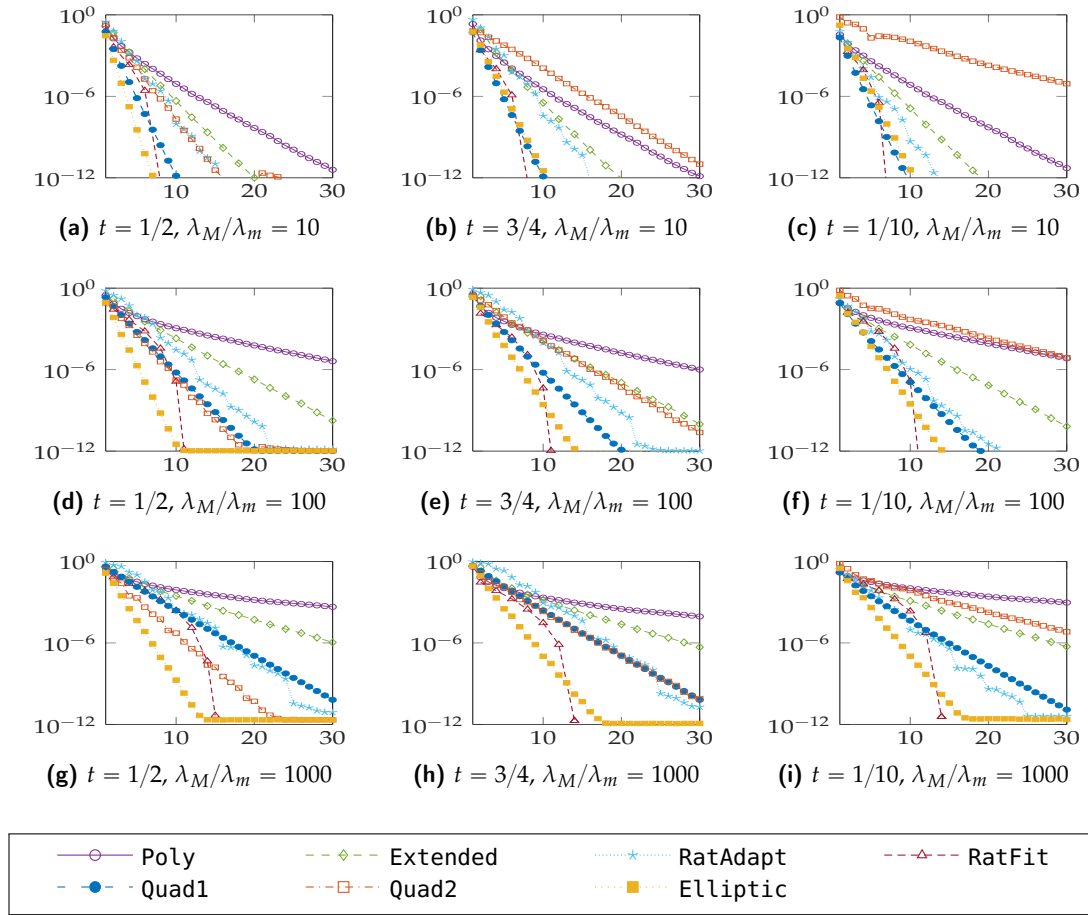


Figure 7.2: Convergence of the methods in Table 7.1 for computing $(A\#_t B)v$ for $t \in \{1/2, 3/4, 1/10\}$ and $\lambda_M/\lambda_m \in \{10, 100, 1000\}$, where λ_M and λ_m are the extreme eigenvalues of $A^{1/2}B^{-1}A^{1/2}$. We consider on the x -axis the number of nodes for quadrature methods and the dimension of the subspace for Krylov methods; and on the y -axis the relative error with respect to a reference solution.

Schur algorithm described in [35, sect. 3], which is implemented by the sharp function of the Matrix Means Toolbox [13].

For each method, the relative forward error of the computed value \tilde{w} with respect to the reference value, namely

$$\varepsilon = \frac{\|\tilde{w} - w\|}{\|w\|},$$

is measured in the spectral norm for a variable number of nodes of the quadrature methods and for a variable size of the Krylov subspace.

The results are drawn in Figure 7.2. The tests confirm the predicted dependence of the convergence on the conditioning of $A^{1/2}B^{-1}A^{1/2}$. The final accuracy of all methods is comparable, while we observe a different convergence behavior for $t = 1/2$ and for $t \neq 1/2$, for the methods Quad2 and Elliptic.

For $t = 1/2$, Elliptic generates the best rational relative minimax approximation of the function $z^{-1/2}$ on the interval $[\lambda_m, \lambda_M]$ [30]. This is the reason why it converges faster than the other methods, which produce different rational approximations to $z^{-1/2}$. We note that RatFit converges in a similar number of steps and that Quad2 converges much faster than Quad1 as λ_M/λ_m grows, as predicted in (7.23). Regarding the Krylov subspace methods, we observe linear convergence which is very slow for the Arnoldi method and it is quite fast when the adaptive strategy is used in the rational Krylov method.

For $t \neq 1/2$, Krylov methods and Quad1 have the same behavior they have for $t = 1/2$. The Elliptic method does not produce the best rational approximation anymore, and although A and B are real, it may require the solution of complex linear systems. However, despite in this case it need not be the fastest method, it still shows a remarkably fast convergence. The behavior of Quad2 degrades fast as t gets far from $t = 1/2$, a partial explanation for this is given in section 7.3. The fastest convergence for $t \neq 1/2$ is usually achieved by RatFit.

Test 3. In order to illustrate the behavior of the methods when dealing with large-scale matrices, we consider four pairs of conformable symmetric positive definite matrices from *the University of Florida Sparse Matrix Collection* [16].

The four choices considered in our experiments are described in Table 7.2. In the case of dataset 3, due to the extreme ill-conditioning of one of the two matrices (whose 1-norm condition number is approximatively $3 \cdot 10^{19}$) and the large rate $\lambda_M/\lambda_m \approx 10^{18}$ (where λ_M/λ_m is the conditioning of the matrix $A^{1/2}B^{-1}A^{1/2}$), we were not able to get any result. Since this dataset is interesting being the only one with non-banded matrices, we tamed the conditioning of the data, without affecting the nonzero structure, by adding the matrix $10^{-3}I$ to both matrices.

Table 7.2: ID in the University of Florida Sparse Matrix Collection, size and sparsity pattern of the matrices used in the experiments on large-scale matrices. In dataset 3, the asterisk means that a small multiple of the identity has been added to the two matrices.





Dataset	λ_M/λ_m	IDs in UFsmc	Size	Pattern
1	71.1	1312 & 1314	40 000	
2	7.5	1275 & 1276	90 449	
3	299.5	2257* & 2258*	102 158	
4	1.2	942 & 946	504 855	

Table 7.3: Comparison of the algorithms presented in the paper, when applied to large-scale matrices, in terms of CPU time (in seconds) and number of linear systems to be solved (between parentheses). We do not report any data for methods that require more than 1000 system solves to achieve the required accuracy.

	t	Poly	Extended	RatAdapt	RatFit	Quad1	Quad2	Elliptic
1	0.50	1.6 (50)	1.0 (32)	1.6 (21)	1.7 (11)	2.1 (20)	2.1 (20)	1.6 (11)
	0.75	1.3 (45)	1.0 (32)	1.6 (21)	1.7 (11)	2.1 (20)	3.0 (35)	3.8 (13)
	0.10	1.6 (54)	1.0 (32)	1.4 (18)	1.6 (10)	2.0 (19)	6.1 (82)	3.8 (13)
2	0.50	7.7 (13)	6.8 (16)	10.0 (11)	18.8 (07)	21.1 (11)	26.0 (17)	18.6 (07)
	0.75	7.2 (12)	6.8 (16)	10.0 (11)	18.8 (07)	20.3 (10)	40.6 (35)	70.5 (11)
	0.10	7.2 (12)	6.8 (16)	8.1 (09)	18.8 (07)	20.2 (10)	80.6 (83)	65.2 (10)
3	0.50	–	17.8 (106)	15.0 (40)	10.0 (18)	–	16.5 (44)	8.9 (19)
	0.75	–	21.0 (118)	17.6 (46)	9.5 (17)	–	14.0 (36)	20.3 (19)
	0.10	–	10.2 (74)	8.9 (25)	10.4 (19)	–	22.4 (63)	22.2 (21)
4	0.50	18.9 (07)	25.4 (12)	28.3 (07)	69.6 (03)	72.3 (04)	115.9 (16)	75.1 (04)
	0.75	19.0 (07)	23.1 (12)	28.3 (07)	69.5 (03)	72.2 (04)	185.4 (35)	192.9 (06)
	0.10	17.1 (06)	19.3 (10)	24.1 (06)	69.5 (03)	72.3 (04)	364.8 (83)	192.5 (06)

In order to test the methods in Table 7.1, we compare the CPU time required, for $t = 1/2$, $t = 3/4$ and $t = 1/10$, to fulfill the stopping criterion. We do not report the CPU time if the corresponding algorithm does not achieve the accuracy threshold after building a Krylov space of dimension 1000 or using 1000 quadrature nodes.

The results, given in Table 7.3, show that the convergence speed is dictated by the ratio λ_M/λ_m , as predicted, while the CPU time is not necessarily related to the number of linear system solves (between parentheses). Indeed, some methods require spectral information (see Table 7.1) and this task turns out to be costly, when the methods for computing the extreme eigenvalues converge very slowly (datasets 1, 2, and 4), while it does not influence dramatically the computational cost when it converges quickly (dataset 3). In particular, in dataset 3, if we denote by $\lambda_1 \geq \dots \geq \lambda_n$

the eigenvalues of $A^{-1}B$, where n is the size of A , then the parameters that determine the convergence of the power and inverse power methods, say $\gamma_1 = \lambda_2/\lambda_1$ and $\gamma_2 = \lambda_n/\lambda_{n-1}$, are bounded by 0.981, so that the extreme eigenvalues are computed very efficiently and the methods requiring the spectrum perform relatively well.

The Arnoldi and the extended Krylov subspace methods require no spectral information and the solution of linear systems with the same left hand side. In our code, we exploit this fact and begin by finding the Cholesky factorization of A and B and use it to solve efficiently all subsequent linear systems. To cope with sparse non-banded matrices and avoid excessive fill-in, we reorder the rows and columns of the matrix by applying an approximate symmetric minimum degree permutation, which we compute by means of the MATLAB `symamd` function. Notice that Poly gives good results for the dataset 4, where λ_M/λ_m is exceptionally small; when λ_M/λ_m grows, the fastest convergence of other Krylov methods makes them preferable. Extended and RatAdapt are good options if nothing is known about the problem, but when λ_M/λ_m is large (and an approximation of the spectrum can be reasonably computed) as in dataset 3, they may be overtaken by RatFit or by the quadrature methods.

On the other hand, the methods based on quadrature do not seem to be competitive for $t \neq 1/2$. While Quad1 converges too slowly, and this results in a large computational cost, the convergence of Quad2 is fast for $t = 1/2$, but its performance degrades rapidly as t approaches 0 or 1. Finally, the method based on the conformal transformation (Elliptic) requires a very small number of linear system to be solved, but these systems, for $t \neq 1/2$, are not positive definite and this results in a generally larger computational cost.

Finally, we wish to point out that in the dataset 4, the reason for the overhead of RatFit, among Krylov methods is related to the cost of the approximation of the spectrum of $A^{-1}B$. The big difference between the overall cost of Quad1 and Quad2, with respect to the number of linear system solves, depends on the fact that, in our implementation, Quad1 spends most of the time trying to compute the extreme eigenvalues of $A^{-1}B$. On the contrary, in the dataset 3, the conditioning is high, thus the convergence of the methods is slow, but the convergence of the power and inverse

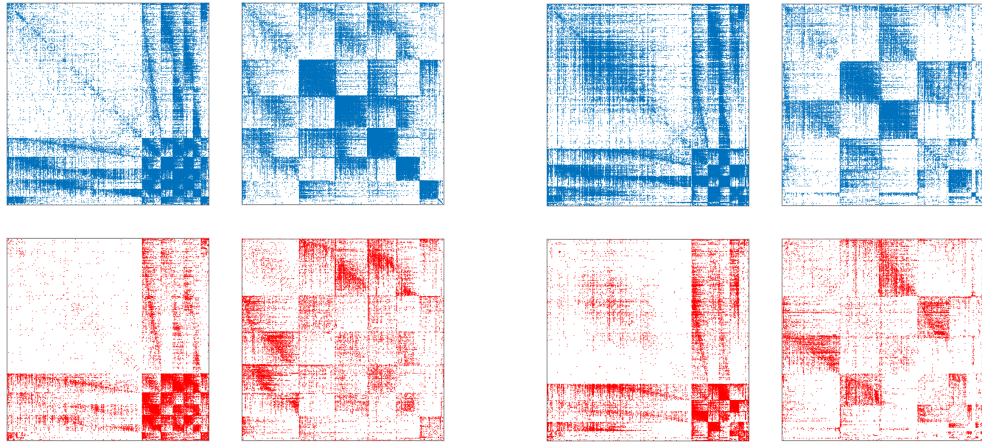
(a) Clustering for $t = 0.35$.(b) Clustering for $t = 1/2$.

Figure 7.3: The two figures report positive (blue, top left) and negative (red, bottom left) adjacency matrices of the Wikipedia RfA signed network. The rows and columns are reordered according to a clustering of the eigenvectors corresponding to the smallest 30 eigenvalues of $W^{+\#_{0.35}} W^-$ (Figure 7.3a) and $W^{+\#_{1/2}} W^-$ (Figure 7.3b). The right columns shows a detail of the last rows and columns of the corresponding matrix on the left.

power methods is fast, and the extreme eigenvalues are computed very efficiently and the fastest methods are among those requiring the spectrum (i.e., RatFit and Elliptic).

It is worth stressing that our results are just indicative and do not represent exactly what would happen if high performance implementations were used.

Test 4. The weighted geometric mean (with $t = 1/2$) is considered by Mercado, Tudisco, and Hein [44] as a tool for clustering *signed networks*, that is, networks that model both attractive and repulsive relationships by means of positive and negative (weighted) edges, respectively. It is customary to assign to these networks two distinct adjacency matrices, A^+ , for positive edges, and A^- , for negative ones.

The clustering process consists of several steps. After preprocessing the data by discarding all the rows and columns that do not belong to the largest connected component of the undirected graph of the network, the algorithm constructs W^+ , the normalized signed Laplacian of A^+ , and W^- , the normalized signless Laplacian [43] of A^- . The rows (and columns) of the matrix are divided into k communities by

performing a k -means clustering of the eigenvectors of $W^+ \# W^-$ corresponding to the k smallest (in magnitude) eigenvalues. In [44], the eigenpairs of $W^+ \# W^-$ are computed by means of the inverse power method [52, Lect. 27], where each linear system of the form $(W^+ \# W^-)^{-1}v$ is solved by constructing an Extended Krylov subspace.

We test the methods discussed here on the Wikipedia Request for Adminship signed network [54], which is available as part of the Stanford Large Network Dataset Collection (SNAP) [42]. The matrices in this dataset have size 8297, and W^+ and W^- have density (number of nonzero entries divided by the total number of elements) 4.10×10^{-3} and 9.98×10^{-4} , respectively. After the preprocessing stage, the size reduces to 6186 and the density to 1.20×10^{-3} and 4.30×10^{-3} , which makes them large and sparse enough to benefit from sparse matrix techniques.

First, we observe that with a similar computational effort, one can obtain a clustering using a weighted geometric mean with $t \neq 1/2$. Figure 7.3, compares the reordering obtained using $k = 30$ eigenvectors for $t = 0.35$ and $t = 1/2$. A quantitative comparison of the two results is not possible, since there is no widely accepted metric for measuring the quality of the clustering of a signed network. We point out, however, that the reordering for $t = 0.35$ shows a k -balanced behavior: after the reordering, the nonzeros of A^+ tend to appear in blocks along the diagonal, whereas those of A^- are localized in non-diagonal blocks.

Suitable clusterings are provided by using different values of t . An interesting open problem could be to identify the value of t providing the clustering more adherent to the model problem.

In Figure 7.4 we show how the parameter t influences the CPU time needed by the methods to solve the linear system $(W^+ \#_t W^-)^{-1}v$. Most methods perform better for values of t larger than $1/2$, the execution time of Quad1 does not seem to depend on t and that of Quad2 is symmetric with respect to $1/2$. For this network, the best methods, with a comparable CPU time, are Poly and Extended.

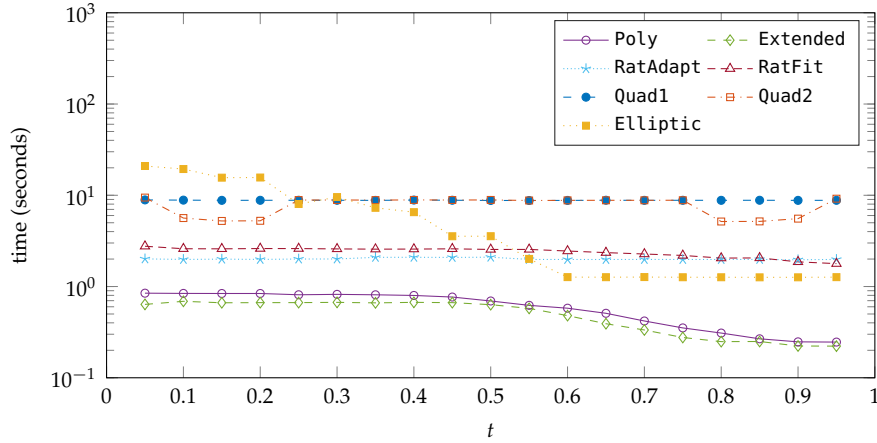


Figure 7.4: CPU time needed by the various methods for computing $(A\#_t B)^{-1}v$ with respect to t .

7.7 CONCLUSIONS

We consider several numerical algorithms for the approximation of $(A\#_t B)v$ and $(A\#_t B)^{-1}v$ for $t \in (0, 1)$. These methods exploit rational approximation of the function z^{-t} by either performing numerical quadrature or building a Krylov subspace. In both cases the problem is reduced to the solution of a certain number of linear systems, and thus assessing the performance of any of the algorithms discussed throughout the paper amounts to estimating the number and nature of linear systems to be solved.

The number of linear systems depends on the degree of the quadrature formula, for quadrature methods, and on the dimension of the constructed subspace, for Krylov methods. Note that this number can be efficiently estimated *a priori* in the former case, by applying the method to either a scalar or a 2×2 case, but cannot be predicted so easily in the latter.

On the other hand, the performance is influenced by the kind of linear system to be solved. For instance, when $t \neq 1/2$ the method `Elliptic` is quasi-optimal with respect to the convergence, being not far from the rational minimax approximation, but it requires the solution of complex linear systems with non-positive definite coefficient, which results in a sensible increase in terms of computational cost. Another example is represented by the extended Krylov subspace method (`Extended`), which despite

requiring more linear systems than the other two rational Krylov methods considered in the paper (RatAdapt and RatFit), is faster when the subspace need to be large. The reason behind this is that since Extended solves linear systems all having the same coefficient matrices, it is usually worth computing a factorization, at the price of a usually negligible overhead, in order to make the solution of the successive linear systems faster. The larger the space is, the more this approach pays off.

According to the experimental results in section 7.6, the choice of the method should be dictated by the spread of the eigenvalues of the matrix $A^{-1}B$ and the structure of A and B . In extremely well-conditioned cases, we expect all the methods to converge in very few iterations, and it is enough to build a polynomial Krylov space to approximate the solution. For mildly ill-conditioned matrices, Extended generates a Krylov subspace which is not too large, and the overhead introduced by the factorization is balanced by the reduction in execution time of the single iterations.

For severely ill-conditioned matrices a general recipe cannot be given, but, in this case, the quadrature methods become competitive. In particular, when $t = 1/2$ or close to 0 and 1, Elliptic seems to be the best choice, whereas for intermediate values of t Quad2 is very effective. The convergence of Quad1 is considerably slowed down and this method is totally impractical in this case. Krylov methods loose their supremacy because of the growth of the space, which implies a massive overhead due to the Gram–Schmidt orthogonalization of the basis. In principle, this problem could be alleviated by making use of suitable restarting techniques during the construction of the Krylov space. This optimization is currently under investigation and will be the subject of future work.

ACKNOWLEDGEMENTS

The authors are grateful to Valeria Simoncini, who advised the first author during his MSc thesis, and to Mario Berljafa, for fruitful discussions about the rational Arnoldi methods. The authors wish to thank Nicholas J. Higham for providing useful com-

ments which improved the presentation of the paper, and Daniel Loghin, Pedro Mercado and Francesco Tudisco, who provided some details regarding the large-scale matrices arising from applications.

BIBLIOGRAPHY

- [1] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Dover, New York, 10th ed., 1972.
- [2] T. ANDO, *Concavity of certain maps on positive definite matrices and applications to Hadamard products*, Linear Algebra Appl., 26 (1979), pp. 203–241.
- [3] T. ANDO, C.-K. LI, AND R. MATHIAS, *Geometric means*, Linear Algebra Appl., 385 (2004), pp. 305–334. Special Issue in honor of Peter Lancaster.
- [4] M. ARIOLI, D. KOUROUNIS, AND D. LOGHIN, *Discrete fractional Sobolev norms for domain decomposition preconditioning*, IMA J. Numer. Anal., 33 (2011), pp. 318–342.
- [5] M. ARIOLI AND D. LOGHIN, *Discrete interpolation norms with applications*, SIAM J. Numer. Anal., 47 (2009), pp. 2924–2951.
- [6] ———, *Spectral analysis of the anisotropic Steklov–Poincaré matrix*, Linear Algebra Appl., 488 (2016), pp. 168–183.
- [7] M. BERLJAFÄ AND S. GÜTTEL, *A Rational Krylov Toolbox for MATLAB*, MIMS EPrint 2014.56, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2014. Available for download at <http://guettel.com/rktoolbox/>.
- [8] ———, *Generalized rational Krylov decompositions with an application to rational approximation*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 894–916.
- [9] M. BERLJAFÄ AND S. GÜTTEL, *The RKFIT algorithm for nonlinear rational approximation*, SIAM J. Sci. Comput., 39 (2017), p. A2049–A2071.

- [10] R. BHATIA, *Matrix Analysis*, vol. 169 of Graduate Texts in Mathematics, Springer-Verlag, New York, 1997.
- [11] —, *Positive Definite Matrices*, Princeton Series in Applied Mathematics, Princeton University Press, Princeton, NJ, USA, 2007.
- [12] —, *The Riemannian Mean of Positive Matrices*, Springer-Verlag, Berlin, 2013, pp. 35–51.
- [13] D. BINI AND B. IANNAZZO, *The Matrix Means Toolbox*. <http://bezout.dm.unipi.it/software/mmttoolbox/>.
- [14] J. R. CARDOSO, *Computation of the matrix p -th root and its Fréchet derivative by integrals*, Electron. Trans. Numer. Anal., 39 (2012), pp. 414–436.
- [15] J. CASTELLINI, *Krylov iterative methods for the geometric mean of two matrices times a vector*, Numer. Algorithms, 74 (2017), pp. 561–571.
- [16] T. A. DAVIS AND Y. HU, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25.
- [17] T. A. DRISCOLL, *Algorithm 756: A MATLAB toolbox for Schwarz–Christoffel mapping*, ACM Trans. Math. Software, 22 (1996), pp. 168–186.
- [18] —, *Algorithm 843: Improvements to the Schwarz–Christoffel toolbox for MATLAB*, ACM Trans. Math. Software, 31 (2005), pp. 239–251.
- [19] T. A. DRISCOLL, N. HALE, AND L. N. TREFETHEN, *Chebfun Guide*, Pafnuty Publications, 2014.
- [20] T. A. DRISCOLL AND L. N. TREFETHEN, *Schwarz–Christoffel Mapping*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, UK, 2002.
- [21] V. DRUSKIN AND L. KNIZHNERMAN, *Extended Krylov subspaces: Approximation of the matrix square root and related functions*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 755–771.

- [22] C. ESTATICO AND F. DI BENEDETTO, *Shift-invariant approximations of structured shift-variant blurring matrices*, Numer. Algorithms, 62 (2013), pp. 615–635.
- [23] A. FROMMER, S. GÜTTEL, AND M. SCHWEITZER, *Efficient and stable Arnoldi restarts for matrix functions based on quadrature*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 661–683.
- [24] A. FROMMER AND V. SIMONCINI, *Matrix functions*, in Model order reduction: theory, research aspects and applications, vol. 13 of Math. Ind., Springer, Berlin, 2008, pp. 275–303.
- [25] W. GAUTSCHI, *A survey of Gauss-Christoffel quadrature formulae*, in E. B. Christoffel (Aachen/Monschau, 1979), Birkhäuser, Boston, MA, USA, 1981, pp. 72–147.
- [26] L. GIRAUD AND J. LANGOU, *When modified Gram–Schmidt generates a well-conditioned set of vectors*, IMA J. Numer. Anal., 22 (2002), pp. 521–528.
- [27] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4th ed., 2013.
- [28] S. GÜTTEL AND L. KNIZHNERMAN, *Automated parameter selection for rational Arnoldi approximation of Markov functions*, Proc. Appl. Math. Mech., 11 (2011), pp. 15–18.
- [29] ———, *A black-box rational Arnoldi variant for Cauchy–Stieltjes matrix functions*, BIT, 53 (2013), pp. 595–616.
- [30] N. HALE, N. J. HIGHAM, AND L. N. TREFETHEN, *Computing A^α , $\log(A)$, and related matrix functions by contour integrals*, SIAM J. Numer. Anal., 46 (2008), pp. 2505–2523.
- [31] N. HALE AND A. TOWNSEND, *Fast and accurate computation of Gauss–Legendre and Gauss–Jacobi quadrature nodes and weights*, SIAM J. Sci. Comput., 35 (2013), pp. A652–A674.
- [32] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002.

- [33] —, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [34] N. J. HIGHAM AND L. LIN, *An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360.
- [35] B. IANNAZZO, *The geometric mean of two matrices from a computational viewpoint*, Numer. Linear Algebra Appl., 23 (2015), pp. 208–229.
- [36] B. IANNAZZO AND C. MANASSE, *A Schur logarithmic algorithm for fractional powers of matrices*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 794–813.
- [37] B. IANNAZZO AND B. MEINI, *The palindromic cyclic reduction and related algorithms*, Calcolo, 52 (2015), pp. 25–43.
- [38] C. JAGELS AND L. REICHEL, *The extended Krylov subspace method and orthogonal Laurent polynomials*, Linear Algebra Appl., 431 (2009), pp. 441–458.
- [39] —, *Recursion Relations for the Extended Krylov Subspace Method*, Linear Algebra Appl., 431 (2009), pp. 441–458.
- [40] L. KNIZHNERMAN AND V. SIMONCINI, *A new investigation of the extended Krylov subspace method for matrix function evaluations*, Numer. Linear Algebra Appl., 17 (2010), pp. 615–638.
- [41] J. LAWSON AND Y. LIM, *Weighted means and Karcher equations of positive operators*, Proc. Natl. Acad. Sci. U.S.A., 110 (2013), pp. 15626–15632.
- [42] J. LESKOVEC AND A. KREVL, *SNAP Datasets: Stanford large network dataset collection*. <http://snap.stanford.edu/data>, June 2014.
- [43] S. LIU, *Multi-way dual Cheeger constants and spectral bounds of graphs*, Advances in Mathematics, 268 (2015), pp. 306–338.
- [44] P. MERCADO, F. TUDISCO, AND M. HEIN, *Clustering signed networks with the geometric mean of laplacians*, in Advances in Neural Information Processing Systems 29,

- D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds., Curran Associates, Inc., 2016, pp. 4421–4429.
- [45] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998. Unabridged, amended version of book first published by Prentice–Hall in 1980.
- [46] W. PUSZ AND S. L. WORONOWICZ, *Functional calculus for sesquilinear forms and the purification map*, Rep. Math. Phys., 8 (1975), pp. 159–170.
- [47] A. RALSTON AND P. RABINOWITZ, *A First Course in Numerical Analysis*, Dover, New York, 2nd ed., 1978.
- [48] A. RUHE, *Rational Krylov sequence methods for eigenvalue computation*, Linear Algebra Appl., 58 (1984), pp. 391–405.
- [49] —, *Rational Krylov algorithms for nonsymmetric eigenvalue problems. II. matrix pairs*, Linear Algebra Appl., 197–198 (1994), pp. 283–295.
- [50] —, *Rational Krylov: A practical algorithm for large sparse nonsymmetric matrix pencils*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 1535–1551.
- [51] V. SIMONCINI, *A new iterative method for solving large-scale Lyapunov matrix equations*, SIAM J. Sci. Comput., 29 (2007), pp. 1268–1288.
- [52] L. N. TREFETHEN AND D. BAU III, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
- [53] B. VON SYDOW, *Error estimates for Gaussian quadrature formulae*, Numer. Math., 29 (1977/78), pp. 59–64.
- [54] R. WEST, H. PASKOV, J. LESKOVEC, AND C. POTTS, *Exploiting social network structure for person-to-person sentiment analysis*, Trans. Assoc. Comput. Linguist., 2 (2014), pp. 297–310.

8

CONCLUSIONS

In arbitrary precision arithmetic, not all functions of matrices are equal. In some cases only little or no modification to fixed precision algorithms is needed to obtain accurate and efficient precision-oblivious methods. For many functions, however, state-of-the-art algorithms rely on a certain amount of precision-dependent computation that would be too expensive to perform in a multiprecision environment, where the precision at which the computation is performed becomes known only at runtime. Extending these algorithms to arbitrary precision environments poses several nontrivial challenges that need to be addressed.

In this thesis we considered methods based on Padé approximation, which belong to this second class. We showed how the scaling and squaring and inverse scaling and squaring algorithms can be adapted to multiprecision in order to compute the matrix exponential and the matrix logarithm, respectively. We achieved this by combining new bounds on the forward error of Padé approximation that are cheap to compute at runtime with new strategies for selecting the algorithmic parameters which, in principle, can deal with approximants of arbitrarily large degree.

Our techniques generalize to other matrix functions. For instance, the state-of-the-art algorithms for computing fractional matrix powers [5], [6] and inverse trigonometric and inverse hyperbolic matrix functions [2] rely on the inverse scaling and squaring approach based on a bound on the forward error of the Padé approximation. These algorithms can readily be adapted to multiprecision by using the techniques we developed for the matrix logarithm [3]. Techniques similar to those used for the computation of the matrix exponential in high precision can be used for evaluating the matrix sine and cosine [1], [4], [7] in arbitrary precision arithmetic.

Many open problems still remain, however. When developing our algorithms, we mostly focused on precision higher than double. In many cases, algorithms we ob-

tained perform well in lower precision as well, but this is not the case, for instance, for the scaling and squaring algorithm presented in Chapter 5. The algorithms discussed there perform poorly in IEEE single and especially half precision arithmetic, as the matrix exponential—and the exponential function itself—is prone to underflow and overflow due to the limited range of these floating point number systems. We believe that scaling the matrix beforehand might help developing robust algorithms for computing the exponential of a matrix in low precision, but finding a scaling strategy that guarantees an accurate result remains an open problem.

Additional evidence of the fact that computing matrix functions in arbitrary precision arithmetic is not yet fully understood is given by Conjecture 5.2. This conjecture would have important implications not only in the development of an efficient algorithm for evaluating the matrix exponential in high precision by using diagonal Padé approximants: proving the conjecture would allow us to develop new algorithms for computing the matrix logarithm in multiprecision by applying the algorithms for the solution of rational equations discussed in Chapter 4 to the matrix equation $e^X = A$. This is an exciting question that will, in all likelihood, be the subject of future work.

BIBLIOGRAPHY

- [1] A. H. AL-MOHY, N. J. HIGHAM, AND S. D. RELTON, *New algorithms for computing the matrix sine and cosine separately or simultaneously*, SIAM J. Sci. Comput., 37 (2015), pp. A456–A487.
- [2] M. APRAHAMIAN AND N. J. HIGHAM, *Matrix inverse trigonometric and inverse hyperbolic functions: Theory and algorithms*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 1453–1477.
- [3] M. FASI AND N. J. HIGHAM, *Multiprecision algorithms for computing the matrix logarithm*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 472–491.

- [4] G. I. HARGREAVES AND N. J. HIGHAM, *Efficient algorithms for the matrix cosine and sine*, Numer. Algorithms, 40 (2005), pp. 383–400.
- [5] N. J. HIGHAM AND L. LIN, *A Schur–Padé algorithm for fractional powers of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1056–1078.
- [6] —, *An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360.
- [7] N. J. HIGHAM AND M. I. SMITH, *Computing the matrix cosine*, Numer. Algorithms, 34 (2003), pp. 13–26.