Algorithms for Matrix Functions and their Frechet Derivatives and Condition Numbers

Relton, S. D.

2014

MIMS EPrint: **2014.55**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

ALGORITHMS FOR MATRIX FUNCTIONS AND THEIR FRÉCHET DERIVATIVES AND CONDITION NUMBERS

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2014

Samuel David Relton
School of Mathematics

Contents

List of Tables

List of Figures

The University of Manchester

Samuel David Relton
Doctor of Philosophy
Algorithms for Matrix Functions and their Fréchet Derivatives and Condition Numbers
July 23, 2014

We discuss several new theoretical results and algorithms relating to the computation of matrix functions, their Fréchet derivatives, and their condition numbers. First, we provide novel algorithms for the matrix cosine and sine based upon backward error analysis in exact arithmetic. This is in contrast to current alternatives based upon forward error bounds. Our new algorithms exploit triangularity in the input matrix and use Padé approximants to the exponential in conjunction with approximants to the sine.

Second, a new algorithm for computing the matrix logarithm using only real arithmetic is given, based upon the algorithm of Al-Mohy and Higham [*SIAM J. Sci. Comput.* 34(4):C153–C169, 2012]. We also extend these algorithms to compute both the Fréchet derivatives and condition number of the matrix logarithm, making efficient reuse of intermediate computation. A backward error analysis of the method used to compute the Fréchet derivative is performed and a single backward error result applicable to both the matrix logarithm and its Fréchet derivative is obtained.

Third, we investigate higher order Fréchet derivatives of matrix functions and the level-2 condition number. Whilst higher order Fréchet derivatives have been researched in the context of Banach spaces there appears to be no previous research specifically focused on matrix functions. We provide proofs for the existence and continuity of such derivatives under certain conditions on the function and input matrix in question; the constructive nature of the proof allows us to derive an algorithm for computing higher order Fréchet derivatives. We also define higher order Kronecker forms and give an algorithm for their computation.

The level-2 condition number (the condition number of the condition number) for matrix functions is also defined and an upper bound is obtained in terms of the second Kronecker form. This bound is applicable to arbitrary functions with the necessary differentiability properties. Furthermore we analyze the properties of the level-2 condition number more closely for general nonsingular matrices with the matrix inverse, for normal matrices with the matrix exponential, and for Hermitian matrices on functions with a strictly monotonic derivative. This latter class of functions includes the logarithm and square root, for example. Numerical experiments show that the relationships we derive may hold approximately in more general circumstances.

Finally, we define the condition number of computing the Fréchet derivative of a matrix function. Using our new results on higher order Fréchet derivatives we derive an algorithm to estimate the condition number for a wide class of functions including the exponential, logarithm, and real matrix powers. The algorithm produces estimates to within a factor $6n$ for an $n \times n$ input matrix and successfully exploits structure in the second Kronecker form. Our numerical experiments show this algorithm to be more reliable than a heuristic estimate used previously.

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

Acknowledgements

I am extremely grateful to Prof. Nicholas Higham whose valuable insight, unwavering support, and careful attention to detail helped me formulate, organize, and present my ideas with a level of clarity I could only hope to achieve alone. Without his efforts this thesis would not have been possible.

Thanks also to Prof. Françoise Tisseur and Assist. Prof. Awad Al-Mohy for interesting discussions and fruitful collaborations during the last three years.

I would also like to thank Mary, Leo, Edvin, and everyone else in the Manchester Numerical Linear Algebra group for their helpful suggestions and much needed diversions.

Finally, I am extremely grateful to Demi and my parents for their love, support, and advice during my time at Manchester. Without them I would never have made it this far.

Publications

This thesis is based on four publications as detailed below.

- The material in chapter 2 is based on the paper: A. H. Al-Mohy, N. J. Higham, and S. D. Relton. New Algorithms for Computing the Matrix Sine and Cosine Separately or Simultaneously. MIMS EPrint 2014.31, June 2014. Submitted to *SIAM J. Sci. Comput.*

- The material in chapter 3 is based on the paper: 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(4), C394–C410, 2013. The algorithms from this chapter have been implemented in the NAG library and SciPy.

- The material in chapter 4 is based on the paper: N. J. Higham and S. D. Relton. Higher Order Fréchet derivatives of Matrix Functions and the Level-2 Condition Number. *SIAM J. Matrix. Anal. Appl.*, 35(3), 1019–1037, 2014.

- The material in chapter 5 is based on the paper: N. J. Higham and S. D. Relton. Estimating the Condition Number of the Fréchet derivative of a Matrix Function. MIMS EPrint 2013.84, December 2013. Accepted for publication in *SIAM J. Sci. Comput.*

Chapter 1

Introduction

Matrices, in various guises, have been used to solve linear equations for thousands of years but it was only in 1858 that Arthur Cayley realised matrices are interesting objects of study in their own right. In "A Memoir on the Theory of Matrices" [22] Cayley laid the foundations for both matrix algebra and matrix functions. In particular this included an investigation into the square root of 2×2 and 3×3 matrices. Other major advancements in the 19th century include Laguerre defining the matrix exponential via its power series in 1867 and Sylvester defining matrix functions $f(A)$ for general f using interpolating polynomials—see equation (1.1.7)—in 1883 [101].

Matrices were of mainly theoretical interest until 1938 when the book "Elementary Matrices and Some Applications to Dynamics and Differential Equations" by Frazer, Duncan, and Collar [41] showed how matrices could be utilized in applied mathematics. This was "the first book to treat matrices as a branch of applied mathematics" [26] and (among other things) highlighted the importance of the matrix exponential in solving differential equations.

After the arrival of the digital computer researchers were primarily focused on other aspects of matrices until the 1970s when interest in matrix functions began to grow. One particularly noteworthy paper from this period is "Nineteen Dubious Ways to Compute the Exponential of a Matrix" by Moler and Van Loan [82], [83], which critiques a variety of ways to compute the matrix exponential. Since then the amount of literature focused on matrix functions and their use in applications has increased rapidly [57, p. 379]. A detailed history of the early contributions to the field can be found in [57, Sec. 1.10].

Today matrix functions can be found in many areas of applied mathematics with a plethora of applications spanning the natural sciences, engineering, optimization, and social sciences. For example, the matrix exponential is used to design real-time reduced order models [10], to analyze the importance of nodes in networks [37], [38], [39], to track the flow of contaminants within buildings [87], and to determine nuclear burnup [91]. Meanwhile the matrix logarithm is used in image identification [12], [68], model reduction [90], and computer graphics [93], [94]. Additionally matrix powers (A^t for $t \in \mathbb{R}$) have been used in areas including the numerical solution of fractional partial differential equations [16], [20], for example. Other functions such as the matrix square root, matrix cosine and sine, and the matrix sign function are also used.

Increasingly the Fréchet derivative (defined in section 1.2) is also required, with recent examples including computation of correlated choice probabilities [1], registration of MRI images [13], computing linearized backward errors for matrix functions [29], Markov models applied to cancer data [43], matrix geometric mean computation [70], model reduction [89], [90], and tensor-based morphometry [108]. Higher order Fréchet derivatives have been used to solve nonlinear equations on Banach spaces by generalizing the Halley method [9, Sec. 3].

The growing interest in matrix functions is reflected by the increasing amount of numerical software containing matrix function algorithms. The Matrix Function Toolbox [51], the NAG library [76], SciPy [96], and ExpoKit [100], are examples of some popular libraries containing matrix function routines. The recent catalogue of numerical software for matrix functions by Higham and Deadman [59] surveys the algorithms used in many commercial and freely available libraries, including those mentioned above.

This thesis explores both theoretical and computational aspects of matrix functions, their derivatives, and condition numbers. The remaining sections within this chapter give a succinct introduction to matrix functions and related results to be built upon in the subsequent chapters. The rest of the thesis is organized as follows.

Chapter 2 identifies novel algorithms for computing the matrix cosine and sine functions which improve upon the current alternatives in terms of both accuracy and stability. We show that our new algorithms are backward stable in exact arithmetic

(defined in section 1.4) which leads to increased reliability when compared with alternative algorithms using forward error bounds, as shown by our extensive numerical experiments.

When computing any matrix function a natural question arises: if the input matrix A is perturbed by some small amount ΔA how close are $f(A)$ and $f(A+\Delta A)$? Such a situation might arise if the elements of the matrix are subject to some uncertainty or obtained from another calculation in which there might be experimental or rounding errors, for example. This notion of sensitivity is encapsulated by the condition number of a matrix function, which we describe in section 1.3, and is the theme of the next three chapters.

Chapter 3 contains algorithms for computing the matrix logarithm, its derivatives, and condition number in an accurate and efficient manner. We perform backward error analysis on a new algorithm for computing the derivatives of the matrix logarithm and use this to estimate the 1-norm condition number of the logarithm at a low cost, by recycling large amounts of computation. Our new algorithms for the derivatives and condition number are shown to be significantly more accurate and cost-effective than alternatives in numerical experiments. We also show how optimizations made for real matrices can significantly improve performance when compared to a more general algorithm applicable to both real and complex matrices.

Chapter 4 provides a more theoretical discussion on higher order derivatives of matrix functions and the level-2 condition number (the condition number of the condition number). We present theorems regarding the existence of arbitrarily many derivatives of a given matrix function and an algorithm for their computation. We also generalize the Kronecker form (see section 1.2) to higher order derivatives. Our investigations into the level-2 condition number include a bound for general matrix functions, a bound for the exponential of normal matrices, bounds for a class of functions including the matrix logarithm and square root of Hermitian matrices, and numerical experiments which could lead to an interesting avenue of future research.

Chapter 5 uses the concepts established in chapter 4 to design an algorithm that estimates the condition number of computing the derivatives of a matrix function. There are a number of applications where these derivatives are required and, to our knowledge, nobody has previously analyzed how accurately one might expect to compute

them. Our numerical experiments show that our new algorithm is far more reliable than the heuristic approximations that have been used previously.

Finally we summarise our findings in chapter 6.

We now introduce the basic concepts forming the foundation of our research.

1.1 Matrix functions

There are many notions of matrix functions to be found in the literature. In the most general sense a matrix function is any function with a matrix input such as the norm, the eigenvalues, and the rank of a matrix. However, throughout this thesis we are primarily interested in functions $f : \mathbb{C}^{n\times n} \mapsto \mathbb{C}^{n\times n}$ where both the inputs and outputs are square matrices and the function f is a generalization of some underlying scalar function. For instance, the underlying scalar function of the square root is $f(x) = x^{1/2}$ and we say X is a square root of A when $X^2 = A$.

For polynomials and rational functions, since addition, multiplication, and division translate readily to matrices, we can evaluate $p(x) = \sum_k \alpha_k x^k$ at a matrix argument as $p(A) = \sum_k \alpha_k A^k$ and $p(x)/q(x)$ as $q(A)^{-1}p(A)$, respectively. If the function has a power series expansion $f(x) = \sum_k \alpha_k x^k$ then we can define $f(A) := \sum_k \alpha_k A^k$ if all eigenvalues of A lie within the radius of convergence. This allows us to define the matrix exponential and logarithm as

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots, \tag{1.1.1}$$

$$\log(I + A) = A - \frac{A^2}{2} + \frac{A^3}{3} - \frac{A^4}{4} + \cdots, \quad \rho(A) \leq 1, \tag{1.1.2}$$

where $\rho(A)$ denotes the spectral radius. Similar expansions can be used for the matrix cosine and sine, for example. The criteria for convergence of an arbitrary Taylor series expanded about a point $\alpha \in \mathbb{C}$ can be found in [57, Thm. 4.7].

There are a number of, essentially equivalent, ways in which we can define more general matrix functions. We will focus on the three most popular definitions which use the Jordan canonical form, Hermite interpolating polynomials, and the Cauchy integral formula.

The *Jordan canonical form* of a matrix $A \in \mathbb{C}^{n\times n}$ is

$$Z^{-1}AZ = J := \operatorname{diag}(J_1, \ldots, J_p),$$

where each Jordan block $J_k \in \mathbb{C}^{m_k \times m_k}$ is of the form

$$J_k = \begin{bmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{bmatrix}, \tag{1.1.3}$$

Z is nonsingular, and $\sum_k m_k = n$. The values λ_k are the eigenvalues of A.

We now define $f(A)$ using this decomposition as

$$f(A) := Zf(J)Z^{-1} = Z\,\mathrm{diag}(f(J_k))Z^{-1}, \tag{1.1.4}$$

where we define

$$f(J_k) := \begin{bmatrix} f(\lambda_k) & f'(\lambda_k) & \cdots & \dfrac{f^{(m_k-1)}(\lambda_k)}{(m_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & f'(\lambda_k) \\ & & & f(\lambda_k) \end{bmatrix}, \tag{1.1.5}$$

and $f^{(j)}(\lambda_k)$ denotes the jth derivative of f at λ_k.

Note that we have made the implicit assumption that f and all necessary derivatives are defined at the eigenvalues of A. More precisely f is said to be *defined on the spectrum of* A if all the required values

$$f^{(j)}(\lambda_k), \quad j = 0 : m_k - 1, \quad k = 1 : p, \tag{1.1.6}$$

exist. When this condition fails $f(A)$ is undefined.

Some further comments on this definition of a matrix function are in order. Firstly, it can be shown that the definition is independent of the Jordan canonical form used, which is not unique. Secondly, for multivalued functions such as the square root and logarithm, even though we are required to choose the same branch of the function within a single Jordan block, we may use different branches on separate Jordan blocks, even if they involve the same eigenvalue. Using different branches of a function on the same eigenvalue results in a *nonprimary matrix function*. For example, $\left[\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ is a nonprimary square root of the 2×2 identity matrix. We will restrict our attention to primary matrix functions from this point onward; additional detail on nonprimary matrix functions can be found in [57, Sec. 1.4].

Our second definition of a matrix function uses interpolating polynomials. Let f be defined on the spectrum of A and let the polynomial $p(x)$ satisfy

$$p^{(j)}(\lambda_k) = f^{(j)}(\lambda_k), \quad j = 0 : m_k - 1, \quad k = 1 : p, \tag{1.1.7}$$

then $f(A)$ is defined as $p(A)$. If two polynomials $p(x)$ and $q(x)$ both satisfy the interpolation conditions (1.1.7) then $p(A) = q(A)$ [57, Thm. 1.3]. Furthermore there is a unique polynomial of minimal degree satisfying these conditions called the *Hermite interpolating polynomial*. Note that the coefficients of the polynomial $p(A)$ depend upon A themselves so that, in general, a new polynomial must be constructed for each input matrix.

Finally we can define matrix functions using the Cauchy integral formula. When f is analytic on and inside some closed contour Γ enclosing the eigenvalues of A we say

$$f(A) := \frac{1}{2\pi i} \int_\Gamma f(z)(zI - A)^{-1} dz. \tag{1.1.8}$$

This definition is particular popular in functional calculus, where the matrix A can be replaced by a more general operator acting on a Banach space, for example.

All three definitions, modulo the requirement for an analytic function f in (1.1.8), are equivalent [57, Thm. 1.12].

We also mention some basic properties of matrix functions which will be used throughout this thesis, the proofs of which can be found in [57, Thm. 1.13]. Each proof follows easily from one of three definitions above.

Theorem 1.1.1. *Let $A \in \mathbb{C}^{n\times n}$ and let f be defined on the spectrum of A. Then*

1. *$f(A)$ commutes with A;*
2. *$f(XAX^{-1}) = Xf(A)X^{-1}$ for any invertible $X \in \mathbb{C}^{n\times n}$;*
3. *the eigenvalues of $f(A)$ are $f(\lambda_i)$ where λ_i are the eigenvalues of A;*
4. *if $A = (A_{ij})$ is block triangular then $F = f(A)$ is also block triangular with the same block structure and moreover $F_{ii} = f(A_{ii})$.*

1.2 Fréchet derivatives

Much of this thesis will be concerned with the sensitivity of matrix functions to perturbations in the input matrix A. To analyze this we make frequent use of the Fréchet derivative. The *Fréchet derivative* of a matrix function at A is a linear function $L_f(A,\cdot) : \mathbb{C}^{n\times n} \mapsto \mathbb{C}^{n\times n}$ which, for any direction E, satisfies

$$f(A+E) - f(A) - L_f(A,E) = o(\|E\|), \tag{1.2.1}$$

where $o(\|E\|)$ means the left-hand side tends to 0 as $\|E\| \to 0$. If the Fréchet derivative exists at A then it is unique. Some authors refer to the Fréchet derivative as $Df(A)(\cdot)$, however our notation is used throughout much of the matrix function literature.

Closely related to the Fréchet derivative is the Gâteaux derivative (also known as the directional derivative) given by

$$G_f(A,E) := \lim_{t\to 0} \frac{f(A+tE) - f(A)}{t}. \tag{1.2.2}$$

The Fréchet derivative is stronger than the Gâteaux derivative: if the Fréchet derivative exists at A then it implies the existence of the Gâteaux derivative and is equal to it. On the other hand, if the Fréchet derivative does not exist, then the existence of the Gâteaux derivative depends upon the direction E. The following result gives the criterion under which the existence of the Gâteaux derivative implies the existence of the Fréchet derivative.

Theorem 1.2.1. *Given a matrix function f which is Gâteaux differentiable at A, if the Gâteaux derivative $G_f(A,E)$ is both linear in all directions E and continuous in A, then the Fréchet derivative exists and $L_f(A,E) = G_f(A,E)$.*

Proof. See, for example, [17, Sec. X.4] and [86, Sec. 8, Rem. 3]. □

The Fréchet derivative also satisfies the sum, product, and chain rules; the proofs of which can be found in [57, Chap. 3], for example.

Another useful property of Fréchet derivatives that we will use repeatedly is the existence of the *Kronecker form*. Since the Fréchet derivative is a linear operator it can be viewed as a matrix $K_f(A) \in \mathbb{C}^{n^2\times n^2}$ satisfying

$$K_f(A)\,\mathrm{vec}(E) = \mathrm{vec}(L_f(A,E)), \tag{1.2.3}$$

where vec is the operator stacking the columns of a matrix vertically from left to right. We refer to $K_f(A)$ as the Kronecker form of f at A.

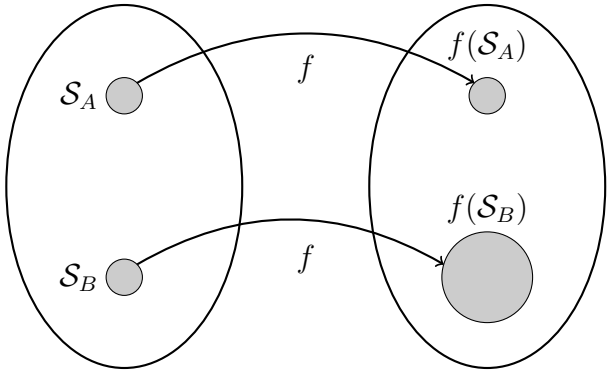

Figure 1.3.1: Shows the sensitivity of f at two points A and B, the centres of the two circles $\mathcal{S}_A$ and $\mathcal{S}_B$, respectively.

1.3 Condition numbers

The *condition number* of a matrix function measures the sensitivity of f at a given matrix A, depending on a norm $\|\cdot\|$. A familiar example is the relative condition number of the matrix inverse $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ which uses $f(X) = X^{-1}$ and the 2-norm. The condition number is a real scalar which measures the maximal sensitivity of the function over all possible perturbations. Following the style of Rice [92], the absolute condition number of a matrix function is defined as

$$\mathrm{cond}_{\mathrm{abs}}(f, A) := \lim_{\epsilon \to 0} \sup_{\|E\| \leq \epsilon} \frac{\|f(A+E) - f(A)\|}{\epsilon}, \tag{1.3.1}$$

and the relative condition number is defined as

$$\mathrm{cond}_{\mathrm{rel}}(f, A) := \lim_{\epsilon \to 0} \sup_{\|E\| \leq \epsilon\|A\|} \frac{\|f(A+E) - f(A)\|}{\epsilon\|f(A)\|}. \tag{1.3.2}$$

Problems with a small condition number are called well conditioned, whereas those with a large condition number are called ill conditioned. Whether a particular condition number is deemed small or large is highly problem dependent.

To illustrate this, Figure 1.3.1 shows the sensitivity of a matrix function f at two matrices A and B. For some small $\delta > 0$ let $\mathcal{S}_A = \{A + \Delta : \|\Delta\| \leq \delta\}$ and $\mathcal{S}_B = \{B + \Delta : \|\Delta\| \leq \delta\}$ denote a set of small perturbations to A and B with $f(\mathcal{S}_A)$ and $f(\mathcal{S}_B)$ representing the image of the two sets under f. Since $f(\mathcal{S}_A)$ is small we conclude that small perturbations to A result in small perturbations to $f(A)$ and hence f is well conditioned at A. However, small perturbations to B can result in large changes to $f(B)$ and so f is ill conditioned at B.

Using the Fréchet derivative (1.2.1) it is easy to see that the condition numbers can be expressed as

$$\operatorname{cond}_{\text{abs}}(f, A) = \max_{\|E\|=1} \|L_f(A, E)\|, \tag{1.3.3}$$

$$\operatorname{cond}_{\text{rel}}(f, A) = \max_{\|E\|=1} \|L_f(A, E)\| \frac{\|A\|}{\|f(A)\|}. \tag{1.3.4}$$

Furthermore we note that the relative condition number is simply a scaled version of the absolute condition number, a fact which often helps in the analysis of these quantities.

By choosing specific norms in (1.3.3) and (1.3.4) we can relate the condition number to the Kronecker form. For instance, the Frobenius norm condition number is equal to the 2-norm of the Kronecker form since

$$\begin{aligned} \operatorname{cond}_{\text{abs}}(f, A) &= \max_{\|E\|_F=1} \|L_f(A, E)\|_F \\ &= \max_{\|\operatorname{vec}(E)\|_2=1} \|K_f(A) \operatorname{vec}(E)\|_2 \\ &= \|K_f(A)\|_2, \end{aligned} \tag{1.3.5}$$

where we have used the fact that $\|X\|_F = \|\operatorname{vec}(X)\|_2$ for any matrix X.

Forming $K_f(A)$ explicitly and taking its norm costs $O(n^5)$ flops [57, Alg. 3.17], which is prohibitively expensive. Instead we often apply norm estimation techniques, such as the power method.

Usually we will be interested in the 1-norm, rather than the Frobenius norm, to take advantage of the computational efficiency offered by the block 1-norm estimator of Higham and Tisseur [66], which is available in MATLAB as function `normest1`. This algorithm estimates the 1-norm of an $n \times n$ matrix B by evaluating products of B and B^* with $n \times t$ matrices, where t is a parameter whose significance is explained below. In the 1-norm we have the following lemma.

Lemma 1.3.1. *For $A \in \mathbb{C}^{n\times n}$ and any matrix function f, using the 1-norm,*

$$\frac{\operatorname{cond}_{\text{abs}}(f, A)}{n} \le \|K_f(A)\|_1 \le n \operatorname{cond}_{\text{abs}}(f, A).$$

Proof. See [57, Lem. 3.18]. □

Using the block 1-norm estimation algorithm with $B = K_f(A)$, we now employ the same idea as [3] and approximate $\operatorname{cond}_{\text{abs}}(f, A) \approx \|K_f(A)\|_1$.

Algorithm 1.3.2. *This algorithm estimates the* 1*-norm of* $K_f(A)$ *using Fréchet derivative evaluations of the matrix function* f *at* A *[57, Alg. 3.22].*

1 Apply [66, Alg. 2.4] with parameter $t = 2$ to the Kronecker matrix $B = K_f(A)$, noting that $By = \mathrm{vec}(L_f(A, E))$ and $B^*y = \mathrm{vec}(L_f^\star(A, E))$, where $\mathrm{vec}(E) = y$.

Here, $\star$ denotes the adjoint. For the class of functions f satisfying $f(z) = \overline{f(\overline{z})}$ we have $L_f^\star(A, E) = L_f(A^*, E) = L_f(A, E^*)^*$ and so it is straightforward to compute $L_f^\star(A, E)$. This large class of functions includes the exponential, logarithm, real powers x^t ($t \in \mathbb{R}$), the sign function, and trigonometric functions, for example.

Known properties of Algorithm 1.3.2 are that it requires $4t$ Fréchet derivative evaluations on average and is rarely more than a factor of 3 away from $\|K_f(A)\|_1$ [57, p. 67]. Higher values of t give greater accuracy at the cost of more derivative evaluations. This means that we can estimate the condition number in the 1-norm to within a factor of $3n$ in $O(n^3)$ flops.

1.4 Forwards and backward errors

Throughout this thesis we will use the *forward error* and *backward error* to both design and test algorithms.

Given an algorithm that approximates $f(X)$ by Y, the quantities $\|f(X) - Y\|$ and $\|f(X) - Y\|/\|f(X)\|$ are called the *absolute* and *relative forward error*, respectively. If there exists a matrix ΔX such that $Y = f(X + \Delta X)$ then $\|\Delta X\|$ and $\|\Delta X\|/\|X\|$ are called the *absolute* and *relative backward error*, respectively. Furthermore when the forward or backward error of a numerical algorithm is guaranteed to be sufficiently small (typically less than a small multiple of the unit roundoff[1]) we say the algorithm is *forward* or *backward stable*, respectively.

The relative forward error, backward error, and condition number are related by the approximate relationship

$$\text{forward error} \lesssim \text{condition number} \times \text{backward error}. \tag{1.4.1}$$

[1] In IEEE floating point arithmetic the unit roundoff is $u = 2^{-24}$ for single precision and $u = 2^{-53}$ for double precision.

Typically we will aim to minimize the backward error in our algorithm design in order to attain good forward errors, assuming the condition number of the problem is not too large. For a full treatment of forward errors, backward errors, and condition numbers we refer the reader to the excellent book by Higham [55].

Chapter 2

New Algorithms for Computing the Matrix Cosine and Sine Separately or Simultaneously

2.1 Introduction

In recent years research into the computation of matrix functions has primarily focused on the matrix exponential, the logarithm, and matrix powers. Also of interest are the matrix sine and cosine, which can be defined for $A \in \mathbb{C}^{n\times n}$ by their Maclaurin series

$$\sin A = A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!} + \cdots, \tag{2.1.1}$$

$$\cos A = I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \cdots. \tag{2.1.2}$$

Their importance stems from their role in second order differential equations. For example, the second order system

$$y''(t) + Ay(t) = g(t), \qquad y(0) = y_0, \quad y'(0) = y_0', \tag{2.1.3}$$

which arises in finite element semidiscretization of the wave equation, has solution

$$y(t) = \cos(\sqrt{A}t)y_0 + (\sqrt{A})^{-1}\sin(\sqrt{A}t)y_0' + \int_0^t (\sqrt{A})^{-1}\sin\bigl(\sqrt{A}(t-s)\bigr)g(s)\,ds, \tag{2.1.4}$$

where $\sqrt{A}$ denotes any square root of A [42, p. 124], [97]; see also [57, Prob. 4.1] for the case $g(t) = 0$. More generally, (2.1.3) arises with a right-hand side of the form $g(t, y(t), y'(t))$ in a wide variety of applications and these problems can have

highly oscillatory solutions [106] or be stiff with A having large norm [81]. Further generalizations of (2.1.3) whose solutions involve the matrix sine and cosine have a right-hand side $f(t, y(t)) + Bu(t)$, where $u(t)$ is a control vector and B a matrix [99], possibly with a delayed linear term $Ay(t-\tau)$ for a constant delay $\tau > 0$ [32].

Serbin and Blalock [98] proposed the following algorithm for the matrix cosine, which has served as a basis for several subsequent algorithms. It employs the double angle formula $\cos(2X) = 2\cos^2 X - I$ [57, Thm. 12.1].

Algorithm 2.1.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes an approximation Y to $\cos A$.*

1 Choose an integer $s \geq 0$ such that $X = 2^{-s}A$ has small norm.
2 $C_0 = r(X)$, where $r(X)$ approximates $\cos X$.
3 for $i = 1 : s$
4 $\quad C_i = 2C_{i-1}^2 - I$
5 end
6 $Y = C_s$

The algorithm has two parameters: the amount of scaling s and the function r, which is usually either a truncated Taylor series or a Padé approximant.

Serbin and Blalock do not propose any specific algorithmic parameters. Higham and Smith [65] develop an algorithm based on the [8/8] Padé approximant with the scaling parameter s chosen with the aid of a forward error bound. Hargreaves and Higham [46] derive an algorithm with a variable choice of Padé degree (up to degree 20), again based on forward error bounds. They also give an algorithm that computes $\cos A$ and $\sin A$ simultaneously at a lower computational cost than computing the two functions separately. However they do not give an algorithm to compute $\sin A$ alone because the double angle formula $\sin(2X) = 2\sin X \cos X$ for the sine involves the cosine. Indeed, as far as we are aware, no algorithm of the general form in Algorithm 2.1.1 has been proposed for computing $\sin A$ directly.

Recently Sastre et al. [95] have derived a new algorithm for the cosine that combines Taylor series approximations of degree up to 40 with sharper forward error bounds derived using ideas similar to those in [4, Sec. 4].

In this chapter we develop three new algorithms for the sine and cosine that are based on backward error analysis and are backward stable in exact arithmetic. Our backward error analysis has two advantages over the forward error analyses used in the derivation of existing algorithms. First, the backward error analysis applies to the overall algorithm, whereas the forward error analyses bound the forward error of the function of the scaled matrix only, and the best overall forward error bound contains a term exponential in the number of double-angle steps [57, Thm. 12.5]. Second, the current forward error-based algorithms actually bound the absolute error of the function of the scaled matrix rather than the relative error, which is a heuristic choice based on numerical experiments. With backward error analysis there is no need for such considerations, as relative backward errors are unequivocally appropriate.

A second key feature of our algorithms is that they exploit triangularity. They optionally reduce the original matrix to Schur form, but particular benefits are obtained when the original matrix is (real quasi-)triangular.

The algorithm for the matrix cosine follows the outline of Algorithm 2.1.1 but uses Padé approximants of the exponential rather than the cosine. The algorithm for the matrix sine scales by powers of 3 rather than 2, employs Padé approximants to the sine and the exponential, and uses the triple angle formula to undo the effects of the scaling. The algorithm for computing the cosine and the sine simultaneously makes use of a new choice of double angle formula that improves stability.

The outline of this chapter is as follows. In section 3.3 we perform backward error analysis of the Padé approximants for the sine and cosine and show how certain limitations can be overcome using alternative rational approximations based upon the matrix exponential. In section 2.3 we give explicit formulas for the cosine and sine of 2×2 (real quasi-)triangular matrices, which are used in the algorithms for better accuracy. In sections 2.4–2.6 we design cost effective methods for computing the matrix cosine and sine (separately and simultaneously) using the double and triple angle formulas. We then compare our algorithms numerically against existing alternatives in section 2.7.

2.2 Backward error analysis for the sine and cosine

We begin by analyzing the backward error of Padé approximants to the matrix sine and cosine. We find that the backward error analysis restricts the range of applicability of the Padé approximants—so much so for the cosine as to make the approximants unusable. We therefore propose and analyze an alternative method of approximation based upon Padé approximants to the exponential.

2.2.1 Padé approximants of matrix sine

Let $r_m(x) = p_m(x)/q_m(x)$ denote the $[m/m]$ Padé approximant to the sine function. Since the sine function is odd the Padé table splits into 2×2 blocks containing identical entries having odd numerator p_m and even denominator q_m [15, p. 65]. From this we can show that for $k \geq 2$ the number of matrix multiplications required to form r_{2k} is equal to that for forming r_{2k+1} and hence we need only consider diagonal Padé approximants of odd order after r_1 and r_2. For a similar discussion on the Padé table of the cosine see [77] and [104, p. 245].

We begin our analysis by defining $h_{2m+1}\colon \mathbb{C} \mapsto \mathbb{C}$ as

$$h_{2m+1}(x) = \arcsin r_m(x) - x, \tag{2.2.1}$$

where $\arcsin x$ denotes the principal arc sine, that is, the one for which $\arcsin x \in [-\pi/2, \pi/2]$ for x on the interval $[-1, 1]$. For $x \in \mathbb{C}$ with $|x| \leq 1$ we have $\sin\arcsin x = x$. It follows that for $X \in \mathbb{C}^{n\times n}$ with $\rho(r_m(X)) \leq 1$, where ρ denotes the spectral radius, $\arcsin r_m(X)$ is defined and, from (2.2.1),

$$r_m(X) = \sin(X + h_{2m+1}(X)) =: \sin(X + \Delta X). \tag{2.2.2}$$

This means that $\Delta X = h_{2m+1}(X)$ represents the backward error of approximating $\sin X$ by the $[m/m]$ Padé approximant, assuming that $\rho(r_m(X)) \leq 1$.

Figure 2.2.1 shows the order stars for the Padé approximants r_m, that is, the regions of the complex plane where $|r_m(x)| \leq 1$, for a range of m [25], [69]. For a given value of m, if all the eigenvalues of X lie within this region then our backward error analysis holds. Since the regions are not circular it is difficult to check this criterion in practice, but we found numerically that for $m = 1\colon 13$ the largest circular disk centered at the

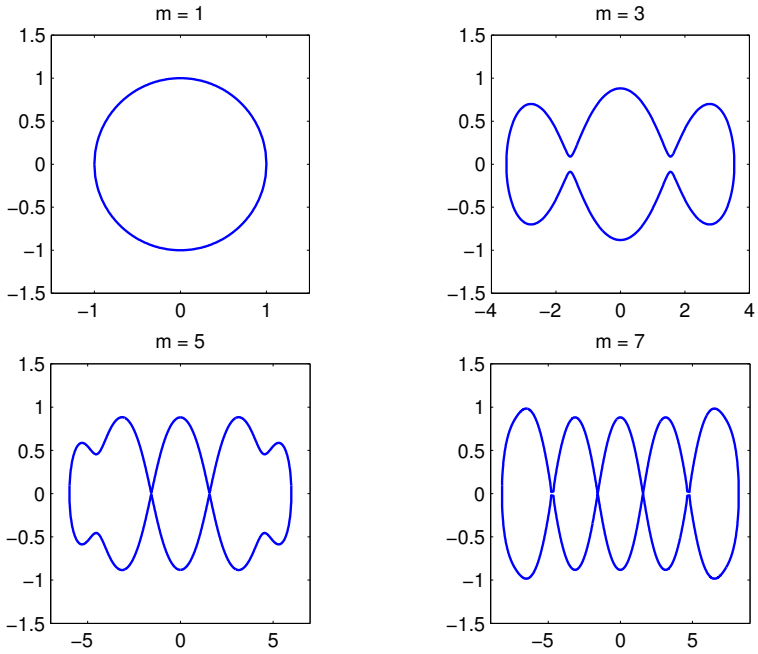

Figure 2.2.1: The boundary of the region within which $|r_m(x)| \leq 1$, where $r_m(x)$ is the $[m/m]$ Padé approximant to the sine function, for $m = 1, 3, 5, 7$.

origin that lies within all the regions has radius approximately $\operatorname{arcsinh} 1 \approx 0.881$. Therefore it is sufficient to check that $\rho(X) \leq \operatorname{arcsinh} 1$ for our analysis to hold.

Since $r_m(x) = \sin x + O(x^{2m+1})$ we have $h_{2m+1}(x) = \arcsin r_m(x) - x = O(x^{2m+1})$, and because arcsin and sin are odd functions we can write

$$h_{2m+1}(X) = X \sum_{k=0}^{\infty} c_{m,k} X^{2(m+k)}, \tag{2.2.3}$$

for some coefficients $c_{m,k}$. Taking the norm of this equation, using [4, Thm. 4.2(b)], and recalling that $\Delta X = h_{2m+1}(X)$, we can bound the normwise relative backward error by

$$\frac{\|\Delta X\|}{\|X\|} \leq \sum_{k=0}^{\infty} |c_{m,k}| \alpha_p(X)^{2(m+k)}, \tag{2.2.4}$$

where

$$\alpha_p(X) = \max\bigl(\|X^{2p}\|^{1/(2p)}, \|X^{2p+2}\|^{1/(2p+2)}\bigr) \tag{2.2.5}$$

and the integer $p \geq 1$ is such that $m \geq p(p-1)$. Note that we have $\rho(X) \leq \alpha_p(X) \leq \|X\|$ and $\alpha_p(X)$ can be much smaller than $\|X\|$ for nonnormal X, so the gains from working with (2.2.4) instead of the corresponding bound expressed solely in terms of $\|X\|$ can be significant. It is easy to show that $\alpha_3(X) \leq \alpha_2(X) \leq \alpha_1(X)$ and $\alpha_4(X) \leq \alpha_2(X)$, but the relationship between $\alpha_3(X)$, $\alpha_4(X)$, and $\alpha_5(X)$ depends upon

X, so we need to compute or estimate all of the latter three quantities and use the smallest, subject to the constraint $m \geq p(p-1)$. For example, to show $\alpha_3(X) \leq \alpha_2(X)$ we know from [4, Lem. 4.1] that $\|X^8\|^{1/8} \leq \|X^4\|^{1/4}$ and therefore

$$\alpha_3(X) = \max(\|X^6\|^{1/6}, \|X^8\|^{1/8}) \leq \max(\|X^6\|^{1/6}, \|X^4\|^{1/4}) = \alpha_2(X).$$

To ensure maximum accuracy we would like to bound the backward error (2.2.4) by the unit roundoff $u = 2^{-53} \approx 1.1 \times 10^{-16}$ in IEEE double precision arithmetic. If we define

$$\beta_m = \max\left\{ \beta : \sum_{k=0}^{\infty} |c_{m,k}| \beta^{2(m+k)} \leq u \right\}, \tag{2.2.6}$$

then $\alpha_p(X) \leq \beta_m \leq \operatorname{arcsinh} 1$ implies that $\|\Delta X\|/\|X\| \leq u$ for $m \leq 13$. In the second row of Table 2.5.1 we show the values β_m calculated using variable precision arithmetic in the Symbolic Math Toolbox for several values of m. We have $\beta_m > \operatorname{arcsinh} 1 \approx 0.881$ for $m \geq 9$, but our backward error bounds require that $\rho(X) \leq \operatorname{arcsinh} 1$. This means that for $\beta_7 < \alpha_p(X) \leq \operatorname{arcsinh} 1$ we can use $m = 9$, but for $\operatorname{arcsinh} 1 < \alpha_p(X) \leq \beta_9$ our backward error results are not applicable, so we artificially reduce β_9 to 0.881. From this point on we will not consider $m > 9$.

In the algorithm of section 2.5 we will use both Padé approximants and another class of rational approximations introduced in section 2.2.3.

2.2.2 Padé approximants of matrix cosine

For the matrix cosine we can attempt a similar analysis. Let $r_m(x)$ be the $[m/m]$ Padé approximant to the cosine and define $h_{2m}(x) = \arccos r_m(x) - x$, where arccos denotes the principal arc cosine, which maps $[-1, 1]$ to $[0, \pi]$. Analogously to the argument in the previous section, in order to obtain a backward error relation we need $|x| \leq 1$.

Figure 2.2.2 shows the order stars of r_m for a range of m. Requiring the eigenvalues of X to lie inside the order stars imposes tight restrictions on them. Even more prohibitively, the eigenvalues must also lie in the strip of the right half-plane with real part between 0 and π, so that the principal branch of arccos gives $\arccos \cos x = x$.

As we are not aware of any satisfactory way to overcome these restrictions we will use an alternative rational approximation whose backward error analysis is applicable for every set of eigenvalues.

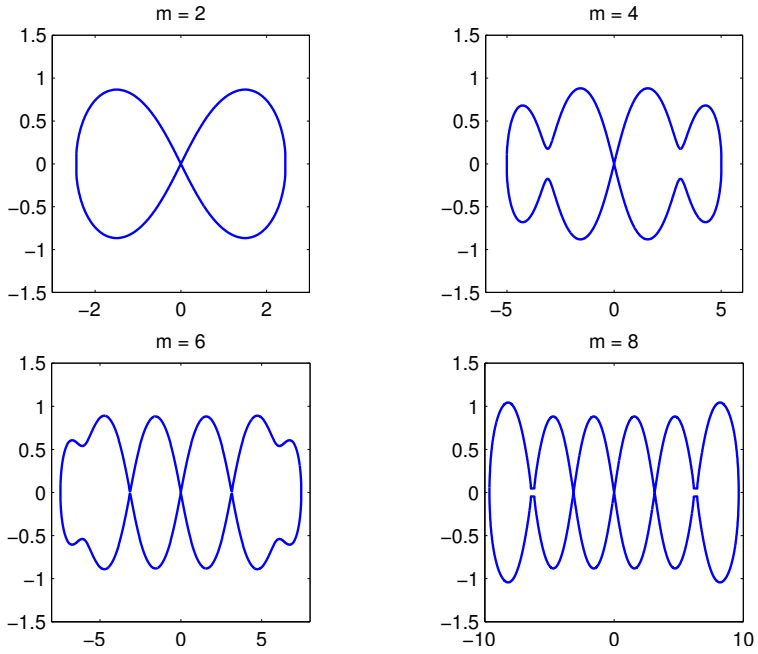

Figure 2.2.2: The boundary of the region within which $|r_m(x)| \leq 1$, where $r_m(x)$ is the $[m/m]$ Padé approximant to the cosine function, for $m = 2, 4, 6, 8$.

2.2.3 Exploiting Padé approximants of the exponential

Let $r_m(x)$ denote the $[m/m]$ Padé approximant to e^x. Al-Mohy and Higham [4, Sec. 3] show that if $\rho(e^{-x}r_m(X) - I) < 1$ and $\rho(X) < \min\{\, |t| : q_m(t) = 0 \,\}$ then

$$r_m(X) = e^{X + h_{2m+1}(X)}, \tag{2.2.7}$$

where $h_{2m+1}(X) := \log(e^{-X}r_m(X))$, with log the principal logarithm. As shown in [4, Sec. 5], h_{2m+1} is an odd function, so $h_{2m+1}(-X) = -h_{2m+1}(X)$. Hence

$$r_m(-X) = e^{-(X + h_{2m+1}(X))}. \tag{2.2.8}$$

Furthermore, writing $h_{2m+1}(X) = \sum_{k=m}^{\infty} c_{m,k} X^{2k+1}$ one can easily show that

$$h_{2m+1}(iX) = i \sum_{k=m}^{\infty} (-1)^k c_{m,k} X^{2k+1} =: i\phi(X). \tag{2.2.9}$$

Now [57, (12.2)]

$$\cos X = \frac{e^{iX} + e^{-iX}}{2}, \quad \sin X = \frac{e^{iX} - e^{-iX}}{2i}, \tag{2.2.10}$$

which suggests using as approximations the rational functions with real coefficients

$$c_m(x) = \frac{r_m(ix) + r_m(-ix)}{2}, \qquad s_m(x) = \frac{r_m(ix) - r_m(-ix)}{2i}. \tag{2.2.11}$$

From (2.2.7) and (2.2.9) we have

$$c_m(X) = \frac{r_m(iX) + r_m(-iX)}{2} = \frac{e^{i(X+\phi(X))} + e^{-i(X+\phi(X))}}{2} = \cos(X + \phi(X)) \quad (2.2.12)$$

and

$$s_m(X) = \frac{r_m(iX) - r_m(-iX)}{2i} = \frac{e^{i(X+\phi(X))} - e^{-i(X+\phi(X))}}{2i} = \sin(X + \phi(X)), \quad (2.2.13)$$

so the two approximations have the same backward error, $\phi(X)$. To bound this backward error, for a power series $p(x)$ let $\widetilde{p}(x)$ be the power series where each coefficient of $p(x)$ is replaced by its modulus. Then $\widetilde{\phi}(x) = \sum_{k=m}^{\infty} |c_{k,m}| x^{2k+1}$ and so we have

$$\frac{\|\phi(X)\|}{\|X\|} \le \frac{\widetilde{\phi}(\|X\|)}{\|X\|} = \frac{\widetilde{h}_{2m+1}(\|X\|)}{\|X\|}, \quad (2.2.14)$$

since $\widetilde{\phi}(x) = \widetilde{h}_{2m+1}(x)$. From [4, Sec. 3] it follows that $\|\phi(X)\|/\|X\| \le u$ if $\alpha_p(X) \le \theta_m$, where the θ_m are given in Table 2.4.1 (reproduced from [56, Table 2.1]) and p is chosen to minimize α_p in (2.2.5) subject to $m \ge p(p-1)$.

For the cosine we will use c_m but for the sine we will use a mixture of s_m and Padé approximants to $\sin x$. We now need to devise a strategy for choosing the parameters s and m. We have $\theta_m \ge \beta_m$ for $m \le 21$, as can partially be seen from Tables 2.4.1 and 2.5.1, which means that less scaling is required if we approximate $\sin X$ by s_m than by the Padé approximant r_m. On the other hand, the numerator and denominator polynomials of c_m and s_m are of higher degree than those of the Padé approximants of the cosine and sine, so c_m and s_m will be more expensive to evaluate. In all cases considered here, s_m is a $[2m-1/2m]$ order rational approximant whereas c_m is of order $[2m/2m]$. For example

$$s_3(x) = \frac{x + 7x^3/60 - x^5/600}{1 + x^2/20 + x^4/600 + x^6/14400},$$

$$c_3(x) = \frac{1 - 9x^2/20 + 11x^4/600 - x^6/14400}{1 + x^2/20 + x^4/600 + x^6/14400}.$$

In sections 2.4–2.6 we explain how to balance the degree of the rational approximant with the amount of scaling in order to achieve the desired backward error at minimal cost.

2.2.4 Backward error propagation in multiple angle formulas

We have shown how to achieve a small backward error for the rational approximation of the sine or cosine of the scaled matrix. We now show that this backward error propagates linearly through the multiple angle formula phase of the algorithm (lines 3–5 of Algorithm 2.1.1), resulting in a small overall backward error. We state the result for the cosine; an analogous result holds for the sine.

Lemma 2.2.1. *Let $A \in \mathbb{C}^{n\times n}$ and $X = \eta^{-s}A$ for a positive integer η and nonnegative integer s, and suppose that $r(X) = \cos(X + \Delta X)$ for a rational function r. Then the approximation Y from the "scaling and multiple angle" method satisfies $Y = \cos(A + \Delta A)$, where $\Delta A = \eta^s \Delta X$ in exact arithmetic, and hence $\|\Delta A\|/\|A\| = \|\Delta X\|/\|X\|$.*

Proof. We prove the result for $\eta = 2$ (the double angle formula) for simplicity, though the same argument can be applied for any integer η (and the corresponding multiple angle formula).

By assumption, the initial approximation to the cosine from the rational approximant is $C_0 = \cos(X + \Delta X)$, where $X = 2^{-s}A$. Applying the double angle formula s times gives

$$\begin{aligned} C_1 &= 2C_0^2 - I \quad &&= \cos(2X + 2\Delta X), \\ C_2 &= 2C_1^2 - I \quad &&= \cos(4X + 4\Delta X), \\ &\vdots \\ C_s &= 2C_{s-1}^2 - I &&= \cos(2^s X + 2^s \Delta X). \end{aligned}$$

Therefore we have $\cos A \approx Y = C_s = \cos(A + \Delta A)$ where $\Delta A = 2^s \Delta X$, and $\|\Delta X\|/\|X\| = \|2^s \Delta X\|/\|2^s X\| = \|\Delta A\|/\|A\|$. □

Lemma 2.2.1 shows that there is no growth in the relative backward error during the multiple angle phase in exact arithmetic. Hence by choosing the parameters s and m so that $\|\Delta X\|/\|X\| \le u$ we achieve an overall backward error bounded by u. This contrasts with the algorithms for the matrix cosine in [46], [57, Alg. 12.6], [95], which choose r to make $\|r(X) - \cos X\|$ small, since the overall error $\|r(A) - \cos A\|$ bears no simple relation to $\|r(X) - \cos X\|$.

2.3 Recomputing the cosine and sine of 2×2 matrices

If we begin with an initial (real) Schur decomposition of our input matrix A, so that we are working with upper (quasi-)triangular matrices, then we can obtain higher overall accuracy by explicitly computing (instead of approximating) the diagonal blocks (including all of the first superdiagonal) explicitly throughout the algorithm. This idea has been used to good effect in recent algorithms for the matrix exponential [4], the logarithm[1] [6], and matrix powers [60], [61].

For 2×2 triangular matrices $T = \left[\begin{smallmatrix} \lambda_1 & t \\ 0 & \lambda_2 \end{smallmatrix}\right]$ it is known that [57, (4.16)]

$$f(T) = \begin{bmatrix} f(\lambda_1) & tf[\lambda_1, \lambda_2] \\ 0 & f(\lambda_2) \end{bmatrix}, \tag{2.3.1}$$

where $f[\lambda_1, \lambda_2]$ is the divided difference

$$f[\lambda_1, \lambda_2] = \begin{cases} (f(\lambda_1) - f(\lambda_2))/(\lambda_1 - \lambda_2), & \text{if } \lambda_1 \neq \lambda_2, \\ f'(\lambda_1), & \text{if } \lambda_1 = \lambda_2. \end{cases} \tag{2.3.2}$$

For the cosine and sine we can avoid subtractive cancellation when $\lambda_1 \approx \lambda_2$ by computing

$$\cos[\lambda_1, \lambda_2] = -\left[\sin\left(\frac{\lambda_1 + \lambda_2}{2}\right)\sin\left(\frac{\lambda_1 - \lambda_2}{2}\right)\right] / \left(\frac{\lambda_1 - \lambda_2}{2}\right), \tag{2.3.3}$$

$$\sin[\lambda_1, \lambda_2] = \left[\sin\left(\frac{\lambda_1 - \lambda_2}{2}\right)\cos\left(\frac{\lambda_1 + \lambda_2}{2}\right)\right] / \left(\frac{\lambda_1 - \lambda_2}{2}\right). \tag{2.3.4}$$

A 2×2 block of a real upper quasi-triangular matrix computed by the LAPACK Schur decomposition code `dgees` [11] has the form

$$B = \begin{bmatrix} a & b \\ c & a \end{bmatrix}, \quad bc < 0. \tag{2.3.5}$$

We can use the polynomial definition of a matrix function [57, Def. 1.4] to show that

$$\cos B = \begin{bmatrix} \cos a \cosh \theta & -\theta^{-1} b \sin a \sinh \theta \\ -\theta^{-1} c \sin a \sinh \theta & \cos a \cosh \theta \end{bmatrix}, \tag{2.3.6}$$

$$\sin B = \begin{bmatrix} \sin a \cosh \theta & b \cos a \sinh \theta \\ \theta^{-1} c \cos a \sinh \theta & \sin a \cosh \theta \end{bmatrix}, \tag{2.3.7}$$

[1] We will also use this technique in chapter 3 within our algorithms for the matrix logarithm and its Fréchet derivatives.

where $\theta = (-bc)^{1/2}$. Assuming that accurate implementations of sinh and cosh are available these formulae will compute the cosine and sine of 2×2 matrices to high componentwise accuracy without any subtractive cancellation.

2.4 Algorithm for the matrix cosine

We now build an algorithm for the matrix cosine based upon the rational approximation c_m of (2.2.11) along with the double angle formula.

The cost of our algorithm will be dominated by the matrix multiplications performed when forming the rational approximant and during the repeated application of the double angle formula. Each multiplication costs $2n^3$ flops for full matrices or $n^3/3$ flops for (quasi-)triangular matrices if a Schur decomposition is used.

We choose the two parameters m, the order of approximation, and s, the number of scalings, to minimize the number of matrix multiplications whilst ensuring a backward error of order u (in exact arithmetic).

First we discuss which values of m need to be considered for the approximant $c_m(X)$. We will consider $m = 1\colon m_{\max}$, where $m_{\max}$ is defined as the largest m for which the denominator of $c_m(X)$ has condition number, in the appropriate norm, smaller than 10 for any X in the region where $\alpha_p(X) \le \theta_m$. The appropriate norm is as follows: for any $\epsilon > 0$ there exists a norm $\|\cdot\|_\epsilon$ satisfying $\|X\|_\epsilon \le \rho(X) + \epsilon \le \alpha_p + \epsilon$, where we will choose $\epsilon = u$ to be the unit roundoff. The corresponding condition number of the denominator $q_m(X)$ can be bounded above (see [4, p. 982]) as

$$\|q_m(X)\|_\epsilon \|q_m(X)^{-1}\|_\epsilon \le \tilde{q}_m(\alpha_p(X) + \epsilon) \sum_{k=0}^{\infty} |a_k| (\alpha_p(X) + \epsilon)^k \tag{2.4.1}$$

$$\le \tilde{q}_m(\theta_m + \epsilon) \sum_{k=0}^{\infty} |a_k| (\theta_m + \epsilon)^k =: \kappa_m, \tag{2.4.2}$$

where $\sum_{k=0}^{\infty} a_k x^k$ is the Taylor series of $q_m(x)^{-1}$ and $\tilde{q}_m$ is the same polynomial as q_m with all coefficients replaced by their absolute values.

Using 250 digit arithmetic from the Symbolic Math Toolbox and calculating the first 350 terms of the Taylor series in the bound (2.4.2), we found that for the matrix cosine this means considering $m_{\max} = 21$, where θ_{21} is reduced from 13.95, which has corresponding condition number 10.75, to 13 with condition number 7.86 for additional

Table 2.4.1: The number of matrix multiplications $\pi(c_m(X))$ required to evaluate $c_m(X)$, values of θ_m, values of p to be considered, and an upper bound κ_m for the condition number of the denominator polynomial, in the $\|X\|_\epsilon$ norm, of c_m (and s_m) for $\alpha_p(X) \leq \theta_m$. Values of m for which $\pi(c_m(X)) = \pi(c_{m+1}(X))$ are not shown as they provide no benefit. For $m = 21$, θ_{21} is artificially reduced from 13.95 to 13 in order to ensure $\kappa_{21} \leq 10$.

m	1	2	3	4	6	8
$\pi(c_m(X))$	1	2	3	4	5	6
θ_m	3.6e-8	5.3e-4	1.5e-2	8.5e-2	5.4e-1	1.47
p	1	2	2	2	3	3
κ_m	1	1	1	1	1.01	1.07

m	10	12	15	18	21
$\pi(c_m(X))$	7	8	9	10	11
θ_m	2.8	4.46	7.34	10.54	13.95 (13)
p	3	3, 4	3, 4	3, 4	3, 4, 5
κ_m	1.2	1.54	2.53	4.89	7.86

stability. Note that this part of our analysis also applies to the matrix sine since c_m and s_m share the same denominator polynomial.

This condition ensures that solving the multiple right-hand side system to form the rational approximant does not introduce significant errors into the computation—at least as long as $\|\cdot\|_\epsilon$ is not too badly scaled. The condition numbers κ_m for each m of interest are shown in Table 2.4.1.

Now that we have a range of m to consider, we must find those values for which c_m can be formed in the smallest number of matrix multiplications. There are many ways to evaluate the rational approximants: in section 2.8 we show that applying the Paterson–Stockmeyer scheme [57, pp. 73–74], [88] is more efficient than explicit computation of the necessary powers. Evaluating the numerator and denominator polynomials using the Paterson–Stockmeyer scheme as described we see that, for example, $c_7(X)$ and $c_8(X)$ can both be formed using a minimum of 6 matrix multiplications. Since $c_8(X)$ can be applied to X with a larger value of $\alpha_p(X)$ it renders $c_7(X)$ redundant.

Table 2.4.1 contains the relevant values of m and θ_m along with the number of matrix multiplications required to form $c_m(X)$. It also shows the values of p that we need to consider in order to minimize $\alpha_p(X)$ in (2.2.5) for each m.

Finally, we consider the choice of the parameters s (where $X = 2^{-s}A$) and m. It may sometimes be cheaper to perform a stronger scaling on A and use a lower order approximant. In particular, each invocation of the double angle formula costs one matrix multiplication and therefore it is worth increasing s by q if more than q multiplications can be saved when evaluating the rational approximant. (Note that this logic applies equally well to full matrices and triangular matrices.) From Table 2.4.1 we see that there are a number of places where such an increase in s is desirable. For example when $\theta_{10} < \alpha_3(X) \le 2\theta_8$ we can perform one extra scaling and use $c_8(X)$ instead of $c_{12}(X)$, saving one multiplication.

We also point out that computing $\alpha_p(X)$ can sometimes require more powers of X than the rational approximant. For instance when $m = 2$ we need $\alpha_2(X) = \min(\|X^4\|^{1/4}, \|X^6\|^{1/6})$, but forming $c_2(X)$ requires only X^4 (we use explicit powers for $m \le 4$, which have the same cost as the Paterson–Stockmeyer scheme: see Table 2.8.1).

However, we will use the 1-norm, for which we can estimate the norm of X^ℓ for any integer $\ell > 0$, without calculating the matrix power explicitly, using the block 1-norm estimator of Higham and Tisseur [66]. A further optimization is that after eliminating $m = 1$ as a possibility, for example, all higher order approximants $c_m(X)$ require X^4 so we can compute and store X^4 and hence use $\|X^4\|_1^{1/4}$ rather than an estimate of it in the logical tests. A similar optimization can be performed after eliminating $m = 2$ and $m = 6$. Taking all these issues into account leads to the following algorithm to determine the parameters s and m. The many logical tests are the price we pay for making the most efficient choice of parameters.

Algorithm 2.4.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the parameters s and m such that the algorithm for approximating* $\cos A$ *based on $c_m(2^{-s}A)$ and the double angle formula produces (in exact arithmetic) a backward error bounded by u. The algorithm uses the parameters θ_m given in Table* 2.4.1.

1 $s = 0$

2 Compute and store A^2.

3 $\alpha_1(A) = \|A^2\|_1^{1/2}$

4 if $\alpha_1(A) \le \theta_1$, $m = 1$, quit, end

5 Compute and store A^4.

6 $d_4 = \|A^4\|_1^{1/4}$

7 Estimate $d_6 = \|A^6\|_1^{1/6}$.

8 $\alpha_2(A) = \max(d_4, d_6)$

9 if $\alpha_2(A) \le \theta_2$, $m = 2$, quit, end

10 Compute and store A^6.

11 $d_6 = \|A^6\|_1^{1/6}$ % Compute exact value and update α_2.

12 $\alpha_2(A) = \max(d_4, d_6)$

13 if $\alpha_2(A) \le \theta_3$, $m = 3$, quit, end

14 if $\alpha_2(A) \le \theta_4$, $m = 4$, quit, end

15 Estimate $d_8 = \|A^8\|_1^{1/8}$.

16 $\alpha_3(A) = \max(d_6, d_8)$

17 if $\alpha_3(A) \le \theta_6$, $m = 6$, quit, end

18 Compute and store A^8.

19 $d_8 = \|A^8\|_1^{1/8}$ % Compute exact value and update α_3.

20 $\alpha_3(A) = \max(d_6, d_8)$

21 if $\alpha_3(A) \le \theta_8$, $m = 8$, quit, end

22 if $\alpha_3(A) \le \theta_{10}$, $m = 10$, quit, end

23 if $\alpha_3(A) \le 2\theta_8$, $s = 1$, $m = 8$, quit, end

24 Estimate $d_{10} = \|A^{10}\|_1^{1/10}$.

25 $\alpha_4(A) = \max(d_8, d_{10})$

26 $\alpha_{3/4} = \min(\alpha_3(A), \alpha_4(A))$

27 if $\alpha_{3/4}(A) \le \theta_{12}$, $m = 12$, quit, end

28 if $\alpha_3(A) \le 2\theta_{10}$, $s = 1$, $m = 10$, quit, end

29 if $\alpha_3(A) \le 4\theta_8$, $s = 2$, $m = 8$, quit, end

30 % Note that $s = 0$ at this point

31 if $\alpha_{3/4}(A) \le \theta_{15}$, $m = 15$, quit, end

32 if $\alpha_{3/4}(A) \le 2\theta_{12}$, $s = s + 1$, $m = 12$, quit, end

33 if $\alpha_3(A) \le 4\theta_{10}$, $s = s + 2$, $m = 10$, quit, end

34 if $\alpha_3(A) \le 8\theta_8$, $s = s + 3$, $m = 8$, quit, end

35 if $\alpha_{3/4}(A) \le \theta_{18}$, $m = 18$, quit, end

36 if $\alpha_{3/4}(A) \le 2\theta_{15}$, $s = s + 1$, $m = 15$, quit, end

37 if $\alpha_{3/4}(A) \le 4\theta_{12}$, $s = s + 2$, $m = 12$, quit, end

38 if $\alpha_3(A) \le 8\theta_{10}$, $s = s + 3$, $m = 10$, quit, end

39 Estimate $d_{12} = \|A^{12}\|_1^{1/12}$.

40 $\alpha_5(A) = \max(d_{10}, d_{12})$

41 $\alpha_{3/4/5}(A) = \min(\alpha_3, \alpha_4, \alpha_5)$

42 if $\alpha_{3/4/5}(A) \le \theta_{21}$, $m = 21$, quit, end

43 % A needs to be scaled. After scaling, $6.5 \approx \theta_{21}/2 < \alpha_{3/4/5}(A) \le \theta_{21}$.

44 $s = \lceil \log_2(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$

45 $\alpha_3(A) = \alpha_3(A)/2^s$

46 $\alpha_{3/4}(A) = \alpha_{3/4}(A)/2^s$

47 $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/2^s$

48 Execute lines 31–38.

49 $m = 21$

We can now present our full algorithm for computing the matrix cosine, which is backward stable in exact arithmetic. The algorithm is written using an initial Schur decomposition but a transformation-free algorithm can be obtained by removing lines 5, 8, and 10 and replacing line 1 with $T = A$.

Algorithm 2.4.2. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes $C = \cos A$. The algorithm is designed for use with IEEE double precision arithmetic.*

1 Compute the (real if $A \in \mathbb{R}^{n\times n}$) Schur decomposition $A = QTQ^*$.

2 Obtain s and m from Algorithm 2.4.1 applied to T.

3 $T \leftarrow 2^{-s}T$ and $T^k \leftarrow 2^{-sk}T^k$ for any powers T^k stored during line 2.

4 Compute $C = c_m(T)$: use the Paterson–Stockmeyer scheme to compute the numerator $p_m(X)$ and denominator $q_m(X)$ and then solve $q_m(T)C = p_m(T)$.

5 Recompute the diagonal blocks of C using (2.3.1), (2.3.3), (2.3.6)

6 for $j = 1 : s$

7 $\quad C \leftarrow 2C^2 - I$

8 Recompute the diagonal blocks of $C = \cos(2^j T)$ using (2.3.1), (2.3.3), (2.3.6).

9 end

10 $C \leftarrow QCQ^*$

Cost: $(28 + (\pi + s + 1)/3)n^3$ flops where π denotes the number of matrix multiplications needed to form $c_m(x)$. The transformation-free version of the algorithm costs $(2(s + \pi) + 8/3)n^3$ flops. Comparing these two we see that it is cheaper to use the Schur decomposition if $\pi + s \geq 16$, The latter inequality is readily satisfied, for example if $A = 448I$, and using the Schur decomposition allows us to carry out the accurate recomputation of the diagonal blocks.

2.5 Algorithm for the matrix sine

In this section we design an algorithm to compute the matrix sine. Whereas for the cosine we used the rational approximations $c_m(X)$, for the sine we must consider both $s_m(X)$ and the Padé approximants. Another difference from the cosine case is that we will use the triple angle formula $\sin(3X) = 3\sin X - 4\sin^3 X$ instead of the double-angle formula $\sin(2X) = 2\sin X \cos X$, as the latter formula requires $\cos X$.

Table 2.5.1 shows the number of matrix multiplications required to form the Padé approximants $r_m(X)$, denoted $\pi(r_m(X))$. The values of $\pi(s_m(X))$ are $\pi(s_1(X)) = 1$ and for $m \geq 2$ we have $\pi(s_m(X)) = \pi(c_m(X)) + 1$, where $\pi(c_m(X))$ is given in Table 2.4.1. The table also contains the values of p that we need to consider and the values β_m such that $\alpha_p(X) \leq \beta_m$ implies the backward error of the result is bounded by u in exact arithmetic. The last row of the table shows an upper bound κ_m on the condition number of the denominator polynomial of $r_m(X)$ for $\alpha_p(X) \leq \beta_m$. For $s_m(X)$ the values for θ_m, p, and κ_m exactly match those already given for the cosine in Table 2.4.1: the values of θ_m match due to the relationship to the matrix exponential (see section 2.2.3), the p match due to the degree of the approximants, and the κ_m match as c_m and s_m have identical denominator polynomials.

For the Padé approximants r_m, the maximum order of approximation considered is $m = 9$ in order to ensure the validity of the bounds, as explained in section 2.2.1. On the other hand, as for the matrix cosine, we consider up to $m = 21$ for the alternative rational approximants $s_m(X)$. The use of $s_m(X)$ allows larger p within the values

Table 2.5.1: The number of matrix multiplications $\pi(\cdot)$ required to evaluate $r_m(x)$, values of β_m in (2.2.6), values of p to be considered, and an upper bound κ_m for the condition number of the denominator polynomial of r_m for $\alpha_p(X) \le \beta_m$. Values of m for which $\pi(r_m(X)) = \pi(r_{m+1}(X))$ are not shown as they provide no benefit. The values for β_9 has been artificially reduced from 1.14 to 8.81e-1 as required by our backward error analysis in section 2.2.1; this changes the value of κ_9.

m	1	3	5	7	9
$\pi(r_m(X))$	0	2	3	4	5
β_m	2.58e-8	8.93e-3	1.47e-1	5.36e-1	1.14 (8.81e-1)
p	1	2	2	3	3
κ_m	1	1	1	1.01	1.05 (1.03)

$\alpha_p(X)$ used in our backward error bounds and can therefore reduce the amount of scaling required. For example, suppose A is nilpotent of degree 10 with $\alpha_3(A) \gg \beta_9$. Using only Padé approximants we would require a large amount of scaling but since A is nilpotent we know that $\alpha_5(A) = 0$, allowing the use of $s_{21}(A)$ with no scaling required.

We scale $X = 3^{-s}A$ and each application of the triple angle formula requires two matrix multiplications. Hence it is beneficial to increase s by q if we can save more than $2q$ matrix multiplications in forming s_m. For example, suppose that $\theta_{12} < \min(\alpha_3(X), \alpha_4(X)) \le \theta_{15}$ but additionally $\alpha_3(X) \le 9\beta_9$; then by performing two extra scalings we can use $r_9(X)$ at a cost of $4 + \pi(r_9(X)) = 9$ multiplications, as opposed to $\pi(s_{15}(X)) = 10$ multiplications. For each X we choose among the r_m and s_m approximants, whilst considering extra scaling as above, always striving for the parameters with minimal cost subject to the backward error constraint.

This discussion leads to the following parameter selection algorithm.

Algorithm 2.5.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the scaling parameter s and rational approximant φ, equal to r_m or s_m, such that the algorithm for approximating* $\sin A$ *based on $\varphi(3^{-s}A)$ and the triple angle formula produces (in exact arithmetic) a backward error bounded by u. The algorithm uses the parameters θ_m and β_m given in Tables 2.4.1 and 2.5.1, respectively.*

1 $s = 0$

2 Estimate $d_2 = \|A^2\|_1^{1/2}$.

3 $\alpha_1(A) = d_2$

4 if $\alpha_1(A) \le \beta_1$, $\varphi(x) = r_1(x)$, quit, end

5 Compute and store A^2.

6 $d_2 = \|A^2\|_1^{1/2}$ % Compute exact value and update α_1.

7 $\alpha_1(A) = d_2$

8 if $\alpha_1(A) \le \theta_1$, $\varphi(x) = s_1(x)$, quit, end

9 Estimate $d_4 = \|A^4\|_1^{1/4}$ and $d_6 = \|A^6\|_1^{1/6}$.

10 $\alpha_2(A) = \max(d_4, d_6)$

11 if $\alpha_2(A) \le \beta_3$, $\varphi(x) = r_3(x)$, quit, end

12 Compute and store A^4.

13 $d_4 = \|A^4\|_1^{1/4}$ % Compute exact value and update α_2.

14 $\alpha_2(A) = \max(d_4, d_6)$

15 if $\alpha_2(A) \le \beta_5$, $\varphi(x) = r_5(x)$, quit, end

16 Compute and store A^6.

17 $d_6 = \|A^6\|_1^{1/6}$

18 Estimate $d_8 = \|A^8\|_1^{1/8}$.

19 $\alpha_3(A) = \max(d_6, d_8)$

20 if $\alpha_3(A) \le \beta_7$, $\varphi(x) = r_7(x)$, quit, end

21 if $\alpha_3(A) \le \beta_9$, $\varphi(x) = r_9(x)$, quit, end

22 if $\alpha_3(A) \le 3\beta_7$, $s = 1$, $\varphi(x) = r_7(x)$, quit, end

23 if $\alpha_3(A) \le 3\beta_9$, $s = 1$, $\varphi(x) = r_9(x)$, quit, end

24 if $\alpha_3(A) \le \theta_{10}$, $\varphi(x) = s_{10}(x)$, quit, end

25 if $\alpha_3(A) \le 9\beta_7$, $s = 2$, $\varphi(x) = r_7(x)$, quit, end

26 Compute and store A^8.

27 $d_8 = \|A^8\|_1^{1/8}$ % Compute exact value and update α_3.

28 $\alpha_3(A) = \max(d_6, d_8)$

29 Compute and store A^{10}.

30 $d_{10} = \|A^{10}\|_1^{1/10}$

31 $\alpha_4(A) = \max(d_8, d_{10})$

32 $\alpha_{3/4} = \min(\alpha_3(A), \alpha_4(A))$

33 % Note that $s = 0$ at this point.

34 if $\alpha_{3/4}(A) \le \theta_{12}$, $\varphi(x) = s_{12}(x)$, quit, end

35 if $\alpha_3(A) \le 9\beta_9$, $s = s + 2$, $\varphi(x) = r_9(x)$, quit, end

36 if $\alpha_{3/4}(A) \le \theta_{15}$, $\varphi(x) = s_{15}(x)$, quit, end

37 if $\alpha_3(A) \le 3\theta_{10}$, $s = s + 1$, $\varphi(x) = s_{10}(x)$, quit, end

38 if $\alpha_{3/4}(A) \le \theta_{18}$, $\varphi(x) = s_{18}(x)$, quit, end

39 if $\alpha_{3/4}(A) \le 3\theta_{12}$, $s = s + 1$, $\varphi(x) = s_{12}(x)$, quit, end

40 Estimate $d_{12} = \|A^{12}\|_1^{1/12}$.

41 $\alpha_5(A) = \max(d_{10}, d_{12})$

42 $\alpha_{3/4/5} = \min(\alpha_3, \alpha_4, \alpha_5)$

43 if $\alpha_{3/4/5}(A) \le \theta_{21}$, $\varphi(x) = s_{21}(x)$, quit, end

44 % A needs to be scaled. After scaling, $4.3 \approx \theta_{21}/3 < \alpha_{3/4/5}(A) \le \theta_{21}$.

45 $s = \lceil \log_3(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$

46 $\alpha_3(A) = \alpha_3(A)/3^s$

47 $\alpha_{3/4}(A) = \alpha_{3/4}(A)/3^s$

48 $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/3^s$

49 if $\alpha_3(A) \le 9\beta_7$, $s = s + 2$, $\varphi(x) = r_7(x)$, quit, end

50 Execute lines 34–39.

51 $\varphi(x) = s_{21}(x)$

We now present our full algorithm for computing the matrix sine. As for the matrix cosine, the algorithm uses a Schur decomposition. To obtain a transformation-free algorithm lines 5, 8, and 10 should be removed and line 1 replaced with $T = A$.

Algorithm 2.5.2. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes $S = \sin A$. The algorithm is designed for use with IEEE double precision arithmetic.*

1 Compute the (real if $A \in \mathbb{R}^{n\times n}$) Schur decomposition $A = QTQ^*$.

2 Obtain s and $\varphi(x)$ from Algorithm 2.5.1 applied to T.

3 $T \leftarrow 3^{-s}T$ and $T^k \leftarrow 3^{-sk}T^k$ for any powers of T stored during line 2.

4 Compute $S = \varphi(T)$: use the Paterson–Stockmeyer scheme to compute the numerator $p(X)$ and denominator $q(X)$ and then solve $q(T)S = p(T)$.

5 Recompute the diagonal blocks of S using (2.3.1), (2.3.4), (2.3.7)

6 for $j = 1 : s$

Table 2.6.1: Number of matrix multiplications π_m required to evaluate both c_m and s_m.

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

7 $S \leftarrow S(3I - 4S^2)$

8 Recompute the diagonal blocks of $S = \sin(3^j T)$ using (2.3.1), (2.3.4), (2.3.7).

9 end

10 $S \leftarrow QSQ^*$

Cost: $(28 + (\pi + 2s + 1)/3)n^3$ flops, where π denotes the number of matrix multiplications needed to form $\varphi(x)$. The corresponding transformation-free algorithm costs $(2(\pi + 2s) + 8/3)n^3$ flops. Comparing these two costs we see that it is cheaper to use the Schur decomposition if $\pi + 2s \geq 16$.

2.6 Algorithm for simultaneous computation of the matrix cosine and sine

We now design an algorithm to compute the matrix cosine and sine simultaneously, a requirement that arises in the evaluation of (2.1.4), for example. We use the rational approximants $c_m(x)$ and $s_m(x)$ introduced in section 2.2.3, since they have the same denominator polynomial: this saves a significant amount of computation since we need only compute one denominator for both approximants and furthermore we can reuse an LU factorization of the denominator when computing $c_m(X)$ and $s_m(X)$.

Since we are now computing both the cosine and sine we can use the double angle formulas for both. However for the cosine there are two such formulas for us to choose from:

$$\cos(2X) = 2\cos^2 X - I, \qquad \cos(2X) = I - 2\sin^2 X. \tag{2.6.1}$$

We have found empirically that using $\cos(2X) = I - 2\sin^2 X$ generally gives more accurate computed results, sometimes significantly so, though a theoretical explanation for this observation is currently lacking.

In Table 2.6.1 we show the number of matrix multiplications π_m required to form both $c_m(x)$ and $s_m(x)$ using the Paterson–Stockmeyer scheme (see section 2.8). Since each invocation of the double angle formulas for the cosine and sine costs two matrix multiplications in total, it is worth performing q extra scalings if more than $2q$ matrix multiplications can be saved in the formation of the rational approximant. This results in the following parameter selection algorithm.

Algorithm 2.6.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the parameters s and m such that the algorithm for approximating $\cos A$ and $\sin A$ based on $c_m(2^{-s}A)$ and $s_m(2^{-s}A)$ and the double angle formulas produces (in exact arithmetic) backward errors bounded by u for both functions. The algorithm uses the parameters θ_m given in Table 2.4.1.*

1 $s = 0$
2 Compute and store A^2.
3 $d_2 = \|A^2\|_1^{1/2}$
4 $\alpha_1(A) = d_2$
5 if $\alpha_1(A) \le \theta_1$, $m = 1$, quit, end
6 Compute and store A^4.
7 $d_4 = \|A^4\|_1^{1/4}$
8 Estimate $d_6 = \|A^6\|_1^{1/6}$.
9 $\alpha_2(A) = \max(d_4, d_6)$
10 if $\alpha_2(A) \le \theta_2$, $m = 2$, quit, end
11 Compute and store A^6.
12 $d_6 = \|A^6\|_1^{1/6}$ % Compute exact value and update α_2.
13 $\alpha_2(A) = \max(d_4, d_6)$
14 if $\alpha_2(A) \le \theta_3$, $m = 3$, quit, end
15 if $\alpha_2(A) \le \theta_4$, $m = 4$, quit, end
16 if $\alpha_2(A) \le \theta_5$, $m = 5$, quit, end
17 Estimate $d_8 = \|A^8\|_1^{1/8}$.
18 $\alpha_3(A) = \max(d_6, d_8)$
19 if $\alpha_3(A) \le \theta_6$, $m = 6$, quit, end
20 Compute and store A^8.

21 $d_8 = \|A^8\|_1^{1/8}$ % Compute exact value and update α_3.

22 $\alpha_3(A) = \max(d_6, d_8)$

23 if $\alpha_3(A) \le \theta_8$, $m = 8$, quit, end

24 Compute and store A^{10}.

25 if $\alpha_3(A) \le \theta_{10}$, $m = 10$, quit, end

26 Compute and store A^{12}.

27 $d_{10} = \|A^{10}\|_1^{1/10}$

28 $\alpha_4(A) = \max(d_8, d_{10})$

29 $\alpha_{3/4}(A) = \min(\alpha_3, \alpha_4)$

30 if $\alpha_{3/4}(A) \le \theta_{12}$, $m = 12$, quit, end

31 if $\alpha_{3/4}(A) \le \theta_{14}$, $m = 14$, quit, end

32 % Note that $s = 0$ at this point.

33 if $\alpha_{3/4}(A) \le \theta_{16}$, $m = 16$, quit, end

34 if $\alpha_{3/4}(A) \le 2\theta_{12}$, $s = s + 1$, $m = 12$, quit, end

35 if $\alpha_{3/4}(A) \le \theta_{18}$, $m = 18$, quit, end

36 if $\alpha_{3/4}(A) \le 2\theta_{14}$, $s = s + 1$, $m = 14$, quit, end

37 $d_{12} = \|A^{12}\|_1^{1/12}$

38 $\alpha_5(A) = \max(d_{10}, d_{12})$

39 $\alpha_{3/4/5}(A) = \min(\alpha_3, \alpha_4, \alpha_5)$

40 if $\alpha_{3/4/5}(A) \le \theta_{21}$, $m = 21$, quit, end

41 % A needs to be scaled. After scaling, $6.5 \approx \theta_{21}/2 \le \alpha_{3/4/5}(A) \le \theta_{21}$.

42 $s = \lceil \log_2(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$

43 $\alpha_{3/4}(A) = \alpha_{3/4}(A)/2^s$

44 $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/2^s$

45 Execute lines 33–36.

46 $m = 21$

We now state our algorithm for computing the matrix cosine and sine. To obtain the corresponding transformation-free algorithm lines 7, 11, 13, and 15–16 should be removed and line 1 replaced by $T = A$.

Algorithm 2.6.2. *Given $A \in \mathbb{C}^{n\times n}$ the following algorithm computes both $C = \cos A$ and $S = \sin A$. The common denominator of $c_m(x)$ and $s_m(x)$ is denoted by $\omega(x)$ so that $c_m(x) = \widehat{c}_m(x)/\omega(x)$ and $s_m(x) = \widehat{s}_m(x)/\omega(x)$. This algorithm is designed for use with IEEE double precision arithmetic.*

1 Compute the (real if $A \in \mathbb{R}^{n\times n}$) Schur decomposition $A = QTQ^*$.
2 Obtain s and m from Algorithm 2.6.1 applied to T.
3 $T \leftarrow 2^{-s}T$ and $T^k \leftarrow 2^{-sk}T^k$ for any powers of T stored during line 2.
4 Compute the shared denominator $\omega_m(X)$ and the numerators $\widehat{c}_m(X)$ and $\widehat{s}_m(X)$ using the Paterson–Stockmeyer method (section 2.8).
5 Compute an LU factorization with partial pivoting $LU = \omega_m(X)$.
6 Compute $S = U^{-1}L^{-1}\widehat{s}_m(X)$ and $C = U^{-1}L^{-1}\widehat{c}_m(X)$ by substitution using the LU factorization.
7 Recompute the block diagonals of S and C using (2.3.1)–(2.3.7).
8 for $j = 1 : s$
9 $\quad S_{old} = S$
10 $\quad S = 2SC$
11 $\quad$ Recompute the block diagonals of $S = \sin(2^jT)$ using (2.3.1), (2.3.4), (2.3.7).
12 $\quad C = I - 2S_{old}^2$
13 $\quad$ Recompute the block diagonals of $C = \cos(2^jT)$ using (2.3.1), (2.3.3), (2.3.6).
14 end
15 $S \leftarrow QSQ^*$
16 $C \leftarrow QCQ^*$

Cost: $(95/3+(\pi+2s)/3)n^3$ flops where π denotes the number of matrix multiplications needed to form both approximants. The corresponding transformation-free algorithm costs $(14/3 + 2(\pi + 2s))n^3$ flops. Comparing these two costs we see that it is cheaper to use the Schur decomposition if $\pi + 2s \geq 17$.

For comparison, the cost of calling Algorithms 2.4.2 and 2.5.2 separately—but using only one Schur decomposition—is $(31+(\pi_c+\pi_s+s_c+s_s+2)/3)n^3$ flops, or $(16/3+2(\pi_c+\pi_s+s_c+s_s))n^3$ flops if the transformation-free version is used, where the subscripts c and s denote the values obtained from the cosine and sine algorithms, respectively. To illustrate the benefit to the overall cost gained by computing the matrix cosine and

sine simultaneously consider the matrix `A = gallery('frank',1000)`: we obtain the values $\pi_c = 9$, $\pi_s = 5$, $s_c = 14$, $s_s = 11$ when using Algorithms 2.4.2 and 2.5.2, and $\pi = 14$, $s = 14$ when using Algorithm 2.6.2. Therefore the cost of computing the matrix cosine and sine separately is $(220/3)n^3$ and $(316/3)n^3$ flops (with and without a Schur decomposition, respectively) as opposed to $(137/3)n^3$ and $(266/3)n^3$ flops when computing both functions simultaneously. For the Schur decomposition algorithm this is a computational saving of around 38 percent.

2.7 Numerical experiments

All our experiments are performed in MATLAB 2013b. The test matrices are mainly 15×15 matrices adapted from the Matrix Computation Toolbox [50], the MATLAB `gallery` function, and the matrix function literature. We select 76 of these matrices for which none of the algorithms overflow and multiply them by some uniform random scalar between 1 and 60 to ensure that some scaling will occur in the algorithms. Similar test matrices were used in recent work on the matrix cosine [46], [57, Chap. 12], [95].

Our numerical experiments compare the normwise forward and backward errors of the competing methods. Recall from section 1.4 that if Y denotes the computed value of $f(A)$ then the forward error is

$$\frac{\|Y - f(A)\|_1}{\|f(A)\|_1} \tag{2.7.1}$$

and the backward error is

$$\eta(Y) = \min \{ \, \|E\|_1/\|A\|_1 : Y = f(A+E) \, \}. \tag{2.7.2}$$

The backward error is difficult to compute exactly so we make a linearized approximation of it, which is justified as we are interested in cases where $\eta \ll 1$. Our approximation is based on the first-order expansion

$$Y = f(A+E) \approx f(A) + L_f(A,E), \tag{2.7.3}$$

where $L_f(A,E)$ is the Fréchet derivative of f at A in the direction E, which is equivalent to

$$K_f(A)\,\mathrm{vec}(E) \approx \mathrm{vec}(Y - f(A)), \tag{2.7.4}$$

where $K_f(A) \in \mathbb{C}^{n^2 \times n^2}$ is the Kronecker form and $\mathrm{vec}(E)$ stacks the columns of E vertically from first to last. Both the Fréchet derivative and Kronecker form were defined in section 1.2. We approximate the backward error $\eta(Y)$ by the minimum 2-norm solution of (2.7.4), where the Kronecker matrix (obtained using [57, Alg. 3.17]) and $f(A)$ are computed, and the system solved, in 250 digit arithmetic using the Symbolic Math Toolbox. A similar idea is used effectively by Deadman and Higham in [29, Sec. 5] to compute linearized backward errors of functional identities.

Since we showed in Lemma 2.2.1 that all our algorithms are backward stable in exact arithmetic we expect the relative error (2.7.1) to be bounded by a modest multiple of $\mathrm{cond}(f, A)u$, where the condition number $\mathrm{cond}(f, A)$ is defined in section 1.3.

Accurate values of $\cos A$ and $\sin A$ for use in (2.7.1) and (2.7.4) are generated in 250 digit arithmetic using the Symbolic Math Toolbox by adding a random perturbation of norm 10^{-125} to A then diagonalizing it and taking the cosine (or sine) of the eigenvalues; the perturbation ensures the eigenvalues are distinct so that the diagonalization is always possible. This idea is based upon [27] and has been used successfully in [61], we will also use this idea in chapters 3 and 5. The condition number of f is estimated using the code `funm_condest1` from the Matrix Function Toolbox [51], [57, Alg. 3.20].

We test our new algorithms against the existing alternatives:

- `cosm` for $\cos A$ from [51], which implements an algorithm of Hargreaves and Higham [46, Alg. 3.1], [57, Alg. 12.6].
- `cosmsinm` for $\cos A$ and $\sin A$ from [51], which implements an algorithm of Hargreaves and Higham [46, Alg. 5.1], [57, Alg. 12.8].
- `costay`[2] for $\cos A$ from Sastre et al. [95].

The first two algorithms use variable degree Padé approximants, while `costay` uses a variable degree truncated Taylor series. None of these algorithms performs a Schur decomposition of the input matrix or recomputes the diagonal blocks (as described in section 2.3), and all are based on forward error analysis. We will perform comparisons both with and without the initial Schur decomposition within our algorithms.

Our first three experiments are designed to test our new algorithms against the aforementioned alternatives on full matrices. We denote our new algorithms by

[2]M-file retrieved from `http://personales.upv.es/~jorsasma/costay.m` on May 2, 2013.

- `cosm_new`: Algorithm 2.4.2,
- `sinm_new`: Algorithm 2.5.2,
- `cosmsinm_new`: Algorithm 2.6.2.

In each case we plot our results in a 4×2 grid. Each plot in the $(1,1)$ position shows the forward errors (2.7.1), with the new algorithm in transformation-free form, along with an estimate of $\mathrm{cond}(f, A)u$. The $(1,2)$ plot presents the same relative errors as a performance profile [36], [49, Sec. 22.4], to which we apply the strategy from [35] to avoid tiny relative errors skewing the results. The $(2,1)$ plot shows the backward errors (2.7.2) and the $(2,2)$ plot is a performance profile for the backward error results. The remaining four plots show the same results when we allow our new algorithms to compute an initial Schur decomposition, allowing recomputation of the diagonal blocks (section 2.3).

Our next experiment tests each algorithm on matrices that are already triangular: we precompute the Schur decomposition of each test matrix before sending the triangular factor to the algorithms. This shows how well the algorithms exploit triangularity. In some applications the original matrices are triangular; see, for example, [107] in the case of the matrix exponential.

The final experiment compares the accuracy of `cosm_new` and `costay` on a matrix resulting from the semidiscretization of a nonlinear wave problem [40, Prob. 4]. We compare the two algorithms for a range of discretization points, where the matrix becomes more nonnormal (and hence the problem more difficult) as the resolution increases.

2.7.1 Matrix cosine

Figure 2.7.1 shows the results for the matrix cosine. For the transformation-free case of `cosm_new` we see from the $(1,2)$ plot that `costay` was often the most accurate algorithm, as shown by the $\alpha = 1$ ordinate, but was less reliable than `cosm_new`, as shown by the relative position of the curves for $\alpha \geq 2$. Indeed not visible in the $(1,1)$ plot is the relative error returned by `costay` for test matrix 4, which was $\gg 1$, as opposed to 9e-8 and 3e-2 for `cosm_new` and `cosm`, respectively. The condition number of this problem was 4.8e9, so we would expect a forward error of order 10^{-7} from a

forward stable algorithm. From the $(2,1)$ and $(2,2)$ plots we see that none of the algorithms is always backward stable and that `cosm_new` shows a small advantage over `costay`. The largest backward errors were $\gg 1$, 2.1e1, and 1.4e-2 for `costay`, `cosm`, and `cosm_new`, respectively.

For the tests in which a Schur decomposition is used (the lower four plots) we see an improvement in backward stability of `cosm_new` at the expense of some slight deterioration in the forward error.

The $(1,1)$ and $(1,2)$ plots of Figure 2.7.4 show the cost of each algorithm in multiples of n^3 flops using the transformation-free and Schur versions, respectively. We see that the transformation-free version of `cosm_new` is marginally more expensive than `costay` and `cosm` in most cases but is significantly cheaper than the latter on occasion. The Schur version has a relatively stable cost of around $32n^3$ flops, due to the fixed overhead of the Schur decomposition, which could be advantageous for highly oscillatory differential equations [106], where the matrices have large eigenvalues and hence a heavy scaling may be needed, requiring a large number of matrix multiplications. The precise criteria under which the Schur algorithm is cheaper than the transformation-free version was explained at the end of section 2.4.

2.7.2 Matrix sine

For our second experiment, since there are no other algorithms dedicated to computing the matrix sine, we compare `sinm_new` against the use of `costay` to compute $\sin A = \cos(A - \pi I/2)$. Our comparison of these two algorithms is shown in Figure 2.7.2. For the transformation-free algorithm we see that `sinm_new` has significantly better forward error and backward error performance. The use of the Schur decomposition improves the forward stability of `sinm_new`: the forward errors in the $(3,1)$ plot are always within a factor 15 of the condition number times u as opposed to 186 for the $(1,1)$ plot.

The cost of each algorithm is given in the $(2,1)$ and $(2,2)$ plots of Figure 2.7.4 for the transformation-free and Schur versions, respectively. In each case `sinm_new` is generally more expensive than `costay`, and using the Schur decomposition gives a fairly stable cost of around $32n^3$ flops.

2.7.3 Matrix cosine and sine

Our third experiment compares `cosmsinm_new` to `cosmsinm`. For each test matrix we show the largest of the two errors in evaluating the matrix cosine and sine: if a particular test matrix results in backward or forward errors e_c and e_s for the cosine and sine respectively we plot $\max(e_c, e_s)$. In the plots the quantity $\mathrm{cond}(f, A)u$ denotes the larger of $\mathrm{cond}(\cos, A)u$ and $\mathrm{cond}(\sin, A)u$. It is clear from Figure 2.7.3 that the new algorithm has significantly better forward and backward error performance than `cosmsinm`. Plots of the individual errors for the sine and cosine have similar forms. The largest relative errors returned by `cosmsinm` were 2.9e2 and 1e3 for the matrix cosine and sine, respectively, but only 1.5e-7 and 5e-7 for `cosmsinm_new`.

Plots showing the cost of each algorithm, both avoiding and utilizing an initial Schur decomposition, are given in the $(3, 1)$ and $(3, 2)$ positions of Figure 2.7.4, respectively. We see that the transformation-free version of `cosmsinm_new` is slightly more expensive than `cosmsinm`, but allowing an initial Schur decomposition makes our new algorithm cheaper in some of the test cases.

2.7.4 Triangular matrices

Figures 2.7.5–2.7.7 show the results of applying the algorithms to matrices that are already (quasi)-triangular, obtained by taking the (real) Schur form of each matrix before passing it to the competing algorithms. The new algorithms are greatly superior to the existing ones in terms of both forward and backward error, often achieving values of order u. By comparison with Figures 2.7.1–2.7.3 it is clear that the Schur decomposition is a significant source of error in our algorithms.

The cost of the competing algorithms for triangular matrices is shown in Figure 2.7.8. In the majority of cases our new algorithms are more efficient than the alternatives.

2.7.5 Wave discretization problem

Our final experiment compares the forward errors of `cosm_new` and `costay` when computing the cosine of a matrix arising from the semidiscretization of the wave equation [40, Prob. 4]

$$\frac{\partial^2 u}{\partial t^2} - a(x)\frac{\partial^2 u}{\partial x^2} + \alpha u = f(t,x,u), \tag{2.7.5}$$

where $x \in (0,1)$, $t > 0$, and

$$f(t,x,u) = u^5 - a(x)^2 u^3 + \frac{a(x)^5}{4}\sin^2(20t)\cos(10t), \quad a(x) = 4x(1-x). \tag{2.7.6}$$

With Dirichlet boundary conditions the solution is $u(x,t) = a(x)\cos(10t)$. If we perform a semidiscretization in the spatial variable with mesh size $1/n$ we obtain a system of the form (2.1.3) with parametrized matrix

$$A = n^2 \begin{bmatrix} 2a(x_1) & -a(x_1) & & & \\ -a(x_2) & 2a(x_2) & -a(x_2) & & \\ & \ddots & \ddots & \ddots & \\ & & -a(x_{n-2}) & 2a(x_{n-2}) & -a(x_{n-2}) \\ & & & -a(x_{n-1}) & 2a(x_{n-1}) \end{bmatrix} + \alpha I. \tag{2.7.7}$$

This matrix becomes increasingly nonnormal as n grows. In the Frobenius norm the departure from normality is measured as $\|T - \mathrm{diag}(T)\|_F$, where T is the Schur factor of A, and for $n = 10, 50, 100, 200$ the departure from normality was approximately 52, 770, 2200, and 6400, respectively.

In Figure 2.7.9 we show the forward errors of the two algorithms when computing $\cos A$ for a range of n as α varies between 0 and 10. The results shown are for the transformation-free version of `cosm_new`; similar results were obtained using the Schur decomposition. For $n = 10$, both algorithms behave in a forward stable manner, with `costay` generally more accurate. As n increases `cosm_new` becomes significantly more accurate than `costay`, the latter showing signs of forward instability and having errors varying much less smoothly with α.

2.8 Evaluating rational approximants with the Paterson–Stockmeyer method

It was mentioned in sections 2.4–2.6 that we can efficiently compute the rational approximants c_m and s_m using the Paterson–Stockmeyer scheme [88]. Given a matrix polynomial $p_m(X) = \sum_{k=0}^{m} a_k X^k$ we can write it in the form

$$p_m(X) = \sum_{k=0}^{\ell} g_k(X)(X^\tau)^k, \tag{2.8.1}$$

where $\tau = 1\colon m$ is some integer, $\ell = \lfloor m/\tau \rfloor$, and

$$g_k(X) = \begin{cases} a_{\tau k+\tau-1}X^{\tau-1} + \cdots + a_{\tau k}I, & k = 0\colon \ell-1, \\ a_m X^{m-\tau\ell} + \cdots + a_{\tau\ell}I, & k = \ell. \end{cases} \tag{2.8.2}$$

From this it can be shown that the cost (in matrix multiplications) of evaluating the polynomial, using Horner's method for (2.8.1), is

$$\tau + \ell - 1 - \phi(\tau, m), \tag{2.8.3}$$

where $\phi(\tau, m) = 1$ if τ divides m and is equal to 0 otherwise [57, pp. 73–74]. We can find the optimal τ by minimizing the cost function (2.8.3) over $\tau = 1\colon m$.

To apply this to the matrix cosine, we see that the rational approximations obtained in section 2.2.2 are of the form

$$c_m(x) = \frac{\sum_{k=0}^{m} a_k x^{2k}}{\sum_{k=0}^{m} b_k x^{2k}}, \tag{2.8.4}$$

so the numerator and denominator of $c_m(x)$ are degree m polynomials in $B = X^2$. If we evaluate the denominator using the procedure above, reusing the same value of τ for the numerator then the overall cost is

$$2\left\lfloor \frac{m}{\tau} \right\rfloor + \tau - 2\phi(\tau, m). \tag{2.8.5}$$

The values $\pi(c_m(X))$ in Table 2.4.1 were generated by minimizing (2.8.5) over $\tau = 1 : m$ for each m.

To show that this Paterson–Stockmeyer scheme is more efficient than computing the necessary powers of X explicitly we show the number of matrix multiplications needed to compute $c_m(X)$ for some values of m between 1 and 21 using our scheme and explicit powers in Table 2.8.1. A bound on the accuracy of these two evaluation schemes (the same bound applies to both) can be found in [57, Thm. 4.5].

Table 2.8.1: The number of matrix multiplications required to form $c_m(X)$ by explicit computation of the powers and by the Paterson–Stockmeyer scheme, along with a parameter τ that attains this cost for the latter.

m	1	2	3	4	6	8	10	12	15	18	21
Explicit powers	1	2	3	4	6	8	10	12	15	18	21
Paterson–Stockmeyer	1	2	3	4	5	6	7	8	9	10	11
τ	1	1	1	1	3	4	5	6	5	6	7

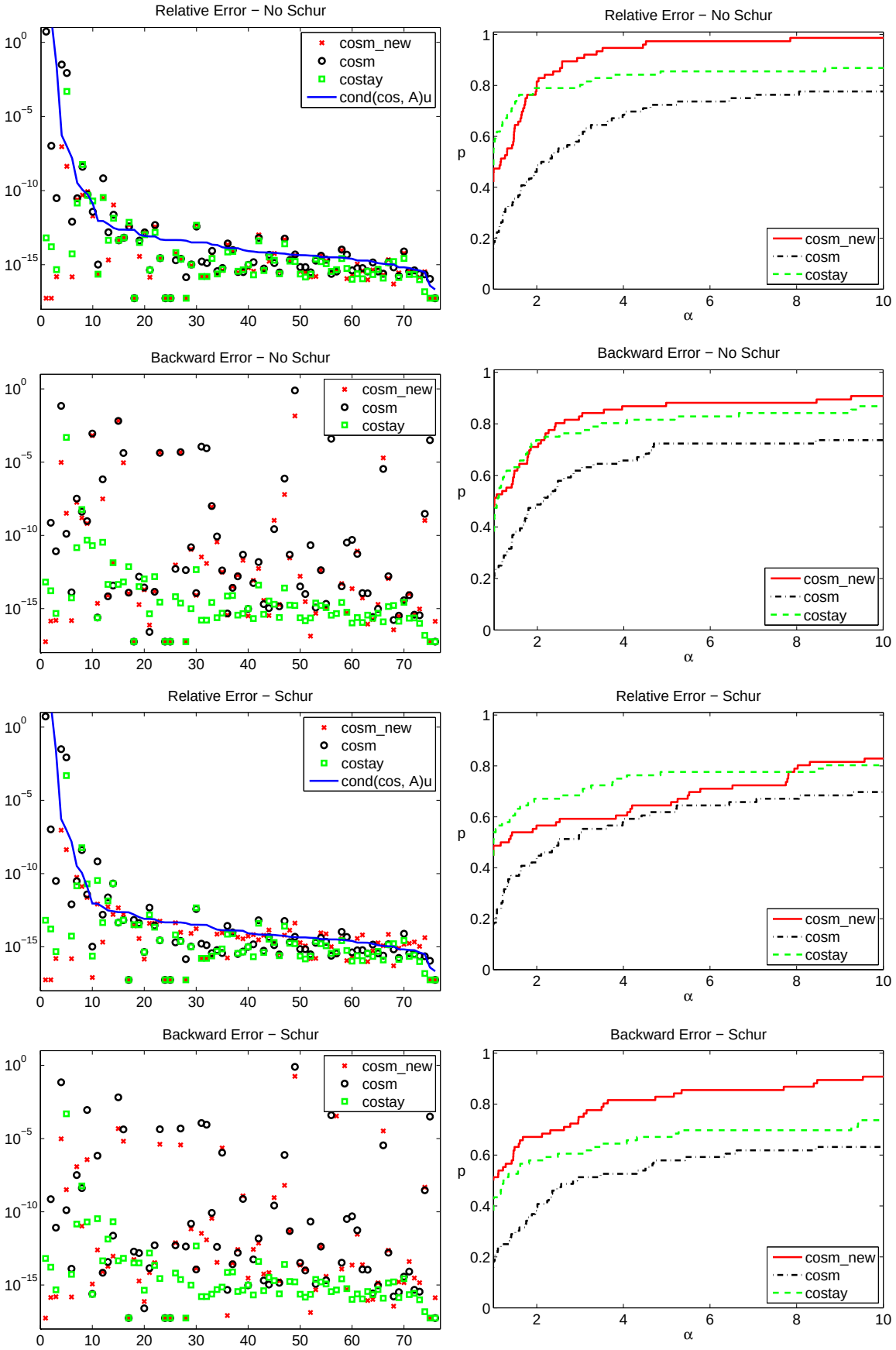

Figure 2.7.1: The forward and backward errors of competing algorithms for the matrix cosine, for full matrices. The first four plots are for the transformation-free version of Algorithm 2.4.2, whereas for the remaining four plots an initial Schur decomposition is used. The results are ordered by decreasing condition number.

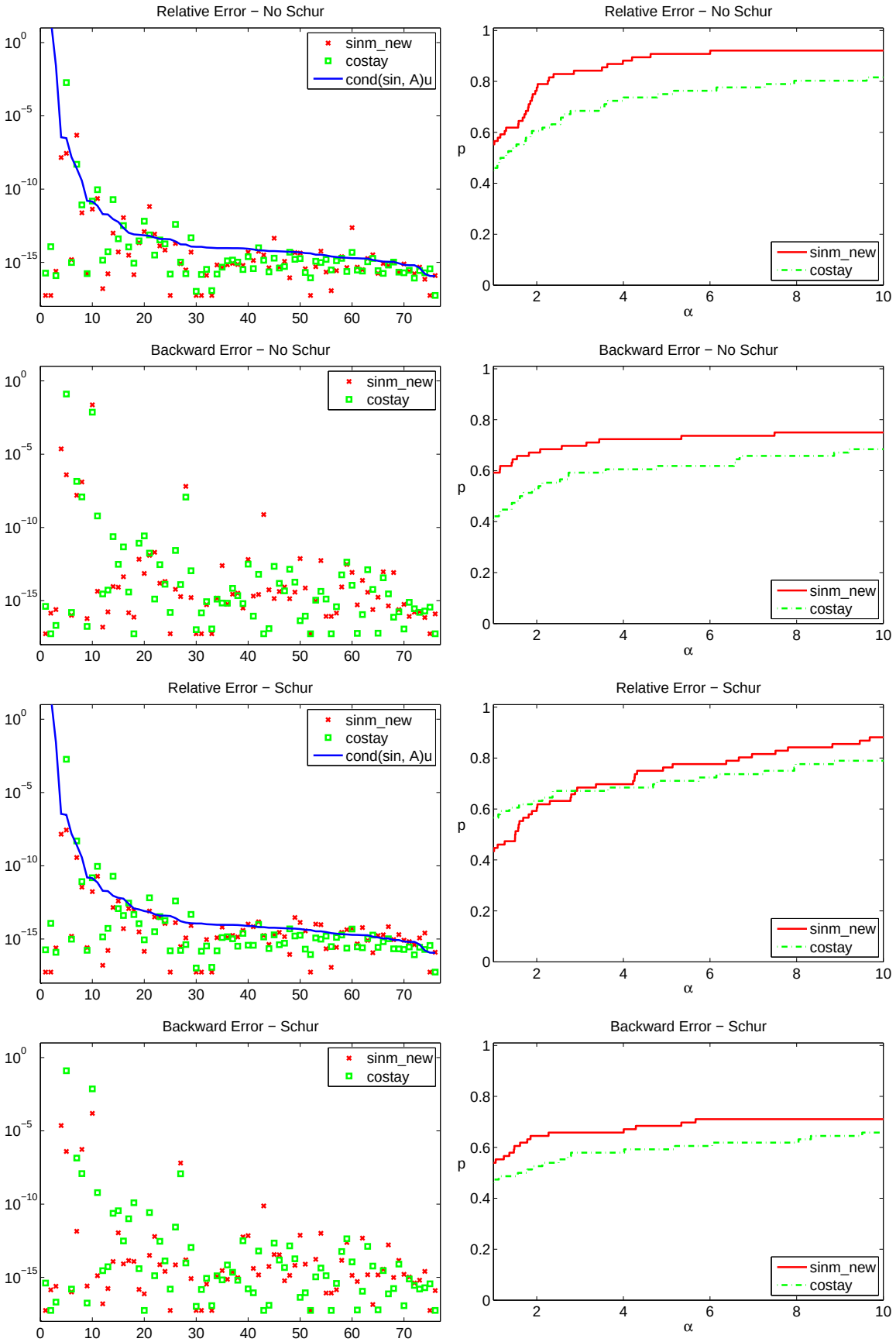

Figure 2.7.2: The forward and backward errors of competing algorithms for the matrix sine, for full matrices. The first four plots are for the transformation-free version of Algorithm 2.5.2, whereas for the remaining four plots an initial Schur decomposition is used. The results are ordered by decreasing condition number.

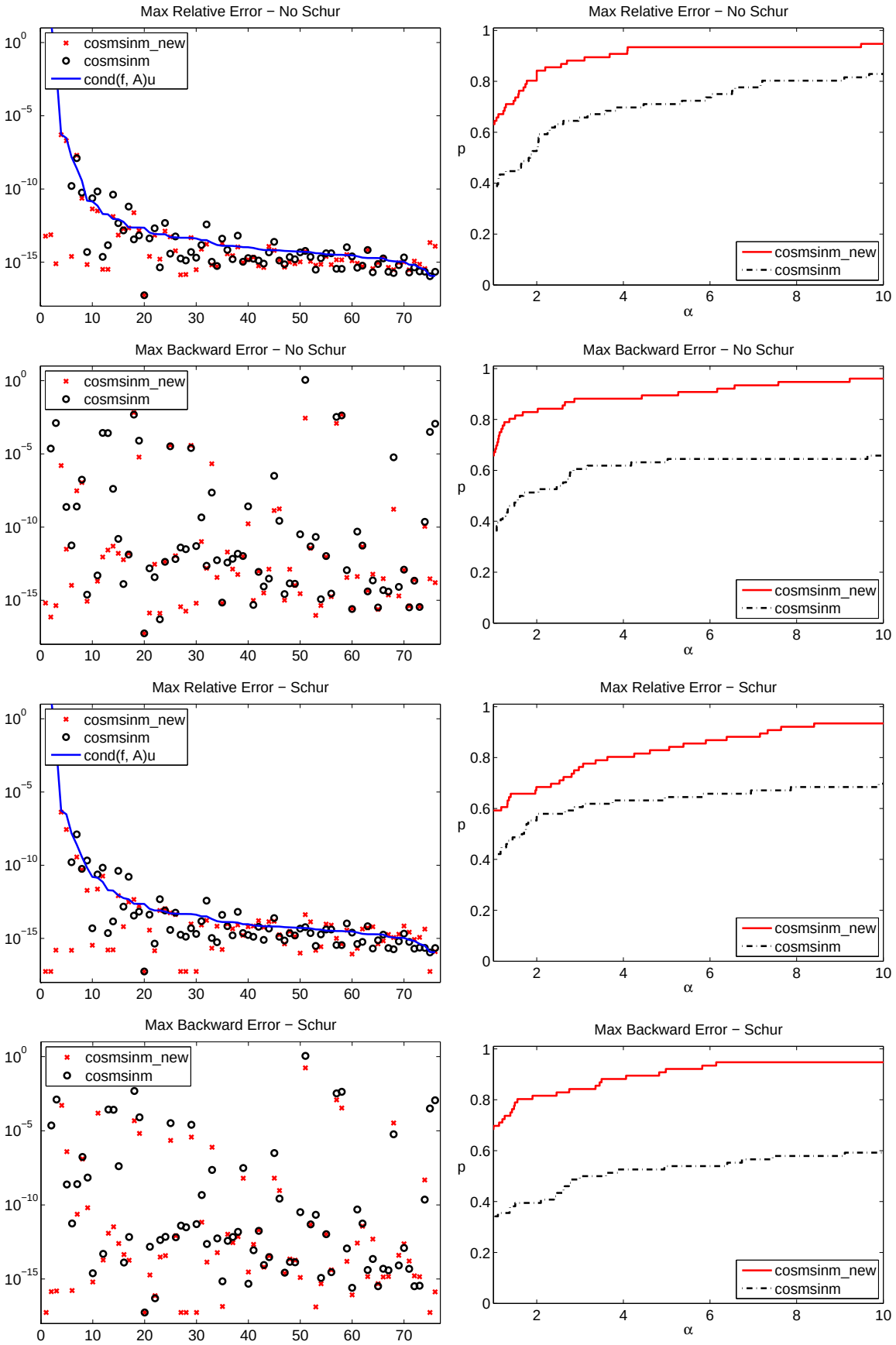

Figure 2.7.3: The maximum forward and backward errors of competing algorithms for the matrix cosine and sine, for full matrices. The first four plots are for the transformation-free version of Algorithm 2.6.2, while an initial Schur decomposition is used for the lower four plots. The results are ordered by decreasing condition number.

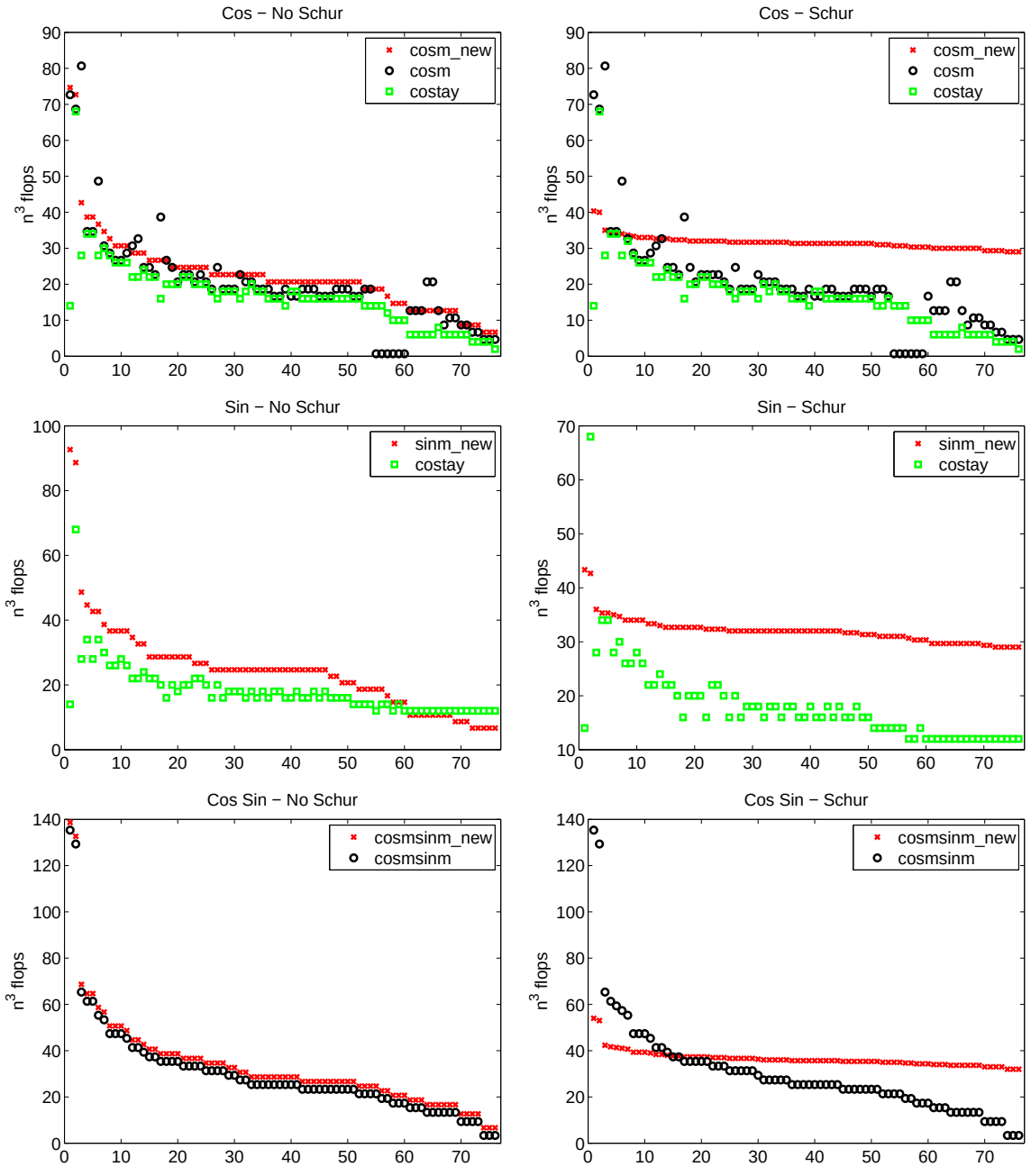

Figure 2.7.4: Cost plots for the experiments in sections 2.7.1, 2.7.2, and 2.7.3. The first row shows the cost of computing the matrix cosine whilst the second and third rows show the cost of computing the matrix sine and both functions together. The left column corresponds to the transformation-free versions of our new algorithms, whilst the right corresponds to an initial Schur decomposition. All plots are ordered by decreasing cost of our new algorithms.

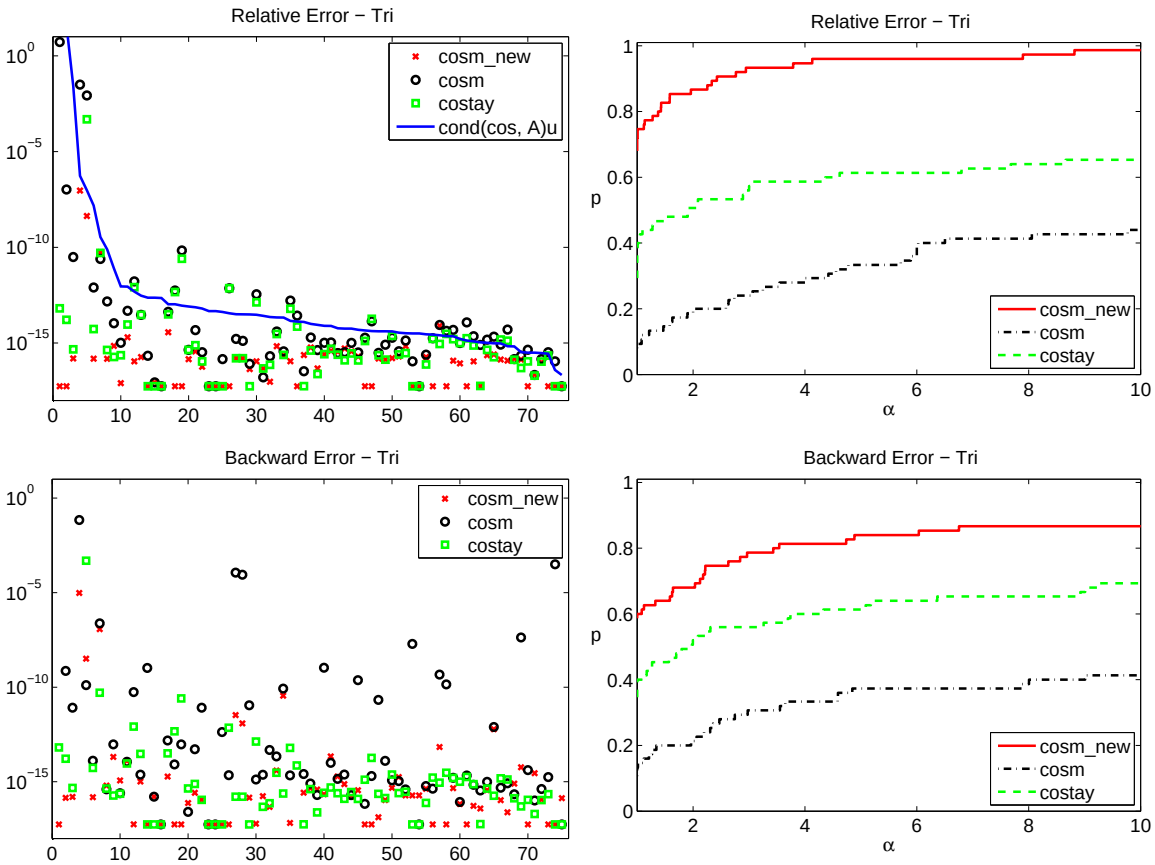

Figure 2.7.5: Forward and backward errors of competing algorithms for the matrix cosine for triangular matrices. The results are ordered by decreasing condition number.

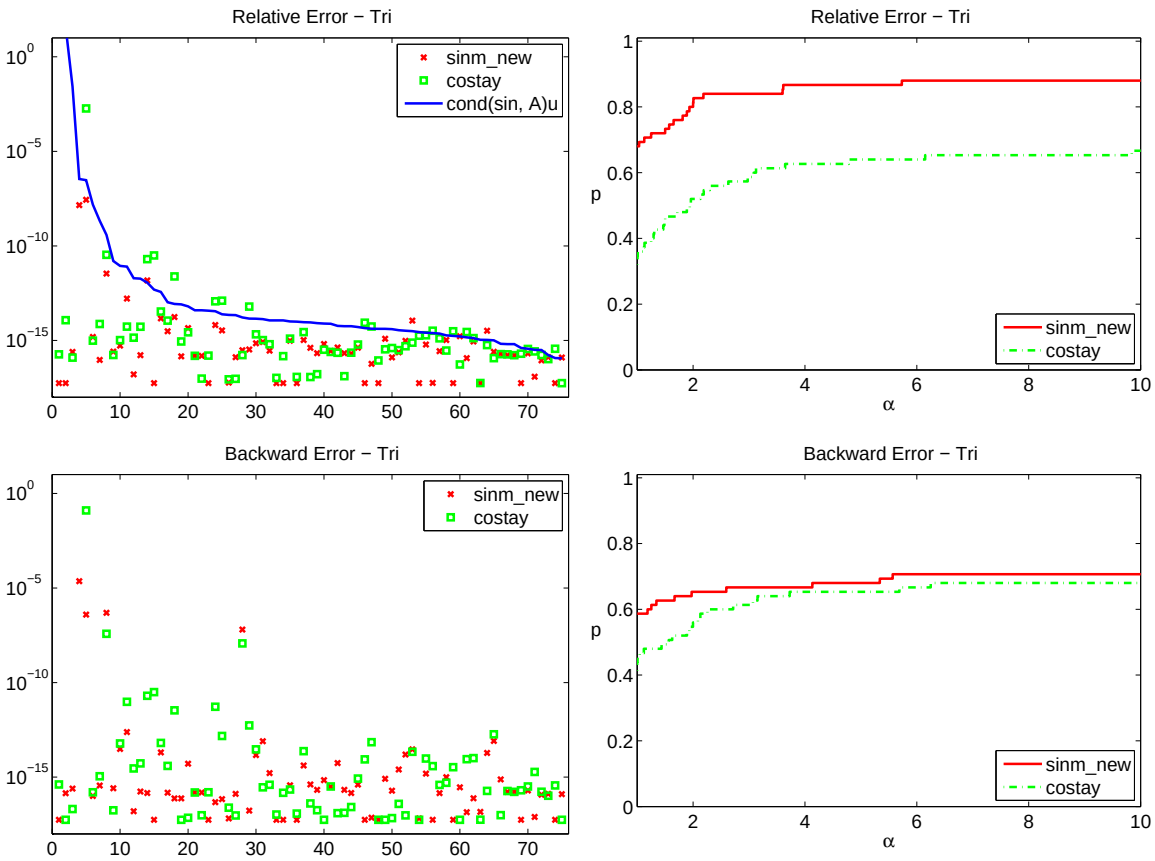

Figure 2.7.6: The forward and backward errors of competing algorithms for the matrix sine for triangular matrices. The results are ordered by decreasing condition number.

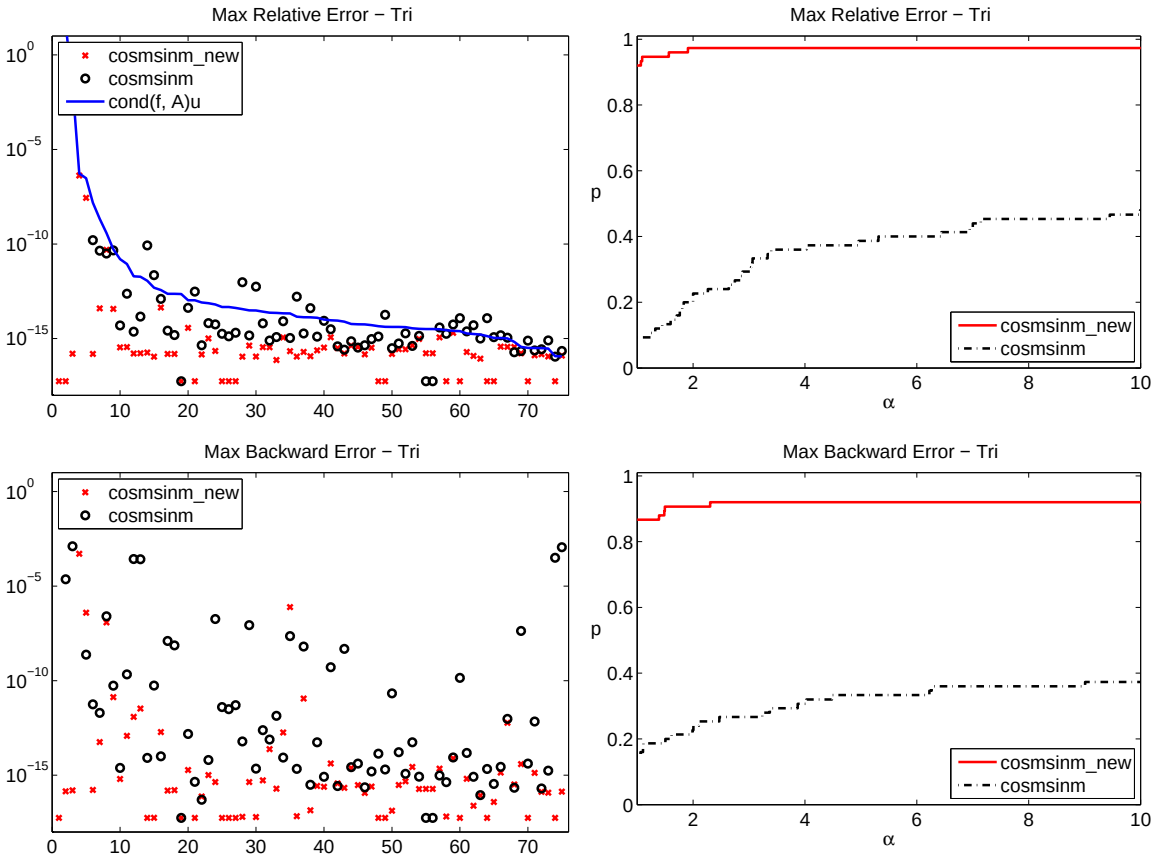

Figure 2.7.7: The maximum forward and backward errors of competing algorithms for the matrix cosine and sine, for triangular matrices. The results are ordered by decreasing condition number.

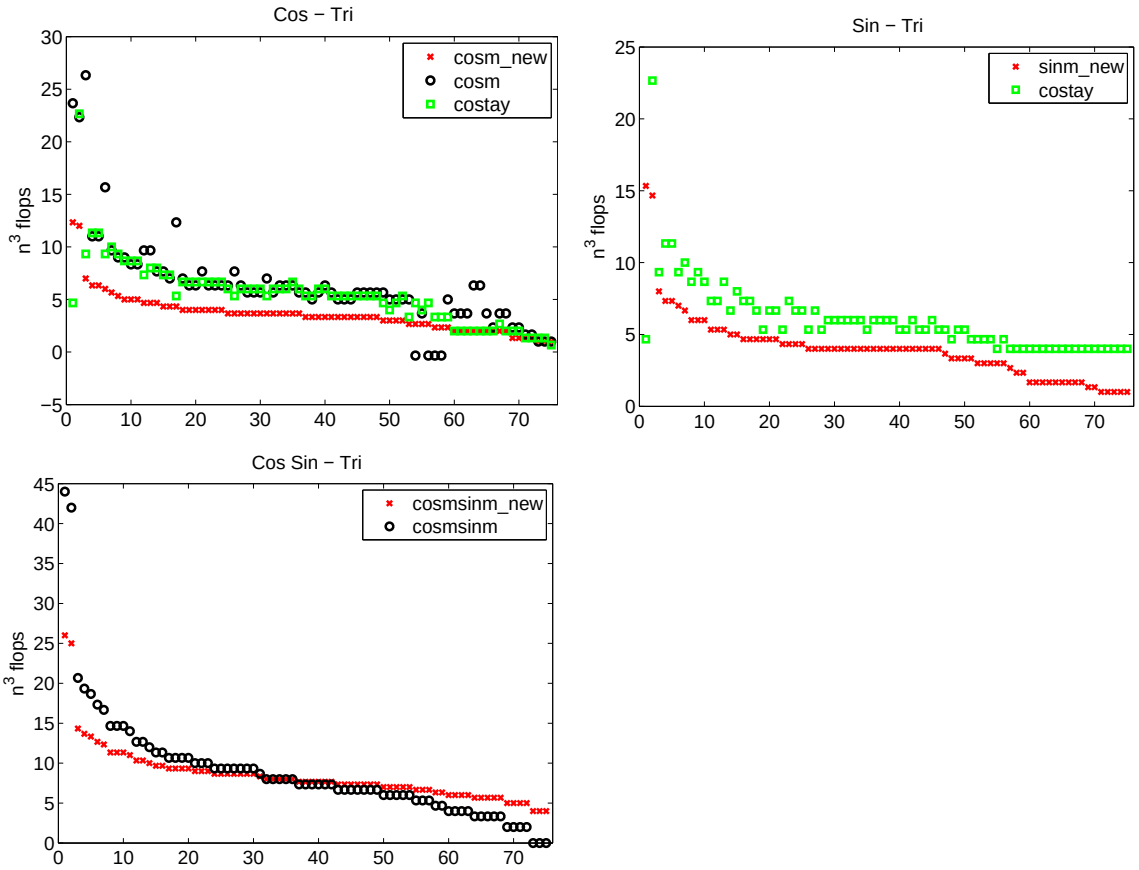

Figure 2.7.8: Cost plots for all the algorithms run on triangular matrices. All plots are ordered by decreasing cost of our new algorithms.

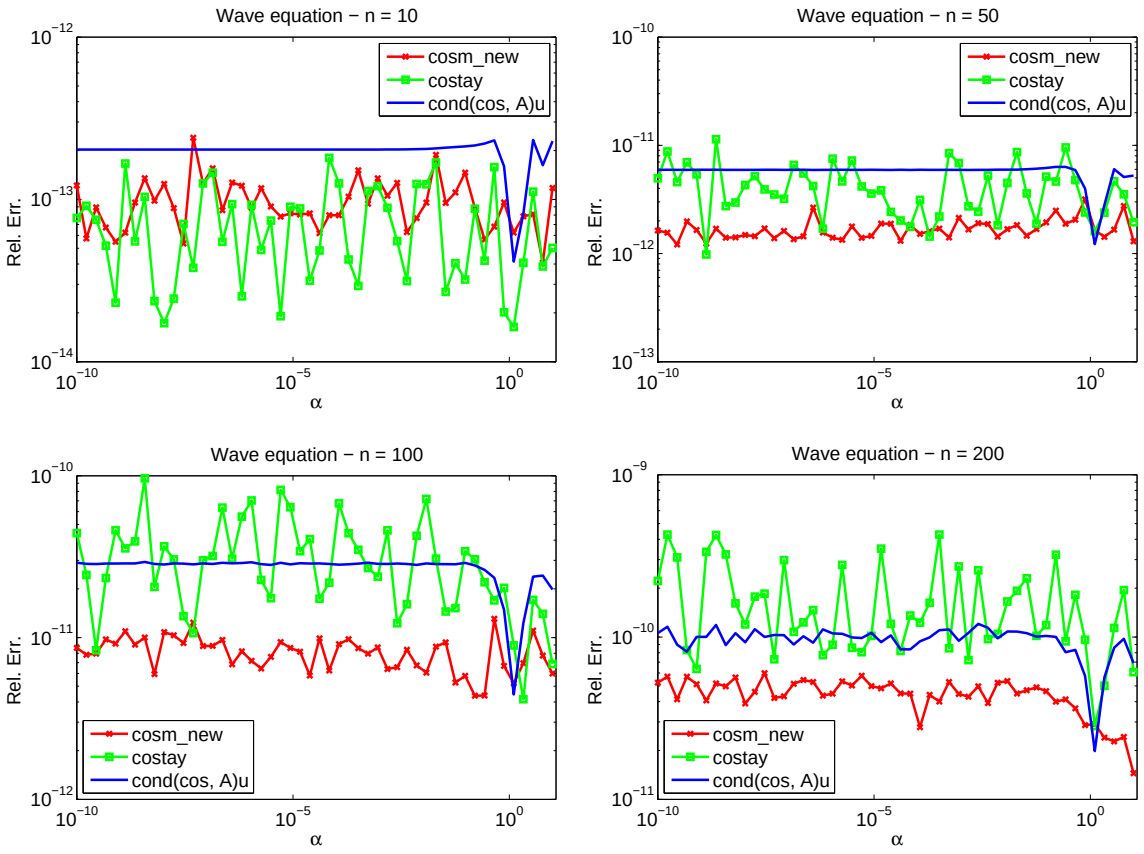

Figure 2.7.9: Forward errors of `cosm_new` and `costay` for the matrix (2.7.7) that arises from the semidiscretization of a nonlinear wave equation with parameter α. The matrix becomes increasingly nonnormal as n increases.

Chapter 3

Algorithms for the Matrix Logarithm, its Fréchet Derivative, and Condition Number

3.1 Introduction

A logarithm of $A \in \mathbb{C}^{n\times n}$ is a matrix X such that $e^X = A$. When A has no eigenvalues on $\mathbb{R}^-$, the closed negative real line, there is a unique logarithm X whose eigenvalues lie in the strip $\{\, z : -\pi < \mathrm{Im}(z) < \pi \,\}$ [57, Thm. 1.31]. This is the principal logarithm denoted by $\log(A)$. Under the same assumptions on A, there is a unique matrix X satisfying $X^2 = A$ that has all its eigenvalues in the open right half-plane [57, Thm. 1.29]. This is the principal square root of A, denoted by $A^{1/2}$.

An excellent method for evaluating the matrix logarithm is the inverse scaling and squaring method proposed by Kenney and Laub [73], which uses the relationship $\log(A) = 2^s \log(A^{1/2^s})$ together with a Padé approximant of $\log(A^{1/2^s})$. This method has been developed by several authors, including Dieci, Morini, and Papini [33], Cardoso and Silva Leite [21], Cheng, Higham, Kenney, and Laub [23], and Higham [57, Sec. 11.5]. Most recently, Al-Mohy and Higham [6] developed backward error analysis for the method and obtained a Schur decomposition-based algorithm that is faster and more accurate than previous inverse scaling and squaring algorithms.

This chapter has three main aims: to develop a version of the algorithm of Al-Mohy and Higham [6] that works with the real Schur decomposition when A is real,

to extend both versions of the algorithm to compute the Fréchet derivative, and to develop an integrated algorithm that first computes $\log(A)$ and then computes one or more Fréchet derivatives with appropriate reuse of information—in particular, to allow efficient estimation of the condition number of the logarithm.

We recall that the Fréchet derivative, Kronecker form, and condition number were defined in sections 1.2 and 1.3. Throughout this chapter, unless otherwise specified, $\|\cdot\|$ can refer to any subordinate matrix norm and $\mathrm{cond}(f, A)$ represents the relative condition number (1.3.2).

The rest of this chapter is organized as follows. In section 3.2 we describe the outline of our algorithm for computing the Fréchet derivative of the matrix logarithm. We derive backward error bounds in section 3.3 and use them to choose the algorithmic parameters. Estimation of the condition number is discussed in section 3.4 and detailed algorithms are then given in section 3.5 for the complex case and section 3.6 for the real case. We compare our algorithms to current alternatives theoretically in section 3.7 before performing numerical experiments in section 3.8.

3.2 Basic algorithm

We begin by deriving the basic structure of an algorithm for approximating the Fréchet derivative of the logarithm. The idea is to Fréchet-differentiate the inverse scaling and squaring approximation $\log(A) \approx 2^s r_m(A^{1/2^s} - I)$, where $r_m(x)$ is the $[m/m]$ Padé approximant to $\log(1 + x)$, following the computational framework suggested in [3], [58, Sec. 7.4].

We will need two tools: the chain rule for Fréchet derivatives, $L_{f\circ g}(A, E) = L_f(g(A), L_g(A, E))$ [57, Thm. 3.4] and the inverse function relation for Fréchet derivatives $L_f(X, L_{f^{-1}}(f(X), E)) = E$ [57, Thm. 3.5].

Applying the inverse function relation to $f(x) = x^2$, for which $L_{x^2}(A, E) = AE + EA$, we see that $L = L_{x^{1/2}}(A, E)$ satisfies $A^{1/2}L + LA^{1/2} = E$. Furthermore using the chain rule on the identity $\log(A) = 2\log(A^{1/2})$ gives $L_{\log}(A, E) = 2L_{\log}(A^{1/2}, E_1)$ where $E_1 = L_{x^{1/2}}(A, E)$. Repeated use of these two results yields $L_{\log}(A, E) = 2^s L_{\log}(A^{1/2^s}, E_s)$ where $E_0 = E$ and

$$A^{1/2^i}E_i + E_i A^{1/2^i} = E_{i-1}, \quad i = 1\colon s. \tag{3.2.1}$$

For suitably chosen s and m we then approximate $L_{\log}(A,E) \approx 2^s L_{r_m}(A^{1/2^s} - I, E_s)$.

The following outline algorithm will be refined in the subsequent sections.

Algorithm 3.2.1. *Given $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$, $E \in \mathbb{C}^{n\times n}$, and nonnegative integers s and m, this algorithm approximates $\log(A)$ and the Fréchet derivative $L_{\log}(A,E)$.*

1 $E_0 = E$
2 for $i = 1\colon s$
3 Compute $A^{1/2^i}$.
4 Solve the Sylvester equation $A^{1/2^i}E_i + E_i A^{1/2^i} = E_{i-1}$ for E_i.
5 end
6 $\log(A) \approx 2^s r_m(A^{1/2^s} - I)$
7 $L_{\log}(A,E) \approx 2^s L_{r_m}(A^{1/2^s} - I, E_s)$

Higham [54] showed that among the various alternative representations of r_m, the partial fraction form given by

$$r_m(X) = \sum_{j=1}^{m} \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} X, \tag{3.2.2}$$

where $\alpha_j^{(m)}, \beta_j^{(m)} \in (0,1)$ are the weights and nodes of the m-point Gauss–Legendre quadrature rule on $[0,1]$ respectively, provides the best balance between efficiency and numerical stability. To calculate the Fréchet derivative L_{r_m} we differentiate (3.2.2) using the product rule. Recalling that $L_{x^{-1}}(X,E) = -X^{-1}EX^{-1}$ we obtain

$$\begin{aligned} L_{r_m}(X,E) &= \sum_{j=1}^{m} \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E - \alpha_j^{(m)} \beta_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E (I + \beta_j^{(m)} X)^{-1} X \\ &= \sum_{j=1}^{m} \left(\alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E \right) \left(I - \beta_j^{(m)} (I + \beta_j^{(m)} X)^{-1} X \right) \\ &= \sum_{j=1}^{m} \alpha_j^{(m)} (I + \beta_j^{(m)} X)^{-1} E (I + \beta_j^{(m)} X)^{-1}. \end{aligned} \tag{3.2.3}$$

3.3 Backward error analysis

We now develop a backward error result for the approximation errors in lines 6 and 7 of Algorithm 3.2.1. Define the function $h_{2m+1}\colon \mathbb{C}^{n\times n} \mapsto \mathbb{C}^{n\times n}$ by $h_{2m+1}(X) =$

Table 3.3.1: Maximal values θ_m of $\alpha_p(X)$ such that the bound in (3.3.2) for $\|\Delta X\|/\|X\|$ does not exceed u.

m	1	2	3	4	5	6	7	8
θ_m	1.59e-5	2.31e-3	1.94e-2	6.21e-2	1.28e-1	2.06e-1	2.88e-1	3.67e-1

m	9	10	11	12	13	14	15	16
θ_m	4.39e-1	5.03e-1	5.60e-1	6.09e-1	6.52e-1	6.89e-1	7.21e-1	7.49e-1

$e^{r_m(X)} - X - I$, which has the power series expansion [6]

$$h_{2m+1}(X) = \sum_{k=2m+1}^{\infty} c_k X^k. \tag{3.3.1}$$

We need the following backward error bound from [6, Thm. 2.2] for the approximation of the logarithm. In the following ρ denotes the spectral radius.

Theorem 3.3.1. *If $X \in \mathbb{C}^{n\times n}$ satisfies $\rho(r_m(X)) < \pi$ then $r_m(X) = \log(I+X+\Delta X)$, where, for any $p \geq 1$ satisfying $2m+1 \geq p(p-1)$,*

$$\frac{\|\Delta X\|}{\|X\|} \leq \sum_{k=2m+1}^{\infty} |c_k| \alpha_p(X)^{k-1}, \tag{3.3.2}$$

for $\alpha_p(X) = \max(\|X^p\|^{1/p}, \|X^{p+1}\|^{1/(p+1)})$. Furthermore, $\Delta X = h_{2m+1}(X)$.

The $\alpha_p(X)$ values were first introduced and exploited in [4]. We recall that $\alpha_p(X) \leq \|X\|$ and that $\alpha_p(X) \ll \|X\|$ is possible for very nonnormal X, so that bounds based upon $\alpha_p(X)$ are potentially much sharper than bounds based solely on $\|X\|$.

Values of $\theta_m := \max\{\, t : \sum_{k=2m+1}^{\infty} |c_k| t^{k-1} \leq u \,\}$, where $u = 2^{-53} \approx 1.1 \times 10^{-16}$ is the unit roundoff for IEEE double precision arithmetic, were determined in [6] for $m = 1\colon 16$ and are shown in Table 3.3.1. It is shown in [6] that $\rho(X) < 0.91$ implies $\rho(r_m(X)) < \pi$. Since we will need $\alpha_p(X) \leq \theta_{16} = 0.749$ and as $\rho(X) \leq \alpha_p(X)$ for all p, the condition $\rho(r_m(X)) < \pi$ in Theorem 3.3.1 is not a practical restriction. Now we give a backward error result for the Fréchet derivative computed via Algorithm 3.2.1.

Theorem 3.3.2. *If $X \in \mathbb{C}^{n\times n}$ satisfies $\rho(r_m(X)) < \pi$ then*

$$L_{r_m}(X, E) = L_{\log}(I + X + \Delta X, E + \Delta E), \tag{3.3.3}$$

where $\Delta X = h_{2m+1}(X)$ and $\Delta E = L_{h_{2m+1}}(X, E)$.

Proof. From Theorem 3.3.1 we know that $r_m(X) = \log(I + X + \Delta X)$ with $\Delta X = h_{2m+1}(X)$. Using the chain rule we obtain

$$\begin{aligned} L_{r_m}(X, E) &= L_{\log}\big(I + X + h_{2m+1}(X), E + L_{h_{2m+1}}(X, E)\big) \\ &=: L_{\log}(I + X + \Delta X, E + \Delta E). \qquad \square \end{aligned}$$

Note that Theorems 3.3.1 and 3.3.2 show that $r_m(X) = \log(I + X + \Delta X)$ and $L_{r_m}(X, E) = L_{\log}(I + X + \Delta X, E + \Delta E)$ with the *same* ΔX, so we have a single backward error result encompassing both r_m and L_{r_m}. It remains for us to bound $\|\Delta E\|$, which can be done using the following lemma [57, Prob. 3.6], [73].

Lemma 3.3.3. *Suppose f has the power series expansion $f(x) = \sum_{k=1}^{\infty} a_k x^k$ with radius of convergence r. Then for $X, E \in \mathbb{C}^{n\times n}$ with $\|X\| < r$,*

$$L_f(X, E) = \sum_{k=1}^{\infty} a_k \sum_{j=1}^{k} X^{j-1} E X^{k-j}. \tag{3.3.4}$$

We would like to use Lemma 3.3.3 with $f = h_{2m+1}$ to obtain a bound similar to (3.3.2) in terms of $\alpha_p(X)$. Unfortunately this is impossible when X and E do not commute (which must be assumed, since we do not wish to restrict E). To see why, note that the sum in (3.3.4) contains terms $a_k X^{j-1} E X^{k-j}$, for $j = 1\colon k$. The next lemma implies that $\|X\|$, for example, is always a factor in the norm of one of these terms for some E, which means that a bound for $\|\Delta E\|$ in terms of $\alpha_p(X)$ cannot be obtained.

Lemma 3.3.4. *For $A, B, E \in \mathbb{C}^{n\times n}$, and some subordinate matrix norm $\|\cdot\|$, we have $\|AEB\| \le \|A\|\,\|E\|\,\|B\|$ and furthermore the bound is attained for a rank-1 matrix E.*

Proof. Only the attainability of the bound is in question. Using properties of the norm $\|\cdot\|_D$ dual to $\|\cdot\|$ [55, Sec. 6.1, Probs. 6.2, 6.3] it is straightforward to show that the bound is attained for $E = xy^*$, where $\|Ax\| = \|A\|\,\|x\|$ and $\|B^*y\|_D = \|B^*\|_D \|y\|_D$. $\square$

Instead of trying to bound $\|\Delta E\|$ in terms of $\alpha_p(X)$, we take norms in (3.3.4) with $f = h_{2m+1}$ (see (3.3.1)) to obtain

$$\frac{\|\Delta E\|}{\|E\|} \le \sum_{k=2m+1}^{\infty} k|c_k|\|X\|^{k-1}. \tag{3.3.5}$$

Table 3.3.2: Maximal values β_m of $\|X\|$ such that the bound in (3.3.5) for $\|\Delta E\|/\|E\|$ does not exceed u, and the values of μ_m.

m	1	2	3	4	5	6	7	8
β_m	2.11e-8	2.51e-4	5.93e-3	2.89e-2	7.39e-2	1.36e-1	2.08e-1	2.81e-1
μ_m	4.00e0	6.00e0	8.06e0	1.03e1	1.27e1	1.54e1	1.85e1	2.20e1

	12	13	14	15	16			
β_m	3.52e-1	4.17e-1	4.77e-1	5.30e-1	5.77e-1	6.18e-1	6.54e-1	6.86e-1
μ_m	2.59e1	3.02e1	3.49e1	4.00e1	4.56e1	5.15e1	5.79e1	6.48e1

Define $\beta_m = \max\{\theta : \sum_{k=2m+1}^{\infty} k|c_k|\theta^{k-1} \le u\}$, so that $\|X\| \le \beta_m$ implies that $\|\Delta E\|/\|E\| \le u$. Table 3.3.2 shows the values of β_m, calculated using the Symbolic Math Toolbox for MATLAB. In each case we have $\beta_m < \theta_m$, as is immediate from the definitions of β_m and θ_m.

It is possible to obtain unified bounds for $\|\Delta X\|$ and $\|\Delta E\|$ in terms of $\alpha_p(X)$ if we change the norm. For $\alpha_p(X) < 1$ there exists $\epsilon > 0$ and a matrix norm $\|\cdot\|_\epsilon$ such that

$$\|X\|_\epsilon \le \rho(X) + \epsilon \le \alpha_p(X) + \epsilon < 1.$$

Taking norms in (3.3.1) using $\|\cdot\|_\epsilon$ and taking $\|\cdot\| = \|\cdot\|_\epsilon$ in (3.3.5) we obtain

$$\frac{\|\Delta X\|_\epsilon}{\|X\|_\epsilon} \le \sum_{k=2m+1}^{\infty} |c_k|\big(\alpha_p(X)+\epsilon\big)^{k-1}, \quad \frac{\|\Delta E\|_\epsilon}{\|E\|_\epsilon} \le \sum_{k=2m+1}^{\infty} k|c_k|\big(\alpha_p(X)+\epsilon\big)^{k-1}.$$

Unfortunately the norm $\|\cdot\|_\epsilon$ is badly scaled if ϵ is small so these bounds are difficult to interpret in practice.

We will build our algorithm for computing the logarithm and its derivative on the condition $\alpha_p(X) \le \theta_m$ that ensures that $\|\Delta X\|/\|X\| \le u$. The bound (3.3.5) for $\|\Delta E\|/\|E\|$ is generally larger than u due to (a) $\|X\|$ exceeding $\alpha_p(X)$ by a factor that can be arbitrarily large and (b) the extra factor k in (3.3.5) compared with (3.3.2). The effect of (b) can be bounded as follows. Suppose we take $\|X\| \le \theta_m$, and define μ_m by

$$\mu_m = \frac{1}{u} \sum_{k=2m+1}^{\infty} k|c_k|\theta_m^{k-1}. \tag{3.3.6}$$

Then $\|X\| \le \theta_m$ implies that $\|\Delta E\|/\|E\| \le \mu_m u$. Table 3.3.2 gives the values of μ_m, which we see are of modest size and increase slowly with m. The algorithm of [6]

normally chooses a Padé approximant of degree $m = 6$ or 7, for which we have the reasonable bound $\|\Delta E\|/\|E\| \leq 18.5u$.

Since our main use of the Fréchet derivative is for condition number estimation, for which only order of magnitude estimates are required, it is reasonable to allow this more liberal bounding on the backward error ΔE. We will see in the numerical experiments of section 3.8 that in fact our algorithm gives very accurate estimates of the Fréchet derivative in practice.

3.4 Condition number estimation

We will estimate the 1-norm condition number of the matrix logarithm via the procedure outlined in section 1.3. This will involve computing the Fréchet derivative for a fixed A in multiple directions E, where the E matrices are not known prior to running the algorithm.

To maximize the computational efficiency of this process we will store certain matrices appearing during the computation of $\log(A)$ for reuse in the Fréchet derivative computations. This assumes the availability of sufficiently large storage on the machine running the algorithm; in modern computing environments there is usually ample storage.

From Algorithm 3.2.1 we see that it would be beneficial to store the square roots $A^{1/2^i}$ (or $T^{1/2^i}$ after the initial Schur decomposition) for $i = 1\colon s$ with which we solve a series of Sylvester equations.

In the next two sections we give algorithms that compute $\log(A)$ and one or more Fréchet derivatives $L_{\log}(A, E)$ and $L^{\star}_{\log}(A, E)$, making appropriate reuse of information from the logarithm computation in the calculation of the Fréchet derivatives.

3.5 Complex algorithm

We now give an algorithm to compute the matrix logarithm and one or more Fréchet derivatives and adjoints of Fréchet derivatives using complex arithmetic, building on Algorithm 3.2.1. As in [6] and [57, Alg. 11.9] we begin with a reduction to Schur form $A = QTQ^*$ (Q unitary, T upper triangular), because working with a triangular

matrix leads to a generally smaller operation count and better accuracy. We will use the relation $L_{\log}(A,E) = QL_{\log}(T, Q^*EQ)Q^*$ [57, Prob. 3.2]. Our algorithm employs the inverse scaling and squaring algorithm in [6, Alg. 4.1] and reduces to it if all the lines associated with the Fréchet derivative are removed.

Algorithm 3.5.1. *Given $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ and one or more $E \in \mathbb{C}^{n\times n}$ this algorithm computes the principal matrix logarithm $X = \log(A)$ and either of the Fréchet derivatives $L_{\log}(A,E)$ and $L^{\star}_{\log}(A,E)$.*

1 Compute a complex Schur decomposition $A = QTQ^*$.
2 $T_0 = T$
3 Determine the integers s and $m \in [1,7]$ as in [6, Alg. 4.1], at the same time computing (and storing, if Fréchet derivatives are required) the matrices $T_{k+1} = T_k^{1/2}$, $k = 0\colon s-1$ using the recurrence of [18], [57, Alg. 6.3].
4 $R = T_s - I$
5 Replace the diagonal and first superdiagonal of R by the diagonal and first superdiagonal of $T_0^{1/2^s} - I$ computed via [2, Alg. 2] and [60, (5.6)], respectively.
6 $X = 0$
7 for $i = 1\colon m$
8 Solve $(I + \beta_j^{(m)}R)U = \alpha_j^{(m)}R$ for U by substitution.
9 $X \leftarrow X + U$
10 end
11 $X \leftarrow 2^s X$
12 Replace $\mathrm{diag}(X)$ by $\log(\mathrm{diag}(T_0))$ and the elements of the first superdiagonal of X with those given by [57, (11.28)] taking $T = T_0$.
13 $X \leftarrow QXQ^*$
14 ... To compute $L_{\log}(A,E)$ for a given E:
15 $E_0 = Q^*EQ$
16 for $i = 1\colon s$
17 Solve the Sylvester equation $T_iE_i + E_iT_i = E_{i-1}$ for E_i by substitution.
18 end

19 $L = 0$

20 for $j = 1\colon m$

21 Compute $Y = \alpha_j^{(m)}(I + \beta_j^{(m)}R)^{-1}E_s(I + \beta_j^{(m)}R)^{-1}$ by substitution.

22 $L \leftarrow L + Y$

23 end

24 $L \leftarrow 2^s QLQ^*$

25 ... To compute $L_{\log}^{\star}(A, E)$ for a given E:

26 Execute lines 15–24 with E replaced by E^* and take the conjugate transpose of the result.

Cost: $25n^3$ flops for the Schur decomposition plus $(3 + (s + m)/3)n^3$ flops for X and $(8 + 2(s + m))n^3$ flops for each Fréchet derivative evaluation.

3.6 Real algorithm

If A and E are real and A has no eigenvalues on $\mathbb{R}^-$ then both $\log(A)$ and $L_{\log}(A, E)$ will be real. To avoid complex arithmetic we can modify Algorithm 3.5.1 to use a real Schur decomposition $A = QTQ^T$, where Q is orthogonal and T is upper quasi-triangular, that is, block upper triangular with diagonal blocks of dimension 1 or 2. The use of real arithmetic increases the efficiency of the algorithm, since a complex elementary operation has the same cost as two or more real elementary operations. It also avoids the result being contaminated by small imaginary parts due to rounding error and halves the required intermediate storage.

The main difference from Algorithm 3.5.1 is that since T is now upper quasi-triangular it is more complicated to replace the diagonal and superdiagonal elements of the shifted square root and final $\log(T)$ with more accurately computed ones.

Consider first the computation of $T^{1/2^s} - I$. To avoid cancellation we recompute the 1×1 diagonal blocks using [2, Alg. 2] and the 2×2 blocks using [6, Alg. 5.1], modified to use the recurrence of [52] for the square roots. We also recompute every superdiagonal $(i, i + 1)$ element for which the (i, i) and $(i + 1, i + 1)$ elements are in 1×1 blocks using [60, (5.6)].

As in the complex version of the algorithm, it is desirable to replace the diagonal blocks of the computed $\log(T)$ with more accurately computed ones. When the real

Schur decomposition is computed using `dgees` from LAPACK [11], as in MATLAB, the 2×2 diagonal blocks are of the form

$$B = \begin{bmatrix} a & b \\ c & a \end{bmatrix},$$

where $bc < 0$ and B has eigenvalues $\lambda_\pm = a \pm i(-bc)^{1/2}$. Let $\theta = \arg(\lambda_+) \in (-\pi, \pi)$. Using the polynomial interpolation definition of a matrix function [57, Def. 1.4] we can derive the formula

$$\log(B) = \begin{bmatrix} \log(a^2 - bc)/2 & \theta b(-bc)^{-1/2} \\ \theta c(-bc)^{-1/2} & \log(a^2 - bc)/2 \end{bmatrix}, \tag{3.6.1}$$

which can also be obtained by specializing a formula in [33, Lem. 3.3]. Since $bc < 0$, (3.6.1) involves no subtractive cancellation; so long as we can compute the scalar logarithm and the argument θ accurately we will obtain $\log(B)$ to high componentwise accuracy.

Algorithm 3.6.1. *Given $A \in \mathbb{R}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ and one or more $E \in \mathbb{R}^{n\times n}$ this algorithm computes the principal matrix logarithm $X = \log(A)$ and either of the Fréchet derivatives $L_{\log}(A, E)$ and $L^\star_{\log}(A, E)$, using only real arithmetic.*

1 Compute a real Schur decomposition $A = QTQ^T$.
2 $T_0 = T$
3 Determine the integers s and $m \in [1, 7]$ as in [6, Alg. 4.1], at the same time computing (and storing, if Fréchet derivatives are required) the matrices $T_{k+1} = T_k^{1/2}$, $k = 0\colon s - 1$ using the recurrence of [52], [57, Alg. 6.7].
4 $R = T_s - I$
5 Replace the diagonal blocks of R by the diagonal blocks of $T_0^{1/2^s} - I$ computed by [2, Alg. 2] for the 1×1 blocks and [6, Alg. 5.1] (with square roots computed by [57, (6.9)]) for the 2×2 blocks.
6 For every i for which t_{ii} and $t_{i+1,i+1}$ are in 1×1 diagonal blocks, recompute $r_{i,i+1}$ using [60, (5.6)].
7 Evaluate $X = 2^s r_m(R)$ as in lines 6–11 of Algorithm 3.5.1.
8 Recompute the block diagonal of X using (3.6.1) for the 2×2 blocks of T_0 and as $\log((T_0)_{ii})$ for the 1×1 blocks.
9 For every i for which x_{ii} and $x_{i+1,i+1}$ are in 1×1 diagonal blocks,

recompute $x_{i,i+1}$ using [57, (11.28)].

10 $X \leftarrow QXQ^T$

11 ... To compute $L_{\log}(A, E)$ or $L^{\star}_{\log}(A, E)$ for a given E,
execute lines 15–24 or line 26 of Algorithm 3.5.1.

The cost of this algorithm is essentially the same as Algorithm 3.5.1, except that the flops are now operations on real (as opposed to complex) operands.

3.7 Comparison with existing methods

In this section we give a comparison between our algorithms and those currently in the literature for computing the matrix logarithm and its Fréchet derivatives, concentrating mainly on computational cost. Section 3.8 contains a variety of numerical experiments comparing the accuracy of the algorithms empirically.

3.7.1 Methods to compute the logarithm

We have obtained a new algorithm for computing the logarithm of a real matrix (Algorithm 3.6.1). The relevant comparison is between the following three methods.

- `iss_schur_complex`: the complex Schur decomposition-based Algorithm 3.5.1, which is equivalent to [6, Alg. 4.1] when just the logarithm is required.
- `iss_schur_real`: our real Schur decomposition-based Algorithm 3.6.1, which is the real analogue of `iss_schur_complex`.
- `iss_noschur`: the transformation-free inverse scaling and squaring algorithm [6, Alg. 5.2] that requires only matrix multiplications and the solution of multiple-right-hand side linear systems. This algorithm uses only real arithmetic when A is real.

In general, `iss_noschur` is much more expensive than `iss_schur_real`. For example, if $s = 3$ and $m = 7$ in both `iss_schur_real` and `iss_noschur` and if `iss_noschur` requires five iterations (a typical average) to compute each square root (for which it uses a Newton iteration) then the operation counts are approximately $31n^3$ flops for `iss_schur_real` and $79n^3$ flops for `iss_noschur`.

It is worth noting that there is no real arithmetic version of the Schur–Parlett algorithm that underlies the MATLAB function `logm`, since the real Schur form is incompatible with the blocking requirements of the block Parlett recurrence [28], [57, Sec. 9.4].

3.7.2 Methods to compute the Fréchet derivative

We now compare our algorithms `iss_schur_complex` and `iss_schur_real` to existing methods for computing the Fréchet derivative.

Kenney and Laub give an algorithm based on a special Kronecker representation of $L_{\log}$ that solves Sylvester equations and employs a Padé approximant to the function $\tanh(x)/x$ [74], [57, Alg. 11.12]. We will refer to this as the Kronecker–Sylvester algorithm. The minimum cost of this algorithm per Fréchet derivative, assuming a Schur decomposition is used and that s is such that $\|I - T^{1/2^s}\|_1 \leq 0.63$, is the cost of solving $s+9$ triangular Sylvester equations and computing 16 products of a triangular matrix with a full matrix. This is to be compared with a smaller *maximum* cost for `iss_schur_complex` and `iss_schur_real` of s triangular Sylvester equations and $2m$ (≤ 14) multiple-right-hand side triangular substitutions. Moreover, the value of s for `iss_schur_complex` and `iss_schur_real` is generally smaller and potentially much smaller than for the Kronecker–Sylvester algorithm, and the latter always requires complex arithmetic, even when A and E are real.

Another method, which we denote by `dbl_size`, evaluates the left-hand side of the formula

$$\log\left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} \log(A) & L_{\log}(A,E) \\ 0 & \log(A) \end{bmatrix}, \tag{3.7.1}$$

from [57, (3.16)], by `iss_schur_real` (or `iss_schur_complex` when A or E is complex). This method has the disadvantages that it doubles the problem size and that the entire logarithm of the block matrix must be re-evaluated if we require further Fréchet derivatives, greatly increasing the cost of this method. On the other hand, the backward error analysis of [6] fully applies, giving a sharper backward error bound than (3.3.5) for the Fréchet derivative. However, backward error is now measured with respect to the matrix $\left[\begin{smallmatrix} A & E \\ 0 & A \end{smallmatrix}\right]$ rather than A and E separately. Moreover, it is unclear how to scale E: its norm is arbitrary because $L_{\log}$ is linear in its second argument, but

the size of $\|E\|$ affects both the accuracy and the cost of the inverse scaling and squaring method. One natural approach is to scale such that $\|E\| = \|A\|$ before computing $L_f(A, E)$ and undoing this scaling afterwards. This tends to avoid numerical errors caused by manipulating floating point numbers with large differences in magnitude. For simplicity we have left E unscaled in our tests since the chosen A and E already have norms of comparable size.

We mention two other methods. Dieci, Morini, and Papini [33] propose using the inverse scaling and squaring approach with adaptive Simpson's rule applied to the integral $L_{\log}(A, E) = \int_0^1 ((A-I)t+I)^{-1} E ((A-I)t+I)^{-1}\, dt$. Since we are interested in computing $L_{\log}$ to full precision the tolerance for the quadrature rule needs to be set to order u, and this makes the method prohibitively expensive, as each function evaluation requires two multiple-right-hand side solves. As noted in [33], this method is more appropriate when only low accuracy is required, so we will test it only for condition number estimation.

We also mention the complex step approximation $L_{\log}(A, E) \approx \operatorname{Im} \log(A + ihE)/h$ suggested by Al-Mohy and Higham [5], which is valid for real A and E and has error $O(h^2)$; unlike finite difference approximations it does not suffer from inherent cancellation, so h can be taken very small. Since the argument $A + ihE$ of the logarithm is complex we must use Algorithm 3.5.1 for the evaluation. As noted in [5], this approximation is likely to suffer from numerical instability if used with an algorithm that intrinsically employs complex arithmetic such as Algorithm 3.5.1. This is indeed what we observed, with relative errors of order at best 10^{-7}, so we will not consider this approach further.

3.8 Numerical experiments

Our numerical experiments are performed in either MATLAB R2012a or Fortran in IEEE double precision arithmetic. Throughout the experiments we use a set of 66 (mostly 10×10) test matrices, extending those used in [6] and [57, Sec. 11.7] which include matrices from the literature, the MATLAB `gallery` function, and the Matrix Computation Toolbox [50].

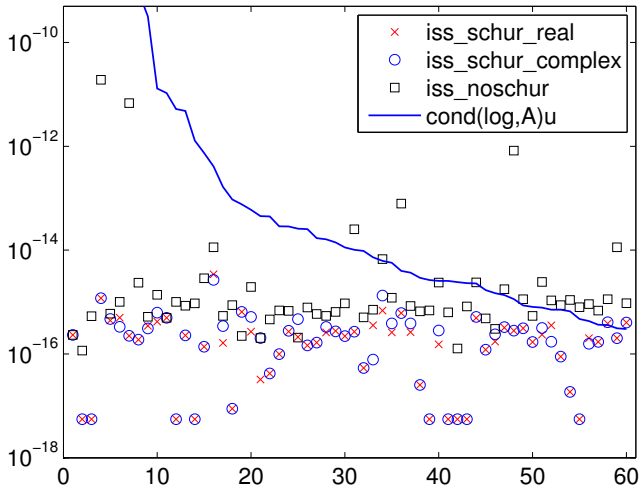

Figure 3.8.1: Normwise relative errors in computing the logarithm.

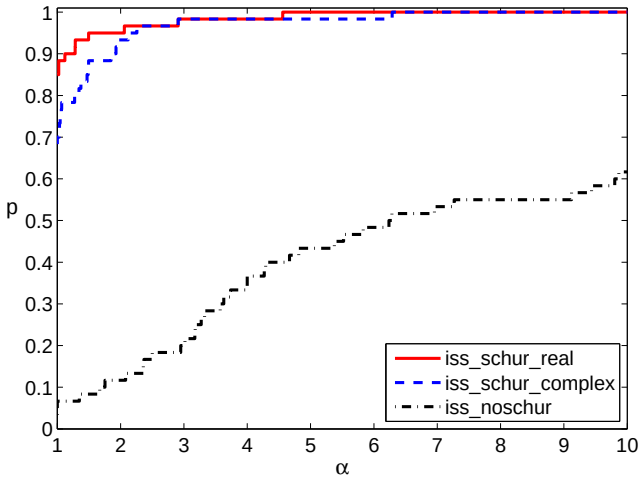

Figure 3.8.2: Performance profile for the data in Figure 3.8.1.

3.8.1 Real versus complex arithmetic for evaluating the logarithm

Our first experiment compares, on the 60 real matrices in the test set, the three algorithms defined in section 3.7.1. The matrices are used in real Schur form. We compute all relative errors in the 1-norm and for our "exact" logarithm we diagonalize A in 250-digit precision, using the Symbolic Math Toolbox, and use the relationship $\log(A) = V \log(D) V^{-1}$ where $A = VDV^{-1}$, rounding the result to double precision. If A is not diagonalizable then we add a small random perturbation of order 10^{-125} so that with high probability we can diagonalize it without affecting the accuracy of the final rounded solution [27].

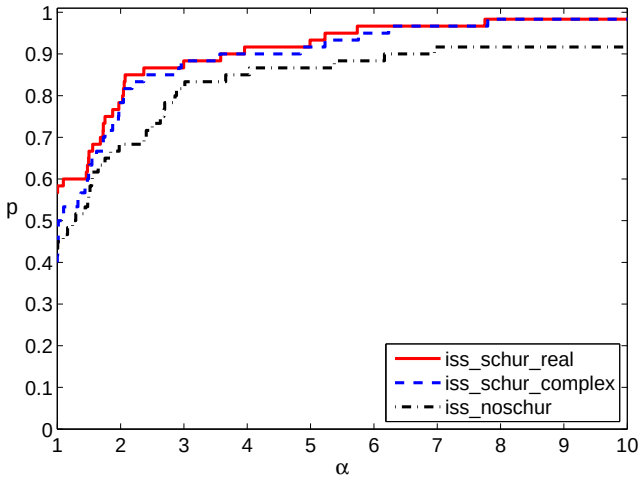

Figure 3.8.3: Performance profile for the accuracy of the logarithm algorithms on full matrices.

Figure 3.8.1 shows the normwise relative errors, with the problems ordered by decreasing condition number. The solid line denotes $\mathrm{cond}(\log, A)u$ (with $\|L_{\log}(A)\|_1$ approximated by $\|K_{\log}(A)\|_1$). Figure 3.8.2 shows the same data in the form of a performance profile [36], [49, Sec. 22.4], for which we use the transformation in [35] to lessen the influence of tiny relative errors.

The results show that `iss_schur_complex` and `iss_schur_real` are both significantly more accurate than `iss_noschur` (as also shown in [6] for `iss_schur_complex`) and that `iss_noschur` is often a little unstable (by which we mean errors exceed $n\,\mathrm{cond}(\log, A)u$, say). Moreover, `iss_schur_real` outperforms `iss_schur_complex`, showing that treating real matrices in real arithmetic benefits accuracy.

We repeated the experiments on full matrices and found that `iss_schur_real` is again the most accurate algorithm overall but that `iss_noschur` is now much more competitive with `iss_schur_real` and `iss_schur_complex`. The performance profile is given in Figure 3.8.3. The reason for the improved relative performance of `iss_noschur` is that rounding errors during the reduction to Schur form within `iss_schur_complex` and `iss_schur_real` tend to lead to larger errors for these algorithms than in the quasi-triangular case above.

Next, we compare the run times of Fortran implementations of `iss_schur_real` and `iss_schur_complex` to quantify the speed benefits of using real versus complex

Table 3.8.1: Run times in seconds for Fortran implementations of `iss_schur_real` and `iss_schur_complex` on random, full $n \times n$ matrices.

n	10	50	100	500	1000	2000
real	1e-3	6e-3	4e-2	1.55	9.22	65.53
complex	2e-3	1.1e-2	7.3e-2	3.6	21.59	147
ratio: complex/real	2.0	1.8	1.8	2.3	2.3	2.2

arithmetic. Square roots of (quasi-) triangular matrices are required in both algorithms, and since this operation is not one of the BLAS [19] it must be coded specially. A straightforward implementation using nested loops does not provide good performance, so a blocked algorithm described by Deadman, Higham, and Ralha [30] is used that yields a much more efficient implementation rich in matrix multiplication. Similarly, triangular Sylvester equations are solved using the recursive algorithm of Jonsson and Kågström [71].

Table 3.8.1 reports timings for the Fortran implementations compiled by `gfortran` linking to ACML BLAS, using the blocked square root algorithm described above and run on a quad-core Intel Xeon 64 bit machine. The tests were performed on full, random matrices with elements selected from the uniform $[0, 1)$ distribution. We see that `iss_schur_real` is around twice as fast as `iss_schur_complex` for all n, which is consistent with the counts of real arithmetic operations (see section 3.6).

3.8.2 Fréchet derivative evaluation

We now test our algorithms against the alternatives mentioned in section 3.7.2 for computing Fréchet derivatives. The matrices are in real or complex Schur form according as the original matrix is real or complex. We define `iss_schur` to be the algorithm that invokes Algorithm 3.6.1 when A is real and otherwise invokes Algorithm 3.5.1. In each test we take for E, the direction in which to calculate the Fréchet derivative, a random real matrix with normal $(0, 1)$ distributed elements.

In other experiments not reported here we took E to be a matrix such that $\|A^{2m+1}E\|_1 = \|A^{2m+1}\|_1\|E\|_1$ in an attempt to maximize the gap between the true

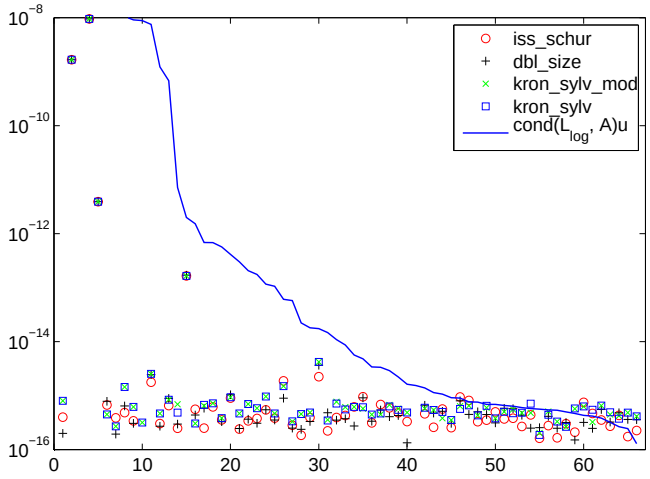

Figure 3.8.4: Normwise relative errors in computing $L_{\log}(A, E)$.

backward error $\|\Delta E\|/\|E\|$ and its upper bound in (3.3.5). We obtained similar results to those presented.

Figure 3.8.4 shows the relative errors of the Fréchet derivatives computed with the methods described in section 3.7.2. Here, `kron_sylv` denotes the implementation of the algorithm of Kenney and Laub [74], [57, Alg. 11.12] in the Matrix Function Toolbox [51] (named `logm_frechet_pade` there), and `kron_sylv_mod` denotes a modification of it in which calls to `logm` are replaced by calls to the more accurate `iss_schur_complex`. The values are ordered by descending condition number $\mathrm{cond}(L_{\log}, A)$, where, analogously to [3], we estimate $\mathrm{cond}(L_{\log}, A)$ with finite differences, by taking (1.3.2) with $f \leftarrow L_{\log}$ and a random direction E of norm 10^{-8}, employing the Kronecker form of the derivative.[1] In Figure 3.8.5 we present the same data as a performance profile.

The relative errors in Figure 3.8.4 show that the algorithms all performed very stably. We see from Figure 3.8.5 that the two most accurate methods for Fréchet derivative computation are `iss_schur` and `dbl_size`. Figure 3.8.5 also shows that using the more accurate logarithm evaluation in `kron_sylv_mod` produces a slight improvement in accuracy over `kron_sylv`.

Testing with full matrices we obtain similar results, although most relative errors are now within an order of magnitude of $\mathrm{cond}(L_{\log}, A)$. The associated performance profile is shown in Figure 3.8.6. The much tighter grouping of the algorithms is again due to the relative errors introduced by the Schur reduction.

[1] In chapter 5 we design a method of estimating $\mathrm{cond}(L_{\log}, A)$ more reliably.

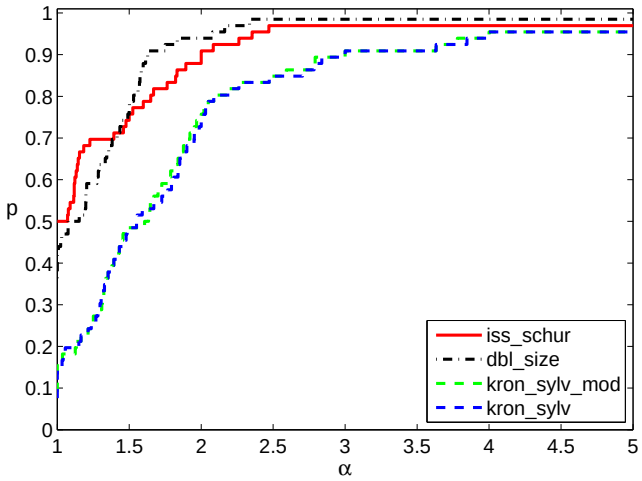

Figure 3.8.5: Performance profile for the data in Figure 3.8.4.

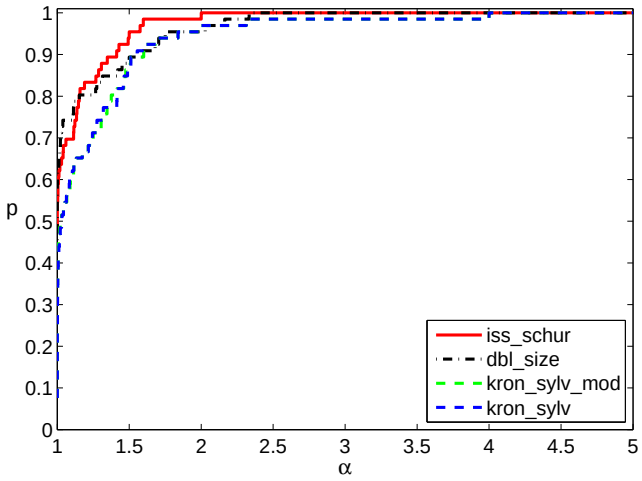

Figure 3.8.6: Performance profile for the accuracy of the $L_{\log}(A, E)$ algorithms for full matrices.

3.8.3 Condition number estimation

We now consider the estimation of $\|K_{\log}(A)\|_1$, which is the key quantity needed to estimate $\mathrm{cond}(\log, A)$ (see Lemma (1.3.1)). We obtain the exact value by explicitly computing the $n^2 \times n^2$ matrix $K_{\log}(A)$ via [57, Alg. 3.17] and taking its 1-norm. The matrices are in real or complex Schur form.

We use Algorithm 1.3.2 with three different methods for computing the Fréchet derivatives. The first is our new algorithm, `iss_schur` (again denoting the use of Algorithm 3.5.1 or Algorithm 3.6.1 as appropriate). The second is a general purpose method based on finite differences, as implemented in the code `funm_condest1` from the Matrix Function Toolbox [51]. The last method, denoted `integral`, is the quadrature method of [33] described in section 3.7.2, used with quadrature tolerance 10^{-10}. We do not test `dbl_size` or `kron_sylv` within Algorithm 1.3.2, as these algorithms are substantially more expensive than `iss_schur` (see section 3.7) and no more accurate (as shown in the previous subsection).

In Figure 3.8.7 we plot the ratios of the estimated condition numbers to the accurately computed ones, sorted by decreasing condition number. These underestimation ratios should be less than or equal to 1 as we use the 1-norm power method to estimate $\|K_{\log}(A)\|_1$. From these ratios we see that while the `funm_condest1` and `integral` methods provide good estimates for most of the well conditioned problems, they produce estimates many orders of magnitude too small for some of the more ill conditioned problems. On the other hand, `iss_schur` gives excellent estimates that are at worst a factor 0.47 smaller than the true value, making it the clear winner.

The unreliability of `funm_condest1` can be attributed to the presence of the potentially very large term $\|\log(A)\|_1$ in the steplength formula used for the finite differences [57, (3.23)]. It is not clear whether a different choice of finite difference steplength leading to better results can be derived.

Estimating the condition number using Algorithm 1.3.2 with $t = 2$, as in our algorithms, requires around 8 Fréchet derivative evaluations. With the mean s and m found to be 4 and 6 respectively in our tests, the cost of computing the logarithm and estimating its condition number via our algorithms is around 8 times that of the logarithm alone. By reducing t in the block 1-norm estimation algorithm to 1, this can be lowered to 4 times the cost at the risk of lower reliability.

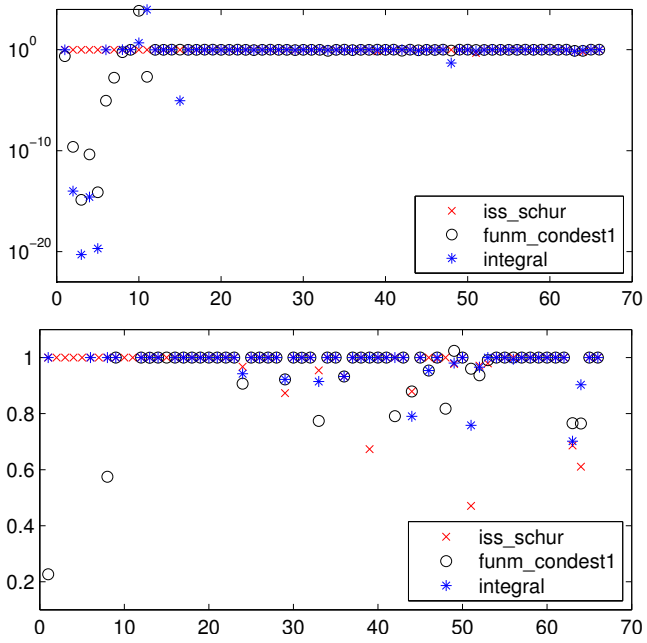

Figure 3.8.7: Underestimation ratios $\eta/\|K_{\log}(A)\|_1$ where η is the estimate of $\|K_{\log}(A)\|_1$; lower plot is zoomed version of upper plot.

Chapter 4

Higher Order Fréchet Derivatives of Matrix Functions and the Level-2 Condition Number

4.1 Introduction

When performing numerical computations it is often helpful to return the condition number of the problem, allowing the user to assess the accuracy of the computed solution. In this chapter we will refer to the condition numbers cond_{abs} and cond_{rel}, defined by equations (1.3.1) and (1.3.2) respectively, as level-1 condition numbers. Since the estimation of the condition number is itself subject to rounding errors it is important to know the condition number of the condition number, which we call the level-2 condition number. It has been shown by Demmel [31, Sec. 7] for matrix inversion, the eigenproblem, polynomial zero-finding, and pole assignment in linear control problems that the level-1 and level-2 (relative) condition numbers are equivalent; for matrix inversion D. J. Higham [48] obtains explicit constants in the equivalence. Cheung and Cucker [24] also derive tight bounds on the level-2 (relative) condition number for a class of functions that includes the matrix inverse. One purpose of our work in this chapter is to investigate the connection between the level-1 and level-2 (absolute) condition numbers of general matrix functions.

The level-2 condition number is intimately connected with the second Fréchet derivative. There is little or no literature on higher Fréchet derivatives of matrix

functions. Another goal of this chapter is to develop the existence theory for higher order derivatives and to derive methods for computing the derivatives.

This chapter is organized as follows. In section 4.2 we define higher order Fréchet derivatives and summarize previous research into derivatives of matrix functions. In section 4.3 we obtain conditions for the existence and continuity of the kth order Fréchet derivative and also give an algorithm for computing it given only the ability to compute the matrix function f. The Kronecker matrix form of the kth order Fréchet derivative is discussed in section 4.4 and an algorithm is given for computing it. In section 4.5 we define and analyze the level-2 condition number. We derive an upper bound for general functions f in terms of the second Kronecker form. For the exponential function we show that the level-1 and level-2 absolute condition numbers are equal and that the level-2 relative condition number cannot be much larger than the level-1 relative condition number. We also derive an exact relation between the level-1 and level-2 absolute condition numbers of the matrix inverse, as well as a result connecting the two absolute condition numbers for Hermitian matrices for a class of functions that includes the logarithm and square root. Via numerical experiments we compare the level-1 and level-2 condition numbers with different functions on unstructured matrices.

4.2 Higher order derivatives

The kth Fréchet derivative of $f : \mathbb{C}^{n\times n} \mapsto \mathbb{C}^{n\times n}$ at $A \in \mathbb{C}^{n\times n}$ can be defined recursively as the unique multilinear function[1] $L_f^{(k)}(A)$ of the matrices $E_i \in \mathbb{C}^{n\times n}$, $i = 1\colon k$, that satisfies

$$\begin{aligned} L_f^{(k-1)}(A + E_k, E_1, \dots, E_{k-1}) - L_f^{(k-1)}(A, E_1, \dots, E_{k-1}) \\ - L_f^{(k)}(A, E_1, \dots, E_k) = o(\|E_k\|), \end{aligned} \tag{4.2.1}$$

where $L_f^{(1)}(A)$ is the first Fréchet derivative. Assuming $L_f^{(k)}(A)$ is continuous at A, we can view the kth Fréchet derivative as a mixed partial derivative

$$L_f^{(k)}(A, E_1, \dots, E_k) = \frac{\partial}{\partial s_1} \cdots \frac{\partial}{\partial s_k}\bigg|_{(s_1,\dots,s_k)=0} f(A + s_1E_1 + \cdots + s_kE_k), \tag{4.2.2}$$

[1] We write $L_f^{(k)}(A)$ as a shorthand for $L_f^{(k)}(A, \cdot, \dots, \cdot)$ when we want to refer to the mapping at A and not its value in a particular set of directions.

as explained by Nashed [86, Sec. 9] in the more general setting of Banach spaces. From this equality it is clear that the order in which the derivatives are taken is irrelevant [17, p. 313], [45, Thm. 8], [84, Thm. 4.3.4], so the E_i can be permuted without changing the value of the Fréchet derivative. The kth Fréchet derivative of a matrix function also satisfies the sum, product, and chain rules (the proofs given in [57, Chap. 3] for the first Fréchet derivative are readily extended to higher order derivatives). Further information on higher order Fréchet derivatives in Banach spaces can be found in [34, Sec. 8.12], [72, Chap. 17], and [84, Sec. 4.3], for example.

We mention that some authors prefer to denote the kth Fréchet derivative by $D^k f(A)(E_1, \dots, E_k)$. Our notation has the advantage of being consistent with the notation in the matrix function literature for the first Fréchet derivative (1.2.1).

Previous research into higher derivatives of matrix functions has primarily focused on different types of derivative. Mathias [80] defines the kth derivative of a matrix function by

$$\left.\frac{d^k}{dt^k}\right|_{t=0} f(A(t)), \tag{4.2.3}$$

where $A(t) : \mathbb{R} \mapsto \mathbb{C}^{n\times n}$ is a k times differentiable path at $t = 0$ with $A(0) = A$. When $A'(0) = E$, the first derivative of f along the path $A(t)$ is equivalent to the first Fréchet derivative (assuming the latter exists) but this agreement does not hold for higher order derivatives.

Najfeld and Havel [85] investigate a special case of Mathias' type of derivative that corresponds to $A(t) = A + tV$. They find that [85, Thm. 4.13]

$$f\left(\begin{bmatrix} A & V & & \\ & \ddots & \ddots & \\ & & A & V \\ & & & A \end{bmatrix}\right) = \begin{bmatrix} f(A) & D_V^{[1]} f(A) & \cdots & \dfrac{D_V^{[q]} f(A)}{q!} \\ & f(A) & \ddots & \vdots \\ & & \ddots & D_V^{[1]} f(A) \\ & & & f(A) \end{bmatrix}, \tag{4.2.4}$$

where the argument of f is a block $q \times q$ matrix and $D_V^{[k]} f(A) = \left.\frac{d^k}{dt^k}\right|_{t=0} f(A(t))$. This is a generalization of the formula for evaluating a matrix function on a Jordan block [57, Def. 1.2].

There is also a componentwise derivative for matrix functions (including the trace and determinant etc.) which Magnus and Neudecker summarize in [79, pp. 171–173]. Athans and Schweppe apply this type of derivative to the matrix exponential in [14].

4.3 Existence and computation of higher Fréchet derivatives

One approach to investigating the existence of higher order Fréchet derivatives is to generalize the series of results for the first Fréchet derivative found in [57, Chap. 3]. However, this yields a somewhat lengthy development. Instead we present an approach that leads more quickly to the desired results and also provides a scheme for computing the Fréchet derivatives.

We first state three existing results on which we will build. Let $\mathcal{D}$ be an open subset of $\mathbb{C}$ and denote by $\mathbb{C}^{n\times n}(\mathcal{D},p)$ the set of matrices in $\mathbb{C}^{n\times n}$ whose spectrum lies in $\mathcal{D}$ and whose largest Jordan block is of size p.

Theorem 4.3.1 (Mathias [80, Lem. 1.1])**.** *Let f be $p-1$ times continuously differentiable on $\mathcal{D}$. Then f exists and is continuous on $\mathbb{C}^{n\times n}(\mathcal{D},p)$.*

Theorem 4.3.2. *Let f be $2p-1$ times continuously differentiable on $\mathcal{D}$. Then for $A\in\mathbb{C}^{n\times n}(\mathcal{D},p)$ the Fréchet derivative $L_f(A,E)$ exists and is continuous in both A and $E\in\mathbb{C}^{n\times n}$.*

Proof. This is a straightforward strengthening of [57, Thm. 3.8] (which has $p=n$) with essentially the same proof. □

Theorem 4.3.3 (Mathias [80, Thm. 2.1])**.** *Let f be $2p-1$ times continuously differentiable on $\mathcal{D}$. For $A\in\mathbb{C}^{n\times n}(\mathcal{D},p)$,*

$$f\left(\begin{bmatrix} A & E\\ 0 & A\end{bmatrix}\right)=\begin{bmatrix} f(A) & L_f(A,E)\\ 0 & f(A)\end{bmatrix}. \tag{4.3.1}$$

We need the Gâteaux derivative of f at A in the direction E which (as we recall from (1.2.2)) is defined by

$$G_f(A,E)=\frac{d}{dt}\Big|_{t=0} f(A+tE)=\lim_{\epsilon\to 0}\frac{f(A+\epsilon E)-f(A)}{\epsilon}. \tag{4.3.2}$$

Furthermore recall that Gâteaux differentiability is a weaker notion than Fréchet differentiability: if the Fréchet derivative exists then the Gâteaux derivative exists and is equal to the Fréchet derivative. Conversely, if the Gâteaux derivative exists, is linear

in E, and is continuous in A, then f is Fréchet differentiable and the Gâteaux and Fréchet derivatives are the same [17, Sec. X.4], [86, Sec. 8, Rem. 3].

Now define the sequence $X_i \in \mathbb{C}^{2^i n \times 2^i n}$ by

$$X_i = I_2 \otimes X_{i-1} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \otimes I_{2^{i-1}} \otimes E_i, \qquad X_0 = A, \tag{4.3.3}$$

where $\otimes$ is the Kronecker product [47], [75, Chap. 12]. Thus, for example, $X_1 = \left[\begin{smallmatrix} A & E_1 \\ 0 & A \end{smallmatrix}\right]$ and

$$X_2 = \begin{bmatrix} A & E_1 & E_2 & 0 \\ 0 & A & 0 & E_2 \\ 0 & 0 & A & E_1 \\ 0 & 0 & 0 & A \end{bmatrix}. \tag{4.3.4}$$

We will need the following lemma, which is a corollary of [67, Thm. 3.2.10.1] and [80, Lem. 3.1].

Lemma 4.3.4. *If the largest Jordan block of $A \in \mathbb{C}^{n\times n}$ is of size p then the largest Jordan block of X_k is of size at most $2^k p$.*

Now we can give our main result, which generalizes Theorems 4.3.2 and 4.3.3.

Theorem 4.3.5. *Let $A \in \mathbb{C}^{n\times n}(\mathcal{D}, p)$, where p is the size of the largest Jordan block of A, and let f be $2^k p-1$ times continuously differentiable on $\mathcal{D}$. Then the kth Fréchet derivative $L_f^{(k)}(A)$ exists and $L_f^{(k)}(A, E_1, \ldots, E_k)$ is continuous in A and $E_1, \ldots, E_k \in \mathbb{C}^{n\times n}$. Moreover, the upper right $n \times n$ block of $f(X_k)$ is $L_f^{(k)}(A, E_1, \ldots, E_k)$.*

Proof. Our proof is by induction on k, with the base case $k = 1$ given by Theorems 4.3.2 and 4.3.3. Suppose the result holds for some m between 1 and $k-1$. To prove that it holds for $m+1$ consider

$$f(X_{m+1}) = f\left(\begin{bmatrix} X_m & I_{2^m} \otimes E_{m+1} \\ 0 & X_m \end{bmatrix}\right), \tag{4.3.5}$$

which exists by Lemma 4.3.4 and Theorem 4.3.1. If we apply (4.3.1) to $f(X_{m+1})$ we see that its upper-right quarter is

$$L_f(X_m, I_{2^m} \otimes E_{m+1}) = \lim_{\epsilon\to 0} \frac{f(X_m + \epsilon(I_{2^m} \otimes E_{m+1})) - f(X_m)}{\epsilon}, \tag{4.3.6}$$

since the Fréchet derivative equals the Gâteaux derivative.

Now consider $X_m+\epsilon(I_{2^m}\otimes E_{m+1})$, which is the matrix obtained from (4.3.3) with A replaced by $A+\epsilon E_{m+1}$. For ϵ sufficiently small the spectrum of $X_m+\epsilon(I_{2^m}\otimes E_{m+1})$ lies within $\mathcal{D}$ by continuity of the eigenvalues. Hence we can apply the inductive hypothesis to both $f(X_m)$ and $f(X_m+\epsilon(I_{2^m}\otimes E_{m+1}))$, to deduce that their upper-right $n\times n$ blocks are, respectively,

$$L_f^{(m)}(A,E_1,\ldots,E_m),\quad L_f^{(m)}(A+\epsilon E_{m+1},E_1,\ldots,E_m). \tag{4.3.7}$$

Hence the upper-right $n\times n$ block of (4.3.6), which is also the upper-right $n\times n$ block of $f(X_{m+1})$ is

$$\begin{aligned}[f(X_{m+1})]_{1n} &= \lim_{\epsilon\to 0}\frac{L_f^{(m)}(A+\epsilon E_{m+1},E_1,\ldots,E_m)-L_f^{(m)}(A,E_1,\ldots,E_m)}{\epsilon}\\ &= \left.\frac{d}{dt}\right|_{t=0} L_f^{(m)}(A+tE_{m+1},E_1,\ldots,E_m),\end{aligned} \tag{4.3.8}$$

which is the Gâteaux derivative of the mth Fréchet derivative in the direction E_{m+1}. From our earlier discussion of the Gâteaux derivative we need to show that (4.3.8) is continuous in A and linear in E_{m+1} so that it is equal to the $(m+1)$st Fréchet derivative.

The continuity in A is trivial since f is sufficiently differentiable to be a continuous function of X_m by Theorem 4.3.1 and the map from $f(X_m)$ to its upper-right $n\times n$ block is also continuous. Now we show the linearity in E_{m+1}. Let us denote by σ the map from a matrix to its upper-right $n\times n$ block. Recalling that (4.3.8) is the upper right $n\times n$ block of (4.3.6), since the first Fréchet derivative is linear in its second argument we have

$$\begin{aligned}\left.\frac{d}{dt}\right|_{t=0} L_f^{(m)}(A+t(E+F),E_1,\ldots,E_m) &= \sigma L_f(X_m,I_{2^m}\otimes(E+F))\\ &= \sigma L_f(X_m,I_{2^m}\otimes E)+\sigma L_f(X_m,I_{2^m}\otimes F)\\ &= \left.\frac{d}{dt}\right|_{t=0} L_f^{(m)}(A+tE,E_1,\ldots,E_m)\\ &\quad + \left.\frac{d}{dt}\right|_{t=0} L_f^{(m)}(A+tF,E_1,\ldots,E_m),\end{aligned}$$

which shows the required linearity. We have now shown that the Gâteaux derivative of the mth Fréchet derivative is equal to the $(m+1)$st Fréchet derivative. The proof follows by induction. □

Example 4.3.6. We can also show that the assumption that f be $2^k p-1$ times continuously differentiable is sometimes necessary. Consider the function $f(x) = \iint H(x)\,dx$ where $H(x)$ is the Heaviside function with $H(0) = 1$. This means that $f(x) = x^2/2$ and $f'(x) = x$ for $x \geq 0$ and $f(x) = f'(x) = 0$ for $x < 0$. In particular f only has one continuous derivative.

Let $J = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$ so that $f(J) = 0$. For a contradiction assume that f is Fréchet differentiable at J. Then the Fréchet derivative is equal to the Gâteaux derivative

$$L_f(J, E) = \lim_{\epsilon \to 0} \frac{f(J + \epsilon E) - f(J)}{\epsilon}. \tag{4.3.9}$$

Let $E = \left[\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right]$ so that $f(J+\epsilon E) = f(\left[\begin{smallmatrix} \epsilon & 1 \\ 0 & -\epsilon \end{smallmatrix}\right])$. If $f[\lambda_1, \lambda_2]$ denotes the divided difference of f at λ_1, λ_2 then from [57, Thm. 4.11]

$$f\left(\begin{bmatrix} \lambda_1 & t \\ 0 & \lambda_2 \end{bmatrix}\right) = \begin{bmatrix} f(\lambda_1) & tf[\lambda_1, \lambda_2] \\ 0 & f(\lambda_2) \end{bmatrix},$$

and therefore we can easily show

$$f(J + \epsilon E) = \begin{cases} \begin{bmatrix} 0 & -\epsilon/4 \\ 0 & \epsilon^2/2 \end{bmatrix} & \text{if } \epsilon < 0, \\ \begin{bmatrix} \epsilon^2/2 & \epsilon/4 \\ 0 & 0 \end{bmatrix} & \text{if } \epsilon > 0. \end{cases} \tag{4.3.10}$$

Taking the left-hand limit and right-hand limit of (4.3.9) we obtain

$$\lim_{\epsilon \to 0-} \frac{f(J + \epsilon E) - f(J)}{\epsilon} = \begin{bmatrix} 0 & -1/4 \\ 0 & 0 \end{bmatrix}, \tag{4.3.11}$$

$$\lim_{\epsilon \to 0+} \frac{f(J + \epsilon E) - f(J)}{\epsilon} = \begin{bmatrix} 0 & 1/4 \\ 0 & 0 \end{bmatrix}. \tag{4.3.12}$$

Since these matrices differ the limit does not exist and therefore f is not Fréchet differentiable at J.

As an example of how to use this to compute higher order Fréchet derivatives, Theorem 4.3.5 shows that the second derivative $L_f^{(2)}(A, E_1, E_2)$ is equal to the $(1, 4)$ block of $f(X_2)$, where X_2 is given by (4.3.4). More generally, the theorem gives the following algorithm for computing arbitrarily high order Fréchet derivatives.

Algorithm 4.3.7. *Given $A \in \mathbb{C}^{n\times n}$, the direction matrices $E_1, \dots, E_k \in \mathbb{C}^{n\times n}$ and a method to evaluate the matrix function f (assumed sufficiently smooth), this algorithm computes $L = L_f^{(k)}(A, E_1, \dots, E_k)$.*

1 $X_0 = A$
2 for $i = 1\colon k$
3 $\quad X_i = I_2 \otimes X_{i-1} + \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right] \otimes I_{2^{i-1}} \otimes E_i$
4 end
5 $F = f(X_k)$
6 Take L to be the upper-right $n \times n$ block of F.

Cost: Assuming that evaluating f at an $n \times n$ matrix costs $O(n^3)$ flops, applying f naively to X_k gives an overall cost of $O(8^k n^3)$ flops. Clearly this algorithm rapidly becomes prohibitively expensive as k grows, though exploiting the block structure of X_k could lead to significant savings in the computation.

To conclude this section we emphasize that the condition in Theorem 4.3.5 that f has $2^k p - 1$ derivatives, which stems from a bound on the worst possible Jordan structure of X_k, is not always necessary. It is easy to show there is always an E such that X_1 has a Jordan block of size $2p - 1$, so the condition is necessary for $k = 1$. However, in section 4.6, we provide an example of a matrix A for which fewer than $4p - 1$ derivatives are required for the existence of $f(X_2)$. Determining the exact number of derivatives needed for the existence of $f(X_k)$ given the Jordan structure of A is an open problem.

4.4 Kronecker forms of higher Fréchet derivatives

The Kronecker form of the first Fréchet derivative is given by (1.2.3). The principal attraction of the Kronecker form is that it explicitly captures the linearity of the Fréchet derivative, so that standard linear algebra techniques can be applied and certain explicit formulas and bounds can be obtained—as explained in section 1.3.

In this section we derive a Kronecker form for the kth Fréchet derivative and show how it can be computed. We assume that the kth Fréchet derivative $L_f^{(k)}(A, E_1, \dots, E_k)$ is continuous in A, which allows reordering of the E_i, as noted in section 4.2.

To begin, since $L_f^{(k)}(A, E_1, \ldots, E_k)$ is linear in E_k we have

$$\operatorname{vec}(L_f^{(k)}(A, E_1, \ldots, E_k)) = K_f^{(1)}(A, E_1, \ldots, E_{k-1}) \operatorname{vec}(E_k), \tag{4.4.1}$$

for some unique matrix $K_f^{(1)}(A, E_1, \ldots, E_{k-1}) \in \mathbb{C}^{n^2 \times n^2}$. Since the E_i can be permuted within the kth Fréchet derivative it follows that the E_i can be permuted in (4.4.1). For example, using the third Fréchet derivative,

$$K_f^{(1)}(A, E_1, E_2) \operatorname{vec}(E_3) = K_f^{(1)}(A, E_3, E_1) \operatorname{vec}(E_2).$$

We can use this fact to show that $K_f^{(1)}(A, E_1, \ldots, E_{k-1})$ is linear in each E_i.

Lemma 4.4.1. *Assuming that $L_f^{(k)}(A)$ is continuous in A, $K_f^{(1)}(A, E_1, \ldots, E_{k-1})$ is linear in each E_i.*

Proof. Using the definition (4.4.1) and the freedom to reorder the E_i within $K_f^{(1)}$ we write

$$\begin{aligned}
&K_f^{(1)}(A, E_1, \ldots, E_i + F_i, \ldots, E_{k-1}) \operatorname{vec}(E_k) \\
&\quad = \operatorname{vec}\big(L_f^{(k)}(A, E_1, \ldots, E_i + F_i, \ldots, E_{k-1}, E_k)\big) \\
&\quad = K_f^{(1)}(A, E_1, \ldots, E_{i-1}, E_{i+1}, \ldots, E_{k-1}, E_k) \operatorname{vec}(E_i + F_i) \\
&\quad = K_f^{(1)}(A, E_1, \ldots, E_{i-1}, E_{i+1}, \ldots, E_{k-1}, E_k) \left(\operatorname{vec}(E_i) + \operatorname{vec}(F_i)\right) \\
&\quad = \big(K_f^{(1)}(A, E_1, \ldots, E_i, \ldots, E_{k-1}) + K_f^{(1)}(A, E_1, \ldots, F_i, \ldots, E_{k-1})\big) \operatorname{vec}(E_k).
\end{aligned}$$

Since this is true for any matrix E_k, the matrices on the left and right-hand sides must be equal, and hence $K_f^{(1)}(A, E_1, \ldots, E_{k-1})$ is linear in E_i. □

Now since $K_f^{(1)}(A, E_1, \ldots, E_{k-1})$ is linear in each E_i it is linear in E_{k-1} and so,

$$\operatorname{vec}(K_f^{(1)}(A, E_1, \ldots, E_{k-1})) = K_f^{(2)}(A, E_1, \ldots, E_{k-2}) \operatorname{vec}(E_{k-1}), \tag{4.4.2}$$

where $K_f^{(2)}(A, E_1, \ldots, E_{k-2}) \in \mathbb{C}^{n^4 \times n^2}$. By the same argument as in the proof of Lemma 4.4.1 this matrix is also linear in each E_i and continuing this process we eventually arrive at $K_f^{(k)}(A) \in \mathbb{C}^{n^{2k} \times n^2}$, which we call the Kronecker form of the kth Fréchet derivative.

We can relate $K_f^{(k)}(A)$ to the kth Fréchet derivative by repeatedly taking vec of

the kth Fréchet derivative and using $\operatorname{vec}(AXB) = (B^T \otimes A)\operatorname{vec}(X)$:

$$\begin{aligned}
\operatorname{vec}(L_f^{(k)}(A, E_1, \dots, E_k)) &= K_f^{(1)}(A, E_1, \dots, E_{k-1})\operatorname{vec}(E_k) \\
&= \operatorname{vec}\big(K_f^{(1)}(A, E_1, \dots, E_{k-1})\operatorname{vec}(E_k)\big) \\
&= (\operatorname{vec}(E_k)^T \otimes I_{n^2}) K_f^{(2)}(A, E_1, \dots, E_{k-2})\operatorname{vec}(E_{k-1}) \\
&= (\operatorname{vec}(E_{k-1})^T \otimes \operatorname{vec}(E_k)^T \otimes I_{n^2})\operatorname{vec}(K_f^{(2)}(A, E_1, \dots, E_{k-2})) \\
&= \cdots \\
&= (\operatorname{vec}(E_1)^T \otimes \cdots \otimes \operatorname{vec}(E_k)^T \otimes I_{n^2})\operatorname{vec}(K_f^{(k)}(A)). \qquad (4.4.3)
\end{aligned}$$

In the remainder of this section we give an algorithm to compute the Kronecker form. Consider $K_f^{(k)}(A)e_m$ where e_m is the mth unit vector, so that this product gives us the mth column of $K_f^{(k)}(A)$. We know from the above that this can be written as $\operatorname{vec}(K_f^{(k-1)}(A, E_1))$, where $\operatorname{vec}(E_1) = e_m$. Therefore to obtain the mth column of $K_f^{(k)}(A)$ we require $K_f^{(k-1)}(A, E_1)$ and as above we can find each column of this matrix as $K_f^{(k-1)}(A, E_1)e_p = \operatorname{vec}(K_f^{(k-1)}(A, E_1, E_2)$ where E_2 is chosen so that $\operatorname{vec}(E_2) = e_p$. Continuing in this way we obtain the following algorithm, of which [57, Alg. 3.17] is the special case with $k = 1$.

Algorithm 4.4.2. *The following algorithm computes the Kronecker form $K_f^{(k)}(A)$ of the kth Fréchet derivative $L_f^{(k)}(A)$.*

```
for m_1 = 1: n^2
    Choose E_1 ∈ R^{n×n} such that vec(E_1) = e_{m_1}.
    for m_2 = 1: n^2
        Choose E_2 ∈ R^{n×n} such that vec(E_2) = e_{m_2}.
            ...
            for m_k = 1: n^2
                Choose E_k ∈ R^{n×n} such that vec(E_k) = e_{m_k}.
                Compute L_f^(k)(A, E_1, ..., E_k) using Algorithm 4.3.7.
                Set the m_kth column of K_f^(1)(A, E_1, ..., E_{k-1})
                to vec(L_f^(k)(A, E_1, ..., E_k)).
            end
            ...
```

12 Set the m_2th column of $K_f^{(k-1)}(A, E_1)$ to $\mathrm{vec}(K_f^{(k-2)}(A, E_1, E_2))$.

13 end

14 Set the m_1th column of $K_f^{(k)}(A)$ to $\mathrm{vec}(K_f^{(k-1)}(A, E_1))$.

15 end

Cost: $O(8^k n^{3+2k})$ flops, since line 8 is executed n^2 times for each of the k matrices E_i.

The cost of this method depends heavily upon k, which governs both the depth of the algorithm and the cost of evaluating the kth Fréchet derivative in line 8. However, even calculating the Kronecker form of the first Fréchet derivative costs $O(n^5)$ flops, so this algorithm is only viable for small matrices and small k. Nevertheless, the algorithm is useful for testing algorithms for estimating $\|K_f^{(k)}(A)\|$ and hence $\|L_f^{(k)}(A)\|$.

4.5 The level-2 condition number of a matrix function

It is important to understand how sensitive the condition number is to perturbations in A, since this will affect the accuracy of any algorithm attempting to estimate it, such as those in [4], [61], and the algorithms in chapter 3. The quantity that measures this sensitivity is called the level-2 condition number.

The level-2 condition number is obtained by taking the absolute (or relative) condition number of the absolute (or relative) condition number, offering four possibilities. In this investigation we mainly limit ourselves to analyzing the absolute condition number of the absolute condition number,

$$\mathrm{cond}_{\mathrm{abs}}^{[2]}(f, A) = \lim_{\epsilon \to 0} \sup_{\|Z\| \le \epsilon} \frac{|\,\mathrm{cond}_{\mathrm{abs}}(f, A + Z) - \mathrm{cond}_{\mathrm{abs}}(f, A)|}{\epsilon}, \tag{4.5.1}$$

where $\mathrm{cond}_{\mathrm{abs}}(f, X)$ is defined in (1.3.1). However in section 4.5.1 for the exponential we also consider the relative condition number of the relative condition number,

$$\mathrm{cond}_{\mathrm{rel}}^{[2]}(f, A) = \lim_{\epsilon \to 0} \sup_{\|Z\| \le \epsilon \|A\|} \frac{|\,\mathrm{cond}_{\mathrm{rel}}(f, A + Z) - \mathrm{cond}_{\mathrm{rel}}(f, A)|}{\epsilon\, \mathrm{cond}_{\mathrm{rel}}(f, A)}. \tag{4.5.2}$$

We begin this section by deriving a bound for the level-2 absolute condition number for general functions f in the Frobenius norm. Using the second Fréchet derivative we

have, from (1.3.1) and (4.2.1),

$$\operatorname{cond}_{\text{abs}}(f, A + Z) = \max_{\|E\|=1} \|L_f(A, E) + L_f^{(2)}(A, E, Z) + o(\|Z\|)\|. \tag{4.5.3}$$

Therefore using the triangle inequality in the numerator of (4.5.1) we obtain

$$\begin{aligned} |\operatorname{cond}_{\text{abs}}(f, A + Z) - \operatorname{cond}_{\text{abs}}(f, A)| &= \Big| \max_{\|E\|=1} \|L_f(A, E) + L_f^{(2)}(A, E, Z) + o(\|Z\|)\| \\ &\qquad - \max_{\|E\|=1} \|L_f(A, E)\| \Big| \\ &\leq \max_{\|E\|=1} \|L_f^{(2)}(A, E, Z) + o(\|Z\|)\|. \end{aligned}$$

Using this inequality in the definition of the level-2 condition number (4.5.1) we see

$$\begin{aligned} \operatorname{cond}_{\text{abs}}^{[2]}(f, A) &\leq \lim_{\epsilon \to 0} \sup_{\|Z\| \leq \epsilon} \max_{\|E\|=1} \|L_f^{(2)}(A, E, Z/\epsilon) + o(\|Z\|)/\epsilon\| \\ &= \sup_{\|Z\| \leq 1} \max_{\|E\|=1} \|L_f^{(2)}(A, E, Z)\| \\ &= \max_{\|Z\|=1} \max_{\|E\|=1} \|L_f^{(2)}(A, E, Z)\|, \end{aligned} \tag{4.5.4}$$

where the supremum can be replaced with a maximum because we maximizing a continuous function over the compact set $\|Z\| \leq 1$ and furthermore the maximum is attained for $\|Z\| = 1$ because the second Fréchet derivative is linear in Z. We will see in subsection 4.5.2 that this upper bound is attained for the matrix inverse and the Frobenius norm. Note that the upper bound (4.5.4) can be thought of as $\|L_f^{(2)}(A)\|$.

Now restricting ourselves to the Frobenius norm and recalling that $\|X\|_F = \|\operatorname{vec}(X)\|_2$ we obtain

$$\begin{aligned} \operatorname{cond}_{\text{abs}}^{[2]}(f, A) &\leq \max_{\|Z\|_F=1} \max_{\|E\|_F=1} \|L_f^{(2)}(A, E, Z)\|_F \\ &= \max_{\|Z\|_F=1} \max_{\|\operatorname{vec}(E)\|_2=1} \|K_f^{(1)}(A, Z) \operatorname{vec}(E)\|_2 \quad \text{by (4.4.1)} \\ &= \max_{\|Z\|_F=1} \|K_f^{(1)}(A, Z)\|_2 \\ &\leq \max_{\|Z\|_F=1} \|K_f^{(1)}(A, Z)\|_F \\ &= \max_{\|\operatorname{vec}(Z)\|_2=1} \|K_f^{(2)}(A) \operatorname{vec}(Z)\|_2 \quad \text{by (4.4.2)} \\ &= \|K_f^{(2)}(A)\|_2. \end{aligned} \tag{4.5.5}$$

For general functions f it is difficult to say more about the level-2 condition number. In the next few subsections we focus on the matrix exponential, inverse, and a class of functions containing both the logarithm and square root.

4.5.1 Matrix exponential

For the matrix exponential let us consider the level-2 absolute condition number in the 2-norm, for normal matrices

$$A = QDQ^*, \qquad Q \text{ unitary}, \ D = \operatorname{diag}(d_i). \tag{4.5.6}$$

Note first that for a normal matrix A, using the unitary invariance of the 2-norm we have $\|e^A\|_2 = \|e^D\|_2 = e^{\alpha(D)} = e^{\alpha(A)}$, where the spectral abscissa $\alpha(A)$ is the greatest real part of any eigenvalue of A.

Van Loan [105] shows that normality implies $\operatorname{cond}_{\mathrm{rel}}(\exp, A) = \|A\|_2$ and from (1.3.1) and (1.3.2) we therefore have $\operatorname{cond}_{\mathrm{abs}}(\exp, A) = e^{\alpha(A)}$.

To analyze the level-2 absolute condition number of the matrix exponential we require the following lemma.

Lemma 4.5.1. *For a normal matrix A and an arbitrary matrix Z,*

$$e^{\alpha(A)-\|Z\|_2} \le \|e^{A+Z}\|_2 \le e^{\alpha(A)+\|Z\|_2},$$

and for a given A both bounds are attainable for some Z.

Proof. To get the lower bound we recall from [57, Thm. 10.12] that $e^{\alpha(X)} \le \|e^X\|_2$ for any matrix $X \in \mathbb{C}^{n\times n}$. We also know from [44, Thm. 7.2.2] that the eigenvalues of $A + Z$ are at most a distance $\|Z\|_2$ from those of A and so

$$e^{\alpha(A)-\|Z\|_2} \le \|e^{A+Z}\|_2,$$

and it is easy to see that this inequality is attained for $Z = -\operatorname{diag}(\epsilon, \dots, \epsilon)$.

For the upper bound, using the Lie–Trotter product formula [57, Cor. 10.7] gives

$$e^{A+Z} = \lim_{m\to\infty} (e^{A/m} e^{Z/m})^m.$$

Hence we have

$$\begin{aligned} \|e^{A+Z}\|_2 &= \lim_{m\to\infty} \|(e^{A/m}e^{Z/m})^m\|_2 \le \lim_{m\to\infty} \|e^{A/m}\|_2^m \|e^{Z/m}\|_2^m \\ &= \lim_{m\to\infty} e^{\alpha(A/m)m} e^{\|Z/m\|_2 m} = e^{\alpha(A)+\|Z\|_2}. \end{aligned}$$

It is straightforward to show that $Z = \operatorname{diag}(\epsilon, \ldots, \epsilon)$ attains this upper bound, completing the proof. □

We can now show that the level-2 absolute condition number of the matrix exponential is equal to the level-1 absolute condition number for normal matrices.

Theorem 4.5.2. *Let $A \in \mathbb{C}^{n\times n}$ be normal. Then in the 2-norm* $\operatorname{cond}_{\text{abs}}^{[2]}(\exp, A) = \operatorname{cond}_{\text{abs}}(\exp, A)$.

Proof. By taking norms in the identity

$$L_{\exp}(A + Z, E) = \int_0^1 e^{(A+Z)(1-s)} E e^{(A+Z)s}\, ds, \tag{4.5.7}$$

from [57, eq. (10.15)] we obtain

$$\begin{aligned}\operatorname{cond}_{\text{abs}}(\exp, A + Z) &= \max_{\|E\|_2=1} \|L_{\exp}(A + Z, E)\|_2 \\ &\leq \int_0^1 \|e^{(A+Z)(1-s)}\|_2 \|e^{(A+Z)s}\|_2\, ds,\end{aligned}$$

and furthermore since scalar multiples of A are normal and $\alpha(sA) = s\alpha(A)$ for $s \geq 0$, we can apply Lemma 4.5.1 twice within the integral to get

$$\operatorname{cond}_{\text{abs}}(\exp, A + Z) \leq e^{\alpha(A)+\|Z\|_2}.$$

Also we can obtain the lower bound

$$\begin{aligned}\operatorname{cond}_{\text{abs}}(\exp, A + Z) &= \max_{\|E\|_2=1} \|L(A + Z, E)\|_2 \\ &\geq \|L_{\exp}(A + Z, I)\|_2 \\ &= \left\| \int_0^1 e^{(A+Z)(1-s)} e^{(A+Z)s}\, ds \right\|_2 \\ &= \|e^{A+Z}\|_2 \geq e^{\alpha(A)-\|Z\|_2},\end{aligned}$$

so that overall, combining these two inequalities,

$$e^{\alpha(A)-\|Z\|_2} \leq \operatorname{cond}_{\text{abs}}(\exp, A + Z) \leq e^{\alpha(A)+\|Z\|_2}. \tag{4.5.8}$$

With some further manipulation we can use these bounds to show that

$$\sup_{\|Z\|_2\leq\epsilon} |\operatorname{cond}_{\text{abs}}(\exp, A + Z) - e^{\alpha(A)}| \leq e^{\alpha(A)+\epsilon} - e^{\alpha(A)}. \tag{4.5.9}$$

From the definition of the level-2 condition number (4.5.1) we have

$$\operatorname{cond}_{\mathrm{abs}}^{[2]}(\exp, A) = \lim_{\epsilon\to 0} \sup_{\|Z\|_2\le\epsilon} \frac{|\operatorname{cond}_{\mathrm{abs}}(\exp, A+Z) - e^{\alpha(A)}|}{\epsilon}.$$

Using the upper bound (4.5.9) on the numerator we see that

$$\operatorname{cond}_{\mathrm{abs}}^{[2]}(\exp, A) \le \lim_{\epsilon\to 0} \frac{e^{\alpha(A)+\epsilon} - e^{\alpha(A)}}{\epsilon} = e^{\alpha(A)}.$$

For the lower bound we use the fact that $\operatorname{cond}_{\mathrm{abs}}(\exp, A+\epsilon I) = e^{\alpha(A)+\epsilon}$ (since $A+\epsilon I$ is normal) to obtain

$$\begin{aligned}\operatorname{cond}_{\mathrm{abs}}^{[2]}(\exp, A) &\ge \lim_{\epsilon\to 0} \frac{|\operatorname{cond}_{\mathrm{abs}}(\exp, A+\epsilon I) - e^{\alpha(A)}|}{\epsilon} \\ &= \lim_{\epsilon\to 0} \frac{e^{\alpha(A)+\epsilon} - e^{\alpha(A)}}{\epsilon} = e^{\alpha(A)}.\end{aligned}$$

This completes the proof, since $\operatorname{cond}_{\mathrm{abs}}(\exp, A) = e^{\alpha(A)}$. □

We can also show that the level-2 relative condition number cannot be much larger than the level-1 relative condition number for normal matrices. In the next result we exclude $A = 0$ because $\operatorname{cond}_{\mathrm{rel}}(\exp, 0) = 0$ and so the computation of $\operatorname{cond}_{\mathrm{rel}}^{[2]}(\exp, 0)$ involves division by 0 and is therefore undefined.

Theorem 4.5.3. *Let $A \in \mathbb{C}^{n\times n} \setminus \{0\}$ be normal. Then in the 2-norm*

$$1 \le \operatorname{cond}_{\mathrm{rel}}^{[2]}(\exp, A) \le 2\operatorname{cond}_{\mathrm{rel}}(\exp, A) + 1.$$

Proof. Combining the definition of the level-2 relative condition number (4.5.2) with the facts that $\operatorname{cond}_{\mathrm{rel}}(\exp, X) = \operatorname{cond}_{\mathrm{abs}}(\exp, X)\|X\|_2/\|e^X\|_2$ for any $X \in \mathbb{C}^{n\times n}$, by (1.3.1) and (1.3.2), and $\operatorname{cond}_{\mathrm{rel}}(\exp, A) = \|A\|_2$ for normal A (mentioned at the beginning of this section), we have

$$\operatorname{cond}_{\mathrm{rel}}^{[2]}(\exp, A) = \lim_{\epsilon\to 0} \sup_{\|Z\|_2\le\epsilon\|A\|_2} \frac{|\operatorname{cond}_{\mathrm{abs}}(\exp, A+Z)\frac{\|A+Z\|_2}{\|e^{A+Z}\|_2} - \|A\|_2|}{\epsilon\|A\|_2}. \tag{4.5.10}$$

For the lower bound note that for any $X \in \mathbb{C}^{n\times n}$ we have $\|e^X\|_2 \le \operatorname{cond}_{\mathrm{abs}}(\exp, X)$ [57, Lem. 10.15] and therefore taking $X = A + Z$ we obtain

$$\frac{\operatorname{cond}_{\mathrm{abs}}(\exp, A+Z)\frac{\|A+Z\|_2}{\|e^{A+Z}\|_2} - \|A\|_2}{\epsilon\|A\|_2} \ge \frac{\|A+Z\|_2 - \|A\|_2}{\epsilon\|A\|_2}.$$

Using this bound in (4.5.10) we see that

$$\begin{aligned}\operatorname{cond}_{\text{rel}}^{[2]}(\exp, A) &\geq \lim_{\epsilon\to 0} \sup_{\|Z\|_2\leq\epsilon\|A\|_2} \frac{\|A+Z\|_2 - \|A\|_2}{\epsilon\|A\|_2} \\ &= \lim_{\epsilon\to 0} \frac{(1+\epsilon)\|A\|_2 - \|A\|_2}{\epsilon\|A\|_2} = 1,\end{aligned}$$

where the supremum is attained for $Z = \epsilon A$.

For the upper bound we first combine Lemma 4.5.1 and (4.5.8) to obtain

$$e^{-2\|Z\|_2} \leq \frac{\operatorname{cond}_{\text{abs}}(\exp, A+Z)}{\|e^{A+Z}\|_2} \leq e^{2\|Z\|_2}.$$

After some further manipulation we obtain the bound

$$\frac{|\operatorname{cond}_{\text{abs}}(\exp, A+Z)\frac{\|A+Z\|_2}{\|e^{A+Z}\|_2} - \|A\|_2|}{\epsilon\|A\|_2} \leq \frac{e^{2\|Z\|_2}-1}{\epsilon} + e^{2\|Z\|_2}\frac{\|Z\|_2}{\epsilon\|A\|_2}.$$

Using this inequality in (4.5.10) we see that

$$\begin{aligned}\operatorname{cond}_{\text{rel}}^{[2]}(\exp, A) &\leq \lim_{\epsilon\to 0} \sup_{\|Z\|_2\leq\epsilon\|A\|_2} \left(\frac{e^{2\|Z\|_2}-1}{\epsilon} + e^{2\|Z\|_2}\frac{\|Z\|_2}{\epsilon\|A\|_2}\right) \\ &= \lim_{\epsilon\to 0} \left(\frac{e^{2\epsilon\|A\|_2}-1}{\epsilon} + e^{2\epsilon\|A\|_2}\right) \\ &= 2\|A\|_2 + 1 \\ &= 2\operatorname{cond}_{\text{rel}}(\exp, A) + 1,\end{aligned}$$

which completes the proof. □

4.5.2 Matrix inverse

Assume now that A is a general nonsingular matrix. For the matrix inverse $f(A) = A^{-1}$, we have $L_f(A, E) = -A^{-1}EA^{-1}$. From the definition of the absolute condition number (1.3.1) we have

$$\operatorname{cond}_{\text{abs}}(x^{-1}, A) = \max_{\|E\|=1} \|A^{-1}EA^{-1}\|,$$

so for any subordinate matrix norm we conclude from Lemma 3.3.4 that

$$\operatorname{cond}_{\text{abs}}(x^{-1}, A) = \|A^{-1}\|^2, \tag{4.5.11}$$

and that this maximum is attained for a rank-1 matrix E. However the level-2 absolute condition number is best analyzed in the Frobenius norm, which is not subordinate.

The absolute condition number in the Frobenius norm is given by

$$\begin{aligned}\operatorname{cond}_{\text{abs}}(x^{-1}, A) &= \max_{\|E\|_F=1} \|A^{-1}EA^{-1}\|_F \\ &= \max_{\|\operatorname{vec}(E)\|_2=1} \|(A^{-T} \otimes A^{-1}) \operatorname{vec}(E)\|_2 \\ &= \|(A^{-T} \otimes A^{-1})\|_2 = \|A^{-1}\|_2^2, \end{aligned} \tag{4.5.12}$$

which is also shown in [48, (2.4)]. Using (4.5.12) in the definition of the level-2 absolute condition number (4.5.1) we have that in the Frobenius norm

$$\begin{aligned}\operatorname{cond}_{\text{abs}}^{[2]}(x^{-1}, A) &= \lim_{\epsilon\to 0} \sup_{\|E\|_F\le\epsilon} \frac{|\|(A+E)^{-1}\|_2^2 - \|A^{-1}\|_2^2|}{\epsilon}, \\ &= \lim_{\epsilon\to 0} \sup_{\|E\|_F\le\epsilon} \frac{|\tilde{\sigma}_n^{-2} - \sigma_n^{-2}|}{\epsilon}, \end{aligned}$$

where σ_n and $\tilde{\sigma}_n$ are the smallest singular values of A and $A+E$, respectively. Now consider the singular value decomposition (SVD) $A = U\Sigma V^*$, where $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_n)$ and $\sigma_1 \ge \cdots \ge \sigma_n > 0$. We know from [44, Cor. 8.6.2] that $\tilde{\sigma}_n = \sigma_n + e$ where $|e| \le \|E\|_2 \le \|E\|_F \le \epsilon$, and clearly the perturbation E that maximizes the numerator of the above equation moves σ_n closer to 0. Therefore when $\epsilon < \sigma_n$ the value of $\tilde{\sigma}_n$ that maximizes the numerator is $\tilde{\sigma}_n = \sigma_n - \epsilon$, which is attained by $E = U \operatorname{diag}(0, \dots, 0, -\epsilon)V^*$, with $\|E\|_2 = \|E\|_F = \epsilon$. Continuing with this choice of E we see that

$$\operatorname{cond}_{\text{abs}}^{[2]}(x^{-1}, A) = \lim_{\epsilon\to 0} \frac{|(\sigma_n - \epsilon)^{-2} - \sigma_n^{-2}|}{\epsilon} = \frac{2}{\sigma_n^3} = 2\|A^{-1}\|_2^3. \tag{4.5.13}$$

In fact the bound (4.5.4) on the level-2 absolute condition number is exact in this case. We can see this by maximizing the Frobenius norm of $L^{(2)}_{x^{-1}}(A, E, Z) = A^{-1}EA^{-1}ZA^{-1} + A^{-1}ZA^{-1}EA^{-1}$ using standard results.

From (4.5.12) and (4.5.13) we obtain the following result.

Theorem 4.5.4. *For nonsingular $A \in \mathbb{C}^{n\times n}$ in the Frobenius norm,*

$$\operatorname{cond}_{\text{abs}}^{[2]}(x^{-1}, A) = 2 \operatorname{cond}_{\text{abs}}(x^{-1}, A)^{3/2}. \tag{4.5.14}$$

This difference between the level-1 and level-2 absolute condition numbers for the inverse is intriguing since D. J. Higham [48, Thm. 6.1] shows that the relative level-1 and level-2 relative condition numbers for the matrix inverse are essentially equal for subordinate norms.

4.5.3 Hermitian matrices

The previous two sections gave relationships between the level-1 and level-2 absolute condition numbers for the exponential and the inverse. Interestingly these correspond closely to relationships between the first and second derivatives of the respective scalar functions: for $f(x) = e^x$, $|f''| = |f'|$ and for $f(x) = x^{-1}$, $|f''| = |2(f')^{3/2}|$. It is therefore natural to wonder whether analogous relations, such as $\operatorname{cond}^{[2]}_{\mathrm{abs}}(\log, A) = \operatorname{cond}_{\mathrm{abs}}(\log, A)^2$ and $\operatorname{cond}^{[2]}_{\mathrm{abs}}(x^{1/2}, A) = 2\operatorname{cond}_{\mathrm{abs}}(x^{1/2}, A)^3$, hold for suitable classes of A. The next result, which applies to Hermitian matrices and a class of functions that includes the logarithm and the square root, provides a partial answer.

Theorem 4.5.5. *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian with eigenvalues λ_i arranged so that $\lambda_1 \geq \cdots \geq \lambda_n$ and let $f : \mathbb{R} \to \mathbb{R}$ be such that $f(A)$ is defined and f has a strictly monotonic derivative. Then in the Frobenius norm,*

$$\operatorname{cond}_{\mathrm{abs}}(f, A) = \max_i |f'(\lambda_i)|. \tag{4.5.15}$$

Moreover, if the maximum in (4.5.15) is attained for a unique i, say $i = k$ (with $k = 1$ or $k = n$ since f' is monotonic), then

$$\operatorname{cond}^{[2]}_{\mathrm{abs}}(f, A) \geq |f''(\lambda_k)|. \tag{4.5.16}$$

Proof. Using [57, Cor. 3.16] we see that $\operatorname{cond}_{\mathrm{abs}}(f, A) = \max_{i,j} |f[\lambda_i, \lambda_j]|$, where $f[\lambda_i, \lambda_j]$ is a divided difference. But $f[\lambda_i, \lambda_j] = f'(\theta)$ for some θ on the closed interval between λ_i and λ_j [57, eq. (B.26)], and since f' is monotonic it follows that $|f[\lambda_i, \lambda_j]| \leq \max(|f'(\lambda_i)|, |f'(\lambda_j)|)$, with equality for $i = j$, and (4.5.15) follows.

We can write $A = Q\Lambda Q^*$, where Q is unitary and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Now define $Z = QDQ^*$, where D differs from the zero matrix only in that $d_{kk} = \epsilon$, so that the eigenvalues of $A + Z$ are λ_i, for $i \neq k$, and $\lambda_k + \epsilon$. Then, by the assumption on k, for sufficiently small ϵ the maximum of $|f'|$ over the eigenvalues of $A + Z$ is $|f'(\lambda_k + \epsilon)|$. Therefore using this Z in (4.5.1) we obtain

$$\begin{aligned}
\operatorname{cond}^{[2]}_{\mathrm{abs}}(f, A) &\geq \lim_{\epsilon\to 0} \left| \frac{\operatorname{cond}_{\mathrm{abs}}(f, A + Z) - \operatorname{cond}_{\mathrm{abs}}(f, A)}{\epsilon} \right| \\
&= \lim_{\epsilon\to 0} \left| \frac{|f'(\lambda_k + \epsilon)| - |f'(\lambda_k)|}{\epsilon} \right| \\
&= \left| \lim_{\epsilon\to 0} \frac{f'(\lambda_k + \epsilon) - f'(\lambda_k)}{\epsilon} \right| \\
&= |f''(\lambda_k)|. \qquad \square
\end{aligned}$$

Note that when applying this result to the matrix logarithm and square root we require A to be Hermitian positive definite, since these functions are not defined for matrices with negative eigenvalues.

4.5.4 Numerical experiments

We have a full understanding of the relationship between the level-1 and level-2 absolute condition numbers for the matrix inverse but our results for the matrix exponential, logarithm and square root are applicable only to normal or Hermitian matrices. We now give a numerical comparison of $\mathrm{cond}_{\mathrm{abs}}(f, A)$ and $\mathrm{cond}^{[2]}_{\mathrm{abs}}(f, A)$ for the matrix exponential, logarithm, and square root using unstructured matrices in the Frobenius norm.

Our test matrices are taken from the Matrix Computation Toolbox [50] and the MATLAB `gallery` function and we use 5×5 matrices because the cost of computing the first and second Kronecker forms using Algorithm 4.4.2 is $O(n^5)$ and $O(n^7)$ flops, respectively. Most of the matrices are neither normal nor Hermitian, so our previous analyses (except for the inverse) do not apply. The matrix exponential and logarithm are computed using the algorithms from [4] and chapter 3, and the square root is computed using the MATLAB function `sqrtm`. All experiments are performed in MATLAB 2013a.

For arbitrary matrices we are unable to compute the level-2 condition number exactly so instead we use the upper bound (4.5.5) which we refer to as `lvl2_bnd` in this section. Experiments comparing `lvl2_bnd` to the exact level-2 condition number for the inverse showed that they agreed reasonably well over our test matrices: the mean and maximum of the factor by which `lvl2_bnd` exceeded the level-2 condition number were 1.19 and 2.24 times, respectively. In 66% of cases the overestimation factor was less than 1.2. The level-1 condition number can be computed exactly in the Frobenius norm using [57, Alg. 3.17].

Figure 4.5.1 compares the level-1 condition number and `lvl2_bnd` for the matrix exponential on the 49 test matrices for which the matrix exponential did not overflow. The values are sorted in decreasing order of $\mathrm{cond}_{\mathrm{abs}}(\exp, A)$. Note that in each case

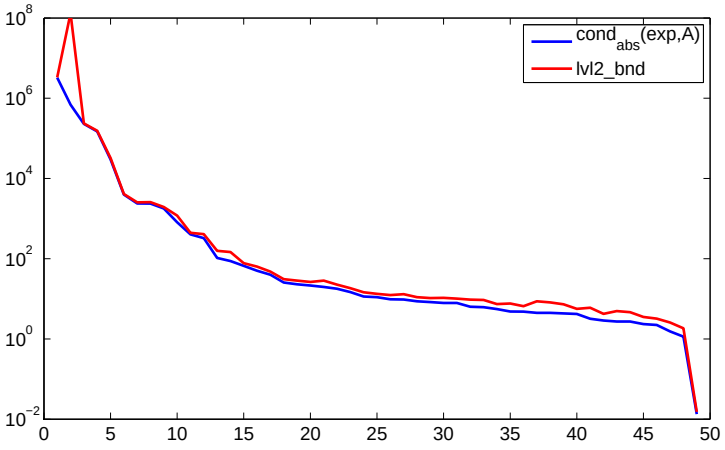

Figure 4.5.1: Level-1 absolute condition number and `lvl2_bnd` for the matrix exponential in the Frobenius norm sorted by decreasing value of $\mathrm{cond}_{\mathrm{abs}}(\exp, A)$.

`lvl2_bnd` is greater than or equal to the level-1 condition number. We see that the two lines are almost equal for arbitrary matrices in the Frobenius norm, with the most serious disagreement on the first few ill conditioned problems. This suggests that for the matrix exponential it may be possible to show that the level-1 and level-2 condition numbers are equal or approximately equal for a wider class of matrices than just the normal matrices, to which Theorem 4.5.2 applies.

Figure 4.5.2 compares the level-1 condition number and `lvl2_bnd` for the matrix logarithm over the 49 test matrices for which the matrix logarithm and its condition number are defined, sorted by decreasing values of $\mathrm{cond}_{\mathrm{abs}}(\log, A)$. In black we have plotted the square of the level-1 condition number; we see that it bears a striking similarity to the level-2 condition number, consistent with Theorem 4.5.5, since $|f''(\lambda)| = |f'(\lambda)^2|$.

Our final experiment compares the level-1 condition number and `lvl2_bnd` for the matrix square root on the 51 test matrices where the square root and its condition number are defined, again sorted by decreasing values of $\mathrm{cond}_{\mathrm{abs}}(x^{1/2}, A)$. Figure 4.5.3 shows two plots with the same data but with different y-axes so the fine details can be seen. In black we have plotted $2\,\mathrm{cond}_{\mathrm{abs}}(x^{1/2}, A)^3$ which, consistent with Theorem 4.5.5, provides a reasonable estimate of `lvl2_bnd` except for the first few problems, which are very ill conditioned.

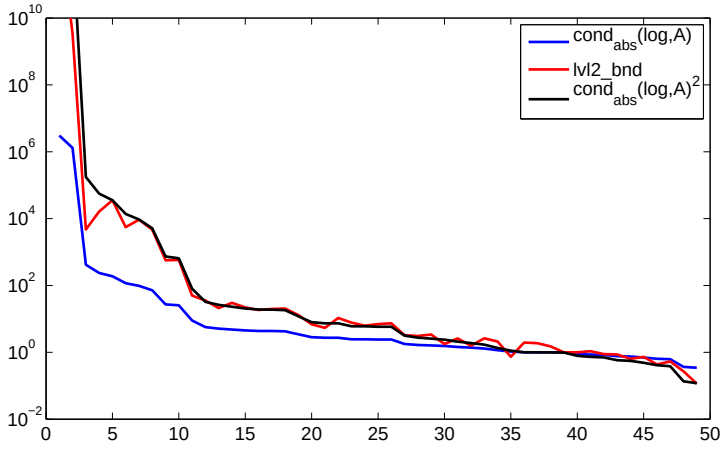

Figure 4.5.2: Level-1 absolute condition number and `lvl2_bnd` for the matrix logarithm in the Frobenius norm sorted by decreasing value of $\mathrm{cond}_{\mathrm{abs}}(\log, A)$. The black line is $\mathrm{cond}_{\mathrm{abs}}(\log, A)^2$.

4.6 The necessity of the conditions in Theorem 4.3.5

As mentioned at the end of section 4.3, our assumption in Theorem 4.3.5 that f has $2^k p - 1$ derivatives is not necessary for the existence of the kth Fréchet derivative. Our method of proof employs X_k and to evaluate $f(X_k)$ we need f to have $p_k - 1$ continuous derivatives, where p_k is the size of the largest Jordan block of X_k (see Theorem 4.3.1). From Theorem 4.3.4 we know that $p_k \leq 2^k p$ where p is the size of the largest Jordan block of A; this is the bound used in Theorem 4.3.5, but it is possible for X_k to have smaller Jordan blocks. The following example shows that for a specially chosen A only $4p - 2$ derivatives are needed for the existence of $f(X_2)$.

Take $A = J$ where $J \in \mathbb{C}^{n\times n}$ is a Jordan block of size n with eigenvalue 0. We first show that $\mathrm{rank}(X_2) \leq n-2$. Note from (4.3.4) that the first column of X_2 is 0 and the $(n+1)$st and $(2n+1)$st columns have at most n nonzero elements corresponding to the first columns of E_1 and E_2, respectively. Since A has 1s on its first superdiagonal we see that columns $2\colon n$ are the unit vectors $e_1, \ldots, e_{n-1}$ and they span all but the last element of the $(n+1)$st and $(2n+1)$st columns. Therefore if $[E_1]_{n,1} = 0$ or $[E_2]_{n,1} = 0$ the respective column can be written as a linear combination of columns $2 : n$. On the other hand if both are nonzero then we can write the $(2n+1)$st column as a linear combination of columns $2 : n+1$. Thus there are at most $n-2$ linearly independent columns in X_2 and so $\mathrm{rank}(X_2) \leq n-2$.

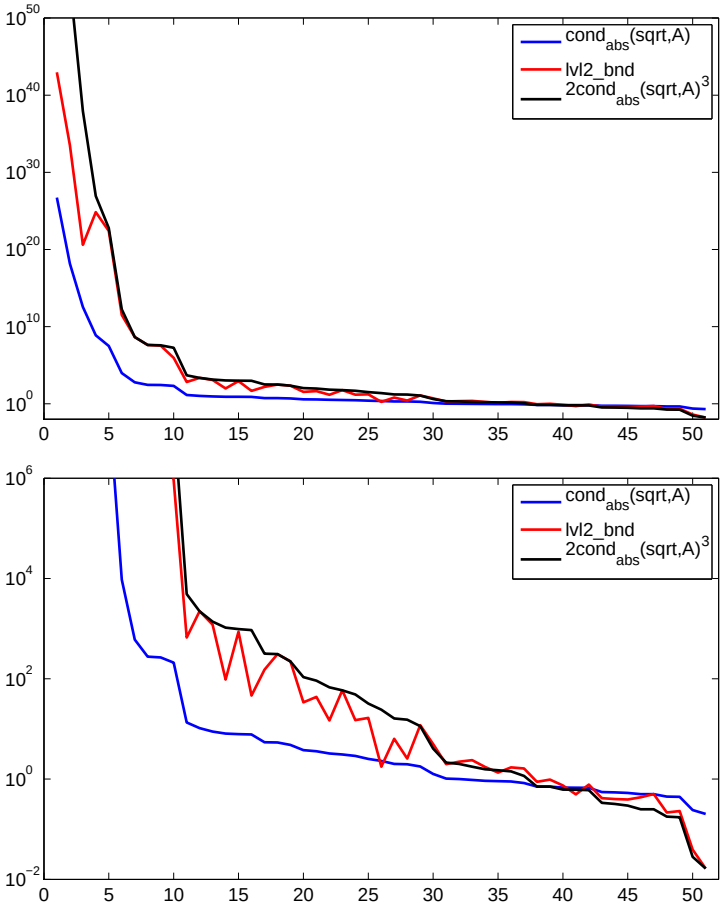

Figure 4.5.3: Top: Level-1 absolute condition number and `lvl2_bnd` for the matrix square root in the Frobenius norm sorted by decreasing value of $\mathrm{cond}_{\mathrm{abs}}(\log, A)$. The black line is $2\,\mathrm{cond}_{\mathrm{abs}}(x^{1/2}, A)^3$.
Bottom: Zoomed view of same data with narrowed y-axis.

This means that X_2 has at least two Jordan blocks and therefore the largest Jordan block is of size at most $4n-1$, meaning $4n-2$ derivatives of f are sufficient for the existence of $f(X_2)$ by Theorem 4.3.1, which is slightly weaker than the requirement in Theorem 4.3.5 (with $k=2$ and $p=n$) of $4n-1$ derivatives.

The general problem of determining the Jordan structure of X_k given the Jordan structure of A remains open. Indeed the minimum number of derivatives required for the existence of the kth Fréchet derivative is also unknown; this number is potentially less than the number of derivatives required for the existence of $f(X_k)$.

Chapter 5

Estimating the Condition Number of the Fréchet Derivative of a Matrix Function

5.1 Introduction

As mentioned in chapter 1 there are an increasing number of practical applications where the Fréchet derivatives of a matrix function are required. However, whilst a number of methods exist to compute Fréchet derivatives there is, to our knowledge, no literature on how accurately one might expect to compute them. In practice the input data can be subject to small perturbations such as rounding or measurement errors [55].

The aims of this chapter are to define the condition number of the Fréchet derivative, obtain bounds for it, and construct an efficient algorithm to estimate it.

Associated with the Fréchet derivative is its Kronecker matrix form (1.2.3), which we will reformulate slightly as

$$\operatorname{vec}(L_f(A,E)) = K_f(A)\operatorname{vec}(E) = (\operatorname{vec}(E)^T \otimes I_{n^2})\operatorname{vec}(K_f(A)), \tag{5.1.1}$$

where vec is the operator which stacks the columns of a matrix vertically from first to last and $\otimes$ is the Kronecker product. The second equality is obtained by using the formula

$$\operatorname{vec}(YAX) = (X^T \otimes Y)\operatorname{vec}(A)$$

in the special case, with $x \in \mathbb{C}^n$,

$$Ax = \operatorname{vec}(Ax) = (x^T \otimes I_n) \operatorname{vec}(A). \tag{5.1.2}$$

To investigate the condition number of the Fréchet derivative we will need higher order Fréchet derivatives of matrix functions, which were investigated in chapter 4. Although in chapter 4 the derivatives considered were of arbitrarily high order, we will only require the second Fréchet derivative; therefore it would be prudent to summarize the key results when specialized to the second Fréchet derivative.

The second Fréchet derivative $L_f^{(2)}(A, E_1, E_2) \in \mathbb{C}^{n \times n}$ is linear in both E_1 and E_2 and satisfies

$$L_f(A + E_2, E_1) - L_f(A, E_1) - L_f^{(2)}(A, E_1, E_2) = o(\|E_2\|). \tag{5.1.3}$$

Theorem 4.3.5 shows that a sufficient condition for the second Fréchet derivative $L_f^{(2)}(A, \cdot, \cdot)$ to exist is that f is $4p-1$ times continuously differentiable on an open set containing the eigenvalues of A, where p is the size of the largest Jordan block of A. This condition is certainly satisfied if f is $4n-1$ times continuously differentiable on a suitable open set. In this work we assume without further comment that this latter condition holds: it clearly does for common matrix functions such as the exponential, logarithm, and matrix powers A^t with $t \in \mathbb{R}$ or indeed for any analytic function. Under this condition it follows that $L_f^{(2)}(A, E_1, E_2)$ is independent of the order of E_1 and E_2 (which is analogous to the equality of mixed second order partial derivatives for scalar functions) as explained in section 4.2.

One method for computing Fréchet derivatives is to apply f to a $2n \times 2n$ matrix and read off the top right-hand block [57, (3.16)], [80]:

$$f\left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} f(A) & L_f(A, E) \\ 0 & f(A) \end{bmatrix}. \tag{5.1.4}$$

In Theorem 4.3.5 we showed that the second Fréchet derivative can be calculated in a similar way, as the top right-hand block of a $4n \times 4n$ matrix:

$$L_f^{(2)}(A, E_1, E_2) = f\left(\begin{bmatrix} A & E_1 & E_2 & 0 \\ 0 & A & 0 & E_2 \\ 0 & 0 & A & E_1 \\ 0 & 0 & 0 & A \end{bmatrix}\right)(1\colon n, 3n+1\colon 4n). \tag{5.1.5}$$

There is also a second order Kronecker matrix form (see section 4.4) analogous to (5.1.1) and denoted by $K_f^{(2)}(A) \in \mathbb{C}^{n^4 \times n^2}$, such that for any E_1 and E_2,

$$\operatorname{vec}(L_f^{(2)}(A, E_1, E_2)) = (\operatorname{vec}(E_1)^T \otimes I_{n^2}) K_f^{(2)}(A) \operatorname{vec}(E_2) \tag{5.1.6}$$

$$= (\operatorname{vec}(E_2)^T \otimes \operatorname{vec}(E_1)^T \otimes I_{n^2}) \operatorname{vec}(K_f^{(2)}(A)). \tag{5.1.7}$$

Note that $K_f^{(2)}(A)$ encodes information about $L_f^{(2)}(A)$—the Fréchet derivative of $L_f(A)$—in n^6 numbers. Our challenge is to estimate the condition number of L_f in just $O(n^3)$ flops.

This chapter is organized as follows. In section 5.2 we define the absolute and relative condition numbers of a Fréchet derivative, relate the two, and bound them above and below in terms of the second Fréchet derivative and the condition number of f. The upper and lower bounds that we obtain differ by at most a factor 2. Section 5.3 relates the bounds to the Kronecker matrix $K_f^{(2)}(A)$ at the cost, for the 1-norm, of introducing a further factor n of uncertainty and this leads to an $O(n^3)$ flops algorithm given in section 5.4 for estimating the 1-norm condition number of the Fréchet derivative. We test the accuracy and robustness of our algorithm via numerical experiments in section 5.5.

5.2 The condition number of the Fréchet derivative

We begin by proposing a natural definition for the absolute and relative condition numbers of a Fréchet derivative and showing that the two are closely related. We define the absolute condition number of a Fréchet derivative $L_f(A, E)$ by

$$\operatorname{cond}_{\text{abs}}(L_f, A, E) = \lim_{\epsilon \to 0} \sup_{\substack{\|\Delta A\| \le \epsilon \\ \|\Delta E\| \le \epsilon}} \frac{\|L_f(A + \Delta A, E + \Delta E) - L_f(A, E)\|}{\epsilon}, \tag{5.2.1}$$

which measures the maximal effect that small perturbations in the data A and E can have on the Fréchet derivative. Similarly we define the relative condition number by

$$\operatorname{cond}_{\text{rel}}(L_f, A, E) = \lim_{\epsilon \to 0} \sup_{\substack{\|\Delta A\| \le \epsilon \|A\| \\ \|\Delta E\| \le \epsilon \|E\|}} \frac{\|L_f(A + \Delta A, E + \Delta E) - L_f(A, E)\|}{\epsilon \|L_f(A, E)\|}, \tag{5.2.2}$$

where the changes are now measured in a relative sense. By taking ΔA and ΔE sufficiently small we can rearrange (5.2.2) to obtain the approximate upper bound

$$\frac{\|L_f(A+\Delta A, E+\Delta E) - L_f(A,E)\|}{\|L_f(A,E)\|} \lesssim \max\left(\frac{\|\Delta A\|}{\|A\|}, \frac{\|\Delta E\|}{\|E\|}\right) \mathrm{cond}_{\mathrm{rel}}(L_f, A, E). \tag{5.2.3}$$

A useful property of the relative condition number is its lack of dependence on the norm of E: for any positive scalar $s \in \mathbb{R}$,

$$\begin{aligned}
\mathrm{cond}_{\mathrm{rel}}(L_f, A, sE) &= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\| \le \epsilon\|A\| \\ \|\Delta E\| \le s\epsilon\|E\|}} \frac{\|L_f(A+\Delta A, sE+\Delta E) - L_f(A,sE)\|}{\epsilon\|L_f(A,sE)\|} \\
&= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\| \le \epsilon\|A\| \\ \|\Delta E/s\| \le \epsilon\|E\|}} \frac{\|L_f(A+\Delta A, E+\Delta E/s) - L_f(A,E)\|}{\epsilon\|L_f(A,E)\|} \\
&= \mathrm{cond}_{\mathrm{rel}}(L_f, A, E).
\end{aligned} \tag{5.2.4}$$

Furthermore we can obtain a similar relationship to (1.3.4) relating the absolute and relative condition numbers. This is useful since it allows us to state results and algorithms using the absolute condition number before reinterpreting them in terms of the relative condition number.

Lemma 5.2.1. *The absolute and relative condition numbers of the Fréchet derivative L_f are related by*

$$\mathrm{cond}_{\mathrm{rel}}(L_f, A, E) = \frac{\mathrm{cond}_{\mathrm{abs}}(L_f, A, sE)\|E\|}{\|L_f(A,E)\|}, \qquad s = \frac{\|A\|}{\|E\|}.$$

Proof. Using (5.2.4) and setting $s\|E\| = \|A\|$ and $\delta = \epsilon\|A\|$ we have

$$\begin{aligned}
\mathrm{cond}_{\mathrm{rel}}(L_f, A, E) &= \mathrm{cond}_{\mathrm{rel}}(L_f, A, sE) \\
&= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\| \le \epsilon\|A\| \\ \|\Delta E\| \le \epsilon s\|E\|}} \frac{\|L_f(A+\Delta A, sE+\Delta E) - L_f(A,sE)\|}{\epsilon\|L_f(A,sE)\|} \\
&= \frac{\|A\|}{\|L_f(A,sE)\|} \lim_{\delta\to 0} \sup_{\substack{\|\Delta A\| \le \delta \\ \|\Delta E\| \le \delta}} \frac{\|L_f(A+\Delta A, sE+\Delta E) - L_f(A,sE)\|}{\delta} \\
&= \frac{\mathrm{cond}_{\mathrm{abs}}(L_f, A, sE)\|A\|}{\|L_f(A,sE)\|} = \frac{\mathrm{cond}_{\mathrm{abs}}(L_f, A, sE)\|E\|}{\|L_f(A,E)\|}. \qquad \square
\end{aligned}$$

In order to bound the relative condition number we will derive computable bounds on the absolute condition number and use the relationship in Lemma 5.2.1 to translate them into bounds on the relative condition number. We first obtain lower bounds.

Lemma 5.2.2. *The absolute condition number of the Fréchet derivative satisfies both the lower bounds*

$$\begin{aligned}\mathrm{cond}_{\mathrm{abs}}(L_f, A, E) &\geq \mathrm{cond}_{\mathrm{abs}}(f, A),\\ \mathrm{cond}_{\mathrm{abs}}(L_f, A, E) &\geq \max_{\|\Delta A\|=1} \|L_f^{(2)}(A, E, \Delta A)\|.\end{aligned}$$

Proof. For the first bound we set $\Delta A = 0$ in (5.2.1) and use the linearity of the derivative:

$$\begin{aligned}\mathrm{cond}_{\mathrm{abs}}(L_f, A, E) &\geq \lim_{\epsilon\to 0} \sup_{\|\Delta E\|\leq\epsilon} \frac{\|L_f(A, E+\Delta E) - L_f(A, E)\|}{\epsilon}\\ &= \lim_{\epsilon\to 0} \sup_{\|\Delta E\|\leq\epsilon} \frac{\|L_f(A, \Delta E)\|}{\epsilon}\\ &= \mathrm{cond}_{\mathrm{abs}}(f, A).\end{aligned}$$

Similarly, for the second bound we set $\Delta E = 0$ and obtain, using (5.1.3),

$$\begin{aligned}\mathrm{cond}_{\mathrm{abs}}(L_f, A, E) &\geq \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\leq\epsilon} \frac{\|L_f(A+\Delta A, E) - L_f(A, E)\|}{\epsilon}\\ &= \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\leq\epsilon} \frac{\|L_f^{(2)}(A, E, \Delta A) + o(\|\Delta A\|)\|}{\epsilon}\\ &= \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\leq\epsilon} \|L_f^{(2)}(A, E, \Delta A/\epsilon)\|\\ &= \max_{\|\Delta A\|=1} \|L_f^{(2)}(A, E, \Delta A)\|. \qquad \square\end{aligned} \tag{5.2.5}$$

Next, we derive an upper bound.

Lemma 5.2.3. *The absolute condition number of the Fréchet derivative satisfies*

$$\mathrm{cond}_{\mathrm{abs}}(L_f, A, E) \leq \max_{\|\Delta A\|=1} \|L_f^{(2)}(A, E, \Delta A)\| + \mathrm{cond}_{\mathrm{abs}}(f, A).$$

Proof. Notice that by linearity of the second argument of L_f,

$$\begin{aligned}
\operatorname{cond}_{\text{abs}}(L_f, A, E) &= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \frac{\|L_f(A+\Delta A, E+\Delta E) - L_f(A,E)\|}{\epsilon} \\
&\le \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \left(\frac{\|L_f(A+\Delta A, E) - L_f(A,E)\|}{\epsilon} \right. \\
&\qquad \left. + \frac{\|L_f(A+\Delta A, \Delta E)\|}{\epsilon} \right) \\
&\le \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\le\epsilon} \frac{\|L_f(A+\Delta A, E) - L_f(A,E)\|}{\epsilon} \\
&\qquad + \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \|L_f(A+\Delta A, \Delta E/\epsilon)\|. \qquad (5.2.6)
\end{aligned}$$

The first term on the right-hand side of (5.2.6) is equal to $\max_{\|\Delta A\|=1} \|L_f^{(2)}(A, E, \Delta A)\|$ by (5.2.5). For the second half of the bound (5.2.6) we have, using (5.1.3) and the fact that $L_f^{(2)}(A, E_1, E_2)$ is linear in E_2,

$$\begin{aligned}
\lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \|L_f(A+\Delta A, \Delta E/\epsilon)\| &= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \big\| L_f(A, \Delta E/\epsilon) + L_f^{(2)}(A, \Delta E/\epsilon, \Delta A) \\
&\qquad + o(\|\Delta A\|) \big\| \\
&= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon \\ \|\Delta E\|\le\epsilon}} \|L_f(A, \Delta E/\epsilon) + O(\epsilon)\| \\
&= \lim_{\epsilon\to 0} \sup_{\|\Delta E\|\le\epsilon} \|L_f(A, \Delta E/\epsilon)\| = \operatorname{cond}_{\text{abs}}(f, A).
\end{aligned}$$

Combining the two halves of the bound gives the result. ∎

We now give the corresponding bounds for the relative condition number.

Theorem 5.2.4. *The relative condition number of the Fréchet derivative L_f satisfies* $\operatorname{cond}_{\text{rel}}(L_f, A, E) \ge 1$ *and*

$$\max(\operatorname{cond}_{\text{abs}}(f, A), sM)r \le \operatorname{cond}_{\text{rel}}(L_f, A, E) \le (\operatorname{cond}_{\text{abs}}(f, A) + sM)r,$$

where $s = \|A\|/\|E\|$, $r = \|E\|/\|L_f(A, E)\|$, and $M = \max_{\|\Delta A\|=1} \|L_f^{(2)}(A, E, \Delta A)\|$.

Proof. To show $\operatorname{cond}_{\text{rel}}(L_f, A, E) \ge 1$ we can use Lemmas 5.2.1 and 5.2.2, along with

the linearity of $L_f(A, E)$ in E, as follows:

$$\begin{aligned}
\mathrm{cond}_{\mathrm{rel}}(L_f, A, E) &= \frac{\mathrm{cond}_{\mathrm{abs}}(L_f, A, sE)\|E\|}{\|L_f(A, E)\|} \\
&\geq \frac{\mathrm{cond}_{\mathrm{abs}}(f, A)\|E\|}{\|L_f(A, E)\|} \\
&= \frac{\max_{\|Z\|=1} \|L_f(A, Z)\|\|E\|}{\|L_f(A, E)\|} \\
&= \frac{\max_{\|Z\|=1} \|L_f(A, Z)\|}{\|L_f(A, E/\|E\|)\|} \geq 1.
\end{aligned}$$

For the other inequalities apply Lemma 5.2.1 to Lemmas 5.2.2 and 5.2.3 similarly. □

Theorem 5.2.4 gives upper and lower bounds for $\mathrm{cond}_{\mathrm{rel}}(L_f, A, E)$ that differ by at most a factor 2. During our numerical experiments in section 5.5 we found that typically $\mathrm{cond}_{\mathrm{abs}}(f, A)$ and sM were of comparable size, though on occasion they differed by many orders of magnitude. Finding sufficient conditions for these two quantities to differ significantly remains an open question which will depend on the complex interaction between f, A, and E.

There are already efficient algorithms for estimating $\mathrm{cond}_{\mathrm{abs}}(f, A)$ using Fréchet derivatives and norm estimation techniques using the procedure outlined in section 1.3 (for example [4], [5], [61], and Algorithms 3.5.1 and 3.6.1). The key question is therefore how to estimate the maximum of $\|L_f^{(2)}(A, E, \Delta A)\|$ over all ΔA with $\|\Delta A\| = 1$. This is the subject of the next section.

5.3 Maximizing the second Fréchet derivative

Our techniques for estimating the required maximum norm of the second Fréchet derivative are analogous to those for estimating $\mathrm{cond}_{\mathrm{abs}}(f, A)$, described in section 1.3.

To briefly recall, we usually estimate $\mathrm{cond}_{\mathrm{abs}}(f, A)$ in the 1-norm by $\|K_f(A)\|_1$, which we know is accurate to within a factor of n by Lemma 1.3.1. To estimate $\|K_f(A)\|_1$ the block 1-norm power method is used (see Algorithm 1.3.2). This approach requires around $4t$ matrix–vector products in total (using both $K_f(A)$ and $K_f(A)^*$) and produces estimates rarely more than a factor 3 from the true norm. The parameter t is usually set to 2, but can be increased for greater accuracy at the cost of extra flops.

Using (5.1.6) we obtain a result similar to (1.3.5) for maximizing the norm of the second Fréchet derivative in the Frobenius norm:

$$\begin{aligned}\max_{\|\Delta A\|_F=1} \|L_f^{(2)}(A,E,\Delta A)\|_F &= \sup_{\|\operatorname{vec}(\Delta A)\|_2\le 1} \|\operatorname{vec}(L_f^{(2)}(A,E,\Delta A))\|_2 \\ &= \sup_{\|\operatorname{vec}(\Delta A)\|_2\le 1} \|\big(\operatorname{vec}(E)^T \otimes I_{n^2}\big) K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_2 \\ &= \|\big(\operatorname{vec}(E)^T \otimes I_{n^2}\big) K_f^{(2)}(A)\|_2. \end{aligned} \tag{5.3.1}$$

The next result shows that using the 1-norm instead gives the same accuracy guarantees as Lemma 1.3.1.

Theorem 5.3.1. *The* 1*-norm of the second Fréchet derivative and the* 1*-norm of the second Kronecker form are related by*

$$\frac{1}{n}M \le \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\|_1 \le nM,$$

where $M = \max_{\|\Delta A\|_1\le 1} \|L_f^{(2)}(A,E,\Delta A)\|_1$.

Proof. Making use of (5.1.6), for the lower bound we have

$$\begin{aligned}\max_{\|\Delta A\|_1\le 1} \|L_f^{(2)}(A,E,\Delta A)\|_1 &\le \sup_{\|\Delta A\|_1\le 1} \|\operatorname{vec}(L_f^{(2)}(A,E,\Delta A))\|_1 \\ &= \sup_{\|\Delta A\|_1\le 1} \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_1 \\ &\le \sup_{\|\operatorname{vec}(\Delta A)\|_1\le n} \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_1 \\ &= n \sup_{\|\operatorname{vec}(\Delta A)\|_1\le 1} \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_1 \\ &= n\|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\|_1. \end{aligned}$$

For the upper bound, using (5.1.6) again,

$$\begin{aligned}\max_{\|\Delta A\|_1\le 1} \|L_f^{(2)}(A,E,\Delta A)\|_1 &\ge \frac{1}{n} \sup_{\|\Delta A\|_1\le 1} \|\operatorname{vec}(L_f^{(2)}(A,E,\Delta A))\|_1 \\ &= \frac{1}{n} \sup_{\|\Delta A\|_1\le 1} \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_1 \\ &\ge \frac{1}{n} \sup_{\|\operatorname{vec}(\Delta A)\|_1\le 1} \|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(\Delta A)\|_1 \\ &= \frac{1}{n}\|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\|_1. \qquad \square \end{aligned}$$

Explicitly computing matrix–vector products with $(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)$ and its conjugate transpose is not feasible, as computing $K_f^{(2)}(A)$ using Algorithm 4.4.2 costs $O(n^7)$ flops. Fortunately we can compute the matrix–vector products implicitly since, by (5.1.6),

$$(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\operatorname{vec}(V) = \operatorname{vec}(L_f^{(2)}(A, E, V)),$$

where the evaluation of the right-hand side costs only $O(n^3)$ flops using (5.1.5). This is analogous to the relation $K_f(A)\operatorname{vec}(V) = \operatorname{vec}(L_f(A, V))$ used in the estimation of $K_f(A)$ in the 1-norm via Algorithm 1.3.2.

Similarly we would like to implicitly compute products with the conjugate transpose $\left[(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\right]^*$ so that the entire 1-norm estimation can be done in $O(n^3)$ flops. To do so we need the following result.

Theorem 5.3.2. *Let f be analytic on an open subset $\mathcal{D}$ of $\mathbb{C}$ for which each connected component is closed under conjugation and let f satisfy $f(z) = \overline{f(\overline{z})}$ for all $z \in \mathcal{D}$. Then for all $k \le m$ and A with spectrum in $\mathcal{D}$,*

$$L_f^{(k)}(A, E_1, \dots, E_k)^* = L_f^{(k)}(A^*, E_1^*, \dots, E_k^*).$$

Proof. Our proof is by induction on k, where the base case $k = 1$ is established by Higham and Lin [61, Lem. 6.2]. Assume that the result holds for the kth Fréchet derivative, which exists under the given assumptions. Then, since the Fréchet derivative is equal to the Gâteaux derivative (see section 1.2),

$$L_f^{(k+1)}(A, E_1, \dots, E_{k+1})^* = \left.\frac{d}{dt}\right|_{t=0} L_f^{(k)}(A + tE_{k+1}, E_1, \dots, E_k)^*.$$

Using the inductive hypothesis the right-hand side becomes

$$\left.\frac{d}{dt}\right|_{t=0} L_f^{(k)}(A^* + tE_{k+1}^*, E_1^*, \dots, E_k^*) = L_f^{(k+1)}(A^*, E_1^*, \dots, E_{k+1}^*). \qquad \square$$

The conditions of Theorem 5.3.2 are not very restrictive; they are satisfied by the exponential, the logarithm, real powers A^t ($t \in \mathbb{R}$), the matrix sign function, and trigonometric functions, for example. The condition $f(z) = \overline{f(\overline{z})}$ is, in fact, equivalent to $f(A)^* = f(A^*)$ for all A with spectrum in $\mathcal{D}$ [62, Thm. 3.2 and its proof]. Under the conditions of the theorem it can be shown that

$$K_f(A)^* = K_f(A^*), \tag{5.3.2}$$

which is implicit in [57, pp. 66–67] and [61], albeit not explicitly stated there (and this equality will be needed in section 5.6). As mentioned in section 1.3, matrix–vector products with $K_f(A)^*$ can therefore be computed efficiently since

$$K_f(A)^* \operatorname{vec}(V) = K_f(A^*) \operatorname{vec}(V) = \operatorname{vec}(L_f(A^*, V)) = \operatorname{vec}(L_f(A, V^*)^*), \tag{5.3.3}$$

using Theorem 5.3.2 for the last equality. The next result gives an analog of (5.3.2) for $\left[(\operatorname{vec}(E)^T \otimes I_{n^2}) K_f^{(2)}(A)\right]^*$.

Theorem 5.3.3. *Under the conditions of Theorem 5.3.2, for $A \in \mathbb{C}^{n\times n}$ with spectrum in $\mathcal{D}$,*

$$\left[(\operatorname{vec}(E)^T \otimes I_{n^2}) K_f^{(2)}(A)\right]^* = (\operatorname{vec}(E^*)^T \otimes I_{n^2}) K_f^{(2)}(A^*).$$

Proof. We will need to use the Kronecker product property

$$(A \otimes B)(C \otimes D) = AC \otimes BD. \tag{5.3.4}$$

We also need the commutation (or vec-permutation) matrix $C_n \in \mathbb{C}^{n^2\times n^2}$, which is a permutation matrix defined by the property that for $A \in \mathbb{C}^{n\times n}$, $\operatorname{vec}(A^T) = C_n \operatorname{vec}(A)$. It is symmetric and satisfies, for $A, B \in \mathbb{C}^{n\times n}$ and $x, y \in \mathbb{C}^n$, [47], [78, Thm. 3.1]

$$(A \otimes B)C_n = C_n(B \otimes A), \tag{5.3.5}$$

$$(x^T \otimes y^T)C_n = y^T \otimes x^T. \tag{5.3.6}$$

We will prove that the two matrices in the theorem statement are equal by showing that they take the same value when multiplied by the arbitrary vector $v = \operatorname{vec}(V)$, where $V \in \mathbb{C}^{n\times n}$. Multiplying both sides by v and taking vec of the right-hand side we find that we need to show

$$\left[(\operatorname{vec}(E)^T \otimes I_{n^2}) K_f^{(2)}(A)\right]^* v = (v^T \otimes \operatorname{vec}(E^*)^T \otimes I_{n^2}) \operatorname{vec}(K_f^{(2)}(A^*)).$$

Manipulating the left-hand side we have

$$\begin{aligned}
\left[(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\right]^* v &= K_f^{(2)}(A)^* \left(\operatorname{vec}(\overline{E}) \otimes I_{n^2}\right) v \\
&= K_f^{(2)}(A)^* \left(\operatorname{vec}(\overline{E}) \otimes v\right) \quad \text{using } v = 1 \otimes v \text{ and (5.3.4)} \\
&= \left[\left(\operatorname{vec}(\overline{E}) \otimes v\right)^T \otimes I_{n^2}\right] \operatorname{vec}(K_f^{(2)}(A)^*) \quad \text{by (5.1.2)} \\
&= \left[((C_n \otimes I_{n^2})(\operatorname{vec}(E^*) \otimes v))^T \otimes I_{n^2}\right] \operatorname{vec}(K_f^{(2)}(A)^*) \\
&= \left[\left((\operatorname{vec}(E^*)^T \otimes v^T)(C_n \otimes I_{n^2})\right) \otimes I_{n^2}\right] \operatorname{vec}(K_f^{(2)}(A)^*) \quad \text{using } C_n = C_n^T \\
&= (\operatorname{vec}(E^*)^T \otimes v^T \otimes I_{n^2})(C_n \otimes I_{n^4}) \operatorname{vec}(K_f^{(2)}(A)^*) \quad \text{by (5.3.4) and } I_{n^2} \otimes I_{n^2} = I_{n^4} \\
&= (v^T \otimes \operatorname{vec}(E^*)^T \otimes I_{n^2})(C_{n^2} \otimes I_{n^2})(C_n \otimes I_{n^4}) \operatorname{vec}(K_f^{(2)}(A)^*),
\end{aligned}$$

using (5.3.6) for the last equality. Therefore it remains to show that

$$(C_{n^2} \otimes I_{n^2})(C_n \otimes I_{n^4}) \operatorname{vec}(K_f^{(2)}(A)^*) = \operatorname{vec}(K_f^{(2)}(A^*)),$$

a proof of which can be found in section 5.6. □

Theorem 5.3.3 shows that we can compute matrix–vector products with the conjugate transpose as

$$\begin{aligned}
\left[(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\right]^* \operatorname{vec}(V) &= (\operatorname{vec}(E^*)^T \otimes I_{n^2})K_f^{(2)}(A^*) \operatorname{vec}(V) \\
&= \operatorname{vec}(L_f^{(2)}(A^*, E^*, V)) \quad \text{by (5.1.6)} \\
&= \operatorname{vec}(L_f^{(2)}(A, E, V^*)^*), \qquad (5.3.7)
\end{aligned}$$

where the final equality is from Theorem 5.3.2. Therefore the block 1-norm estimator can used to estimate efficiently $\|(\operatorname{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\|_1$ in Theorem 5.3.1.

5.4 An algorithm for estimating the relative condition number

We are now ready to state our complete algorithm for estimating the relative condition number of a Fréchet derivative in the 1-norm.

In the following algorithm we use the unvec operator, which for a vector $v \in \mathbb{C}^{n^2}$ returns the unique matrix in $\mathbb{C}^{n \times n}$ such that $\operatorname{vec}(\operatorname{unvec}(v)) = v$.

Algorithm 5.4.1. *Given $A \in \mathbb{C}^{n\times n}$, $E \in \mathbb{C}^{n\times n}$, and f satisfying the conditions of Theorem 5.3.2 this algorithm produces an estimate γ of the relative condition number* $\mathrm{cond}_{\mathrm{rel}}(L_f, A, E)$. *It uses the block* 1*-norm estimation algorithm of* [66] *with* $t = 2$*, which we denote by normest* (*an implementation is* [51, `funm_condest1`]).

1 Compute $f(A)$ and $L_f(A, E)$ via specialized algorithms such as those in [4], [61], or chapter 3 if possible. Alternatively, compute $L_f(A, E)$ by finite differences, the complex step method [5], or (5.1.4).
2 Compute an estimate c of $\mathrm{cond}_{\mathrm{rel}}(f, A)$ using Algorithm 1.3.2 and normest.
3 $c \leftarrow c\|f(A)\|_1/\|A\|_1$ % Now $c = \mathrm{cond}_{\mathrm{abs}}(f, A)$.
4 $s = \|A\|_1/\|E\|_1$
5 Estimate $\mu = \| \left(\mathrm{vec}(E)^T \otimes I_{n^2}\right) K_f^{(2)}(A)\|_1$ using normest with lines 7–14.
6 $\gamma = (c + s\mu)\|E\|_1/\|L_f(A, E)\|_1$
7 ... To compute $(\mathrm{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)v$ for a given v:
8 $V = \mathrm{unvec}(v)$
9 Calculate $W = L_f^{(2)}(A, E, V)$ using (5.1.5) for example.
10 Return $\mathrm{vec}(W)$ to the norm estimator.
11 ... To compute $\left[(\mathrm{vec}(E)^T \otimes I_{n^2})K_f^{(2)}(A)\right]^* v$ for a given v:
12 $V = \mathrm{unvec}(v)$
13 Calculate $W = L_f^{(2)}(A, E, V^*)$ using (5.1.5) for example.
14 Return $\mathrm{vec}(W^*)$ to the norm estimator.

Cost: Around 9 Fréchet derivative evaluations for $L_f(A, E)$ and $\mathrm{cond}_{\mathrm{rel}}(f, A)$, plus about 8 second Fréchet derivative evaluations. The cost depends on which particular methods are chosen to compute the Fréchet derivatives required in lines 1 and 2 and $L_f^{(2)}(A, E, V)$, but the total cost is $O(n^3)$ flops.

The quality of the estimate returned by Algorithm 5.4.1 depends on the quality of the underlying bounds and the quality of the computed norm estimate. The estimate has a factor 2 uncertainty from Theorem 5.2.4 and another factor n uncertainty from Lemma 1.3.1 and Theorem 5.3.1. The norm estimates are usually correct to within a factor 3, so overall we can expect the estimate from Algorithm 5.4.1 to differ from $\mathrm{cond}_{\mathrm{rel}}(f, A)$ by at most a factor $6n$.

Even though the Fréchet derivative $L_f^{(2)}(A, E_1, E_2)$ is linear in E_1 and E_2, the scaling of E_1 and E_2 may affect the accuracy of the computation. Heuristically we might expect that scaling E_1 and E_2 so that $\|A\|_1 \approx \|E_1\|_1 \approx \|E_2\|_1$ would give good accuracy. When implementing Algorithm 5.4.1 we scale E_1 and E_2 in this way before taking the derivatives and rescaling the result.

5.5 Numerical experiments

Our experiments are all performed in MATLAB R2013a. We examine the performance of Algorithm 5.4.1 for the matrix logarithm and matrix powers A^t with $t \in \mathbb{R}$ using the Fréchet derivative evaluation algorithms from chapter 3 and [61], respectively. Throughout this section $u = 2^{-53}$ denotes the unit roundoff. Since the Fréchet derivative algorithms in question have been shown to perform in a forward stable manner in section 3.8.2 and [61] (assessed therein using the `Kronecker` condition number estimator that we will show tends to underestimate the true condition number) we expect their relative errors to be bounded by the condition number times the unit roundoff.

We will compare Algorithm 5.4.1, denoted in this section by `condest_FD`, with three other methods in terms of the accuracy and reliability of using the estimated value of $\text{cond}_{\text{rel}}(L_f, A, E)u$ as a bound on the relative error

$$\frac{\|\widehat{L}_f(A, E) - L_f(A, E)\|_1}{\|L_f(A, E)\|_1},$$

where $\widehat{L}_f(A, E)$ is the computed Fréchet derivative. Unfortunately, we cannot directly assess the quality of our condition number estimates as we have no way to compute the exact condition number $\text{cond}_{\text{rel}}(L_f, A, E)$.

For our tests we need to choose the matrices A and E at which to evaluate the Fréchet derivative and its condition number. For A we use the same test matrices as in section 3.8 and [61]. These (mostly 10×10) matrices are from the Matrix Computation Toolbox [50], the MATLAB `gallery` function, and the literature. Ideally we would choose the direction E as a direction that maximizes the relative error above; however it is unclear how to do so without resorting to expensive optimization procedures. Instead we choose the direction E to be a matrix with normal $(0, 1)$ distributed elements, but we give a specific example of a worst case direction for the matrix logarithm in

section 5.5.3.

To compute an accurate value of $L_f(A,E)$, used solely to calculate the relative errors mentioned above, we evaluate (5.1.4) in 250 digit precision by performing the diagonalization $VDV^{-1} = \left[\begin{smallmatrix} X & E \\ 0 & X \end{smallmatrix}\right]$, applying f to the diagonal matrix D, and returning the $(1,2)$ block. If the matrix $\left[\begin{smallmatrix} X & E \\ 0 & X \end{smallmatrix}\right]$ is not diagonalizable we add a random perturbation of norm 10^{-125} to make the eigenvalues distinct. This idea was introduced by Davis [27] and has been used in chapters 2 and 3, and [61]. These high precision calculations are performed in the Symbolic Math Toolbox.

We compare our algorithm against three approximations. The first is

$$\mathrm{cond}_{\mathrm{rel}}(L_f, A, E) \approx \frac{\|L_f(A+\Delta A, E+\Delta E) - L_f(A,E)\|_1}{\epsilon \|L_f(A,E)\|_1},$$

where ΔA and ΔE are chosen to have normal $(0,1)$ distributed elements and then are scaled so that $\|\Delta A\|_1/\|A\|_1 = \|\Delta E\|_1/\|E\|_1 = \epsilon = 10^{-8}$ (c.f. (5.2.2)). We would expect this method to generally underestimate the condition number since ΔA and ΔE are unlikely to point in the directions of greatest sensitivity. This estimate will be referred to as the `random` method throughout this section. Since this method requires only two Fréchet derivative evaluations (as opposed to around 17 for Algorithm 5.4.1) one possible extension of this method would be to run it k times and take the mean as an estimate of the condition number. Further experiments, not reported here, took $k = 5$, 10, and 20 without seeing any significant change in the results.

Our next alternative approximation is

$$\mathrm{cond}_{\mathrm{rel}}(L_f, A, E) \approx \frac{\|K_f(A+\Delta A) - K_f(A)\|_1}{\epsilon \|K_f(A)\|_1},$$

where $K_f(A)$ is the Kronecker form of the Fréchet derivative in (5.1.1) and ΔA is generated with normal $(0,1)$ distributed elements and then scaled so that $\|\Delta A\|_1/\|A\|_1 = \epsilon = 10^{-8}$. This heuristic approximation has been used in [4] and chapter 3, but has two drawbacks. First, the dependence on E is ignored which (see Lemmas 5.2.2 and 5.2.3) essentially corresponds to neglecting an additive $\mathrm{cond}_{\mathrm{abs}}(f,A)$ term and so could lead to underestimating the condition number. Second, the random direction ΔA will generally not point in the direction in which $K_f(A)$ is most sensitive, again leading to underestimation. We refer to this as the `Kronecker` method in our experiments. This method costs $O(n^5)$ flops and is therefore the most expensive. We might also

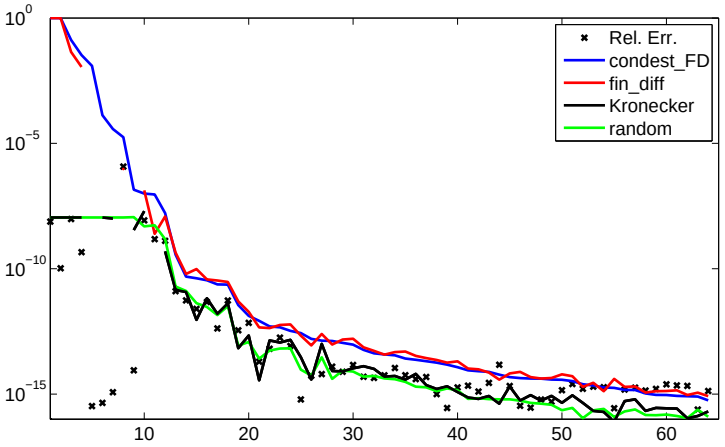

Figure 5.5.1: Relative errors of computed $L_{\log}(A, E)$ and estimates of $\mathrm{cond}_{\mathrm{rel}}(L_{\log}, A, E)u$ for 66 test matrices sorted by decreasing value of `condest_FD`.

try running this method k times and taking the mean of the results, in an attempt to better estimate the condition number. Further experiments averaging $k = 5$, 10, and 20 runs of this algorithm made no significant difference to the results.

The final approximation method for comparison is a modification of Algorithm 5.4.1 that estimates the second Fréchet derivative by the finite difference approximation $L_f^{(2)}(A, E, V) \approx t^{-1}\big(L_f(A + tV, E) - L_f(A, E)\big)$ for a small t instead of using (5.1.5). This is done by invoking `funm_condest1` from the Matrix Function Toolbox [50] on the function $g(A) = L_f(A, E)$ with the option to use finite differences selected, with the default value $t = 10^{-8}$. We will refer to this method as `fin_diff` in our experiments. This method has essentially identical cost to Algorithm 5.4.1, the only difference being the computation of the second Fréchet derivatives.

5.5.1 Condition number of Fréchet derivative of matrix logarithm

In our first experiment we compute the Fréchet derivative of the logarithm of 66 test matrices using the Algorithms 3.5.1 and 3.6.1, depending on whether the input matrix is complex or real, respectively. Figure 5.5.1 shows the normwise relative errors and the estimates of $\mathrm{cond}_{\mathrm{rel}}(L_{\log}, A, E)u$.

We see that `fin_diff` and `condest_FD` give similar output in most cases, as do `Kronecker` and `random`, though neither of these latter two seems able to yield values

higher than 10^{-8} (the length of the finite difference step used in the algorithm). All four methods agree on which problems are well conditioned. On the right-hand side of the figure we see that some relative errors are slightly above the estimates. However all are within a factor 2.7 of the estimate from `condest_FD`, which is much less than the factor $6n$ we can expect in the worst case, as explained at the end of section 5.4.

For the ill conditioned problems both `Kronecker` and `fin_diff` fail to return condition number estimates for some of the test matrices, as indicated by the broken lines at the left end of Figure 5.5.1. This is due to a perturbed matrix $A+V$ having negative eigenvalues during the computation of the Fréchet derivatives using finite differences, which raises an error since the principal matrix logarithm and its Fréchet derivative are not defined for such matrices. In principal this same problem could happen when using the `random` method. Since `condest_FD` computes bounds on the second Fréchet derivative without perturbing A it does not encounter this problem. In section 5.5.3 we analyze the second test matrix in more detail and find that, despite the error bound being pessimistic, the condition number truly is as large as estimated by `fin_diff` and `condest_FD`.

5.5.2 Condition number of Fréchet derivative of matrix power

Our second experiment compares the algorithms on the function A^t with $t = 1/15$ over 60 test matrices from the previous set, where the Fréchet derivative is computed using the algorithm of Higham and Lin [61]. Figure 5.5.2 shows the normwise relative errors and the estimated quantities $\mathrm{cond}(L_{x^t}, A, E)u$, sorted by decreasing `condest_FD`. Again we see that the condition number estimates from `Kronecker` and `random` are bounded above by about 10^{-8}, though the actual relative errors are sometimes much higher.

The methods return similar condition number estimates for the well conditioned problems but give very different results on the ill conditioned problems in the first 10 test cases. In particular `fin_diff`, `Kronecker`, and `random` do not provide reliable error bounds for the badly conditioned cases, their bounds being several orders of magnitude lower than the observed relative errors for test matrices 6 and 9. There is also some significant overestimation by `fin_diff` on test matrix 8. In contrast, `condest_FD` provides reliable error bounds for all the ill conditioned problems.

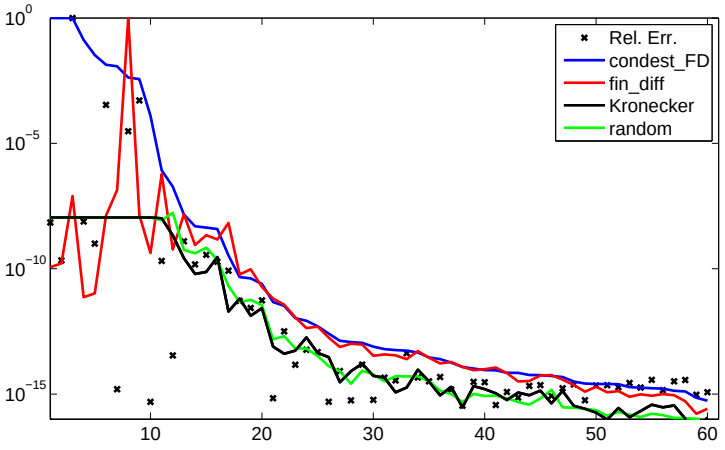

Figure 5.5.2: Relative errors of computed $L_{x^t}(A,E)$ and estimates of $\mathrm{cond}_{\mathrm{rel}}(L_{x^t},A,E)u$, with $t = 1/15$, for 60 test matrices sorted by decreasing value of `condest_FD`.

Similar experiments with the matrix exponential, not reported here, show analogous results: both `condest_FD` and `fin_diff` give good bounds on the relative errors whilst `Kronecker` and `random` generally underestimate them. The only difference is that `fin_diff` also gives good bounds for the ill-conditioned problems, instead of failing or giving spurious results as above.

5.5.3 An ill conditioned Fréchet derivative

In this section we give a more detailed analysis of the Fréchet derivative of the logarithm on test problem 2 of Figure 5.5.1. The matrices A and E are

$$A = \begin{bmatrix} e^{(\pi - 10^{-7}i)} & 1000 \\ 0 & e^{(\pi + 10^{-7}i)} \end{bmatrix}, \quad E = \begin{bmatrix} 0.3 & 0.012 \\ -0.76 & -0.49 \end{bmatrix}.$$

This example is particularly interesting because the condition number estimated by Algorithm 5.4.1 is large, $\mathrm{cond}_{\mathrm{rel}}(L_{\log},A,E) \approx 1.5 \times 10^{20}$, but we observed a relative error of around 10^{-10} when computing the Fréchet derivative in our experiments. We will show that a tiny perturbation to A that greatly changes $L_{\log}(A,E)$ exists.

What we need to do is to find a matrix V with $\|V\|_1 = 1$ such that $\|L^{(2)}_{\log}(A,E,V)\|_1$ is large, since by Theorem 5.2.4 this will imply that $\mathrm{cond}_{\mathrm{rel}}(L_{\log},A,E)$ is large. Such a V can be obtained as output from the 1-norm estimator. However, we will obtain it from first principles by applying direct search optimization [53], with the code `mdsmax`

from [50] that implements the algorithm from [102], [103]. Direct search yields the putative optimal point

$$V = \begin{bmatrix} 0.1535 + 0.1535i & 0.1535 + 0.1535i \\ 0.1535 + 0.7677i & 0.1535 + 0.1535i \end{bmatrix},$$

shown to four significant figures, for which $\|L_{\log}^{(2)}(A, E, V)\|_1 = 1.4 \times 10^{44}$. Calculating the Fréchet derivatives $L_{\log}(A, E)$ and $L_{\log}(A + uV, E)$ in 250 digit arithmetic—using the procedure outlined at the beginning of this section—leads to a relative difference of

$$\frac{\|L_{\log}(A + uV, E) - L_{\log}(A, E)\|_1}{\|L_{\log}(A, E)\|_1} = 1.0318,$$

showing that the Fréchet derivative evaluation is extremely sensitive to perturbations in the direction V. We were fortunate not to experience this sensitivity during the evaluation of $L_{\log}(A, E)$. This computation confirms that, as `condest_FD` suggests, a relative perturbation of order u to A can produce a change of order 1 in the Fréchet derivative. But as we saw in the experiments, ill conditioning is not identified consistently by the approximations from `fin_diff`, `Kronecker`, or `random`.

5.6 Continued proof of Theorem 5.3.3

This section completes the proof of Theorem 5.3.3. We need to show that

$$(C_{n^2} \otimes I_{n^2})(C_n \otimes I_{n^4}) \operatorname{vec}(K_f^{(2)}(A)^*) = \operatorname{vec}(K_f^{(2)}(A^*)).$$

We will begin by showing that $\operatorname{vec}(K_f^{(2)}(A)^*) = \operatorname{vec}(K_f^{(2)}(A^*)C_n)$ which (after some manipulation) reduces the problem to showing that

$$(C_{n^2} \otimes I_{n^2}) \operatorname{vec}(K_f^{(2)}(A^*)) = \operatorname{vec}(K_f^{(2)}(A^*)). \tag{5.6.1}$$

Before proceeding we recall that C_n is a permutation matrix corresponding to some permutation σ on the integers from 1 to n^2. This permutation can be defined by the property that when $\operatorname{vec}(E_i) = e_i$ is the ith standard basis vector then

$$E_{\sigma(i)} = E_i^T, \tag{5.6.2}$$

which follows from the observation that $C_n \operatorname{vec}(E_i) = C_n e_i = e_{\sigma(i)} = \operatorname{vec}(E_{\sigma(i)})$ along with $C_n \operatorname{vec}(E_i) = \operatorname{vec}(E_i^T)$.

Expanding Algorithm 4.4.2 for the case $k = 2$, (or from (4.4.3)), we see that $K_f^{(2)}(X) \in \mathbb{C}^{n^4 \times n^2}$ is made from $n^2 \times 1$ blocks

$$\left[K_f^{(2)}(X)\right]_{ij} = \operatorname{vec}(L_f^{(2)}(X, E_j, E_i)), \qquad i, j = 1 : n^2,$$

so that applying C_n to the right of $K_f^{(2)}(A^*)$ permutes its columns and

$$\begin{aligned}\left[K_f^{(2)}(A^*)C_n\right]_{ij} &= \operatorname{vec}(L_f^{(2)}(A^*, E_{\sigma(j)}, E_i)) \\ &= \operatorname{vec}(L_f^{(2)}(A^*, E_j^T, E_i)) \\ &= \operatorname{vec}(L_f^{(2)}(A, E_j, E_i^T)^*),\end{aligned}$$

because $L_f^{(2)}(A^*, F, G) = L_f^{(2)}(A, F^*, G^*)^*$ by Theorem 5.3.2. Similarly $K_f^{(2)}(A)^*$ is made from $1 \times n^2$ blocks

$$\left[K_f^{(2)}(A)^*\right]_{ij} = \operatorname{vec}(L_f^{(2)}(A, E_i, E_j))^*, \qquad i, j = 1 : n^2.$$

To continue, note that $K_f^{(2)}(A^*)C_n$ and $K_f^{(2)}(A)^*$ are of sizes $n^4 \times n^2$ and $n^2 \times n^4$ respectively and so cannot be equal, though we only need to prove that their vectorizations are equal. We need to show that each $n^2 \times n^2$ block column of $K_f^{(2)}(A)^*$ is equal to the "unvec" of the corresponding $n^4 \times 1$ column of $K_f^{(2)}(A^*)C_n$. That is for $j = 1 : n^2$ we want to show that

$$\begin{bmatrix} \operatorname{vec}(L_f^{(2)}(A, E_1, E_j))^* \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(A, E_{n^2}, E_j))^* \end{bmatrix} = \begin{bmatrix} \operatorname{vec}(L_f^{(2)}(A, E_j, E_1^T)^*) & \cdots & \operatorname{vec}(L_f^{(2)}(A, E_j, E_{n^2}^T)^*) \end{bmatrix}. \tag{5.6.3}$$

To do so, we will expand the rows and columns then show they are equal elementwise. Since, as explained in chapter 4,

$$L_f^{(2)}(A, E_k, E_j) = \left.\frac{d}{dt}\right|_{t=0} L_f(A(t), E_k), \qquad A(t) = A + tE_j,$$

the left-hand side of (5.6.3) can be written as

$$\begin{bmatrix} \operatorname{vec}(L_f^{(2)}(A, E_1, E_j))^* \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(A, E_{n^2}, E_j))^* \end{bmatrix} = \left.\frac{d}{dt}\right|_{t=0} \begin{bmatrix} e_1^T\overline{\operatorname{vec}(L_f(A(t), E_1))} & \cdots & e_{n^2}^T\overline{\operatorname{vec}(L_f(A(t), E_1))} \\ \vdots & \ddots & \vdots \\ e_1^T\overline{\operatorname{vec}(L_f(A(t), E_{n^2}))} & \cdots & e_{n^2}^T\overline{\operatorname{vec}(L_f(A(t), E_{n^2}))} \end{bmatrix}.$$

Similarly using (5.6.2) on the right-hand side of (5.6.3) we have

$$\operatorname{vec}(L_f^{(2)}(A, E_j, E_i^T)^*) = \left.\frac{d}{dt}\right|_{t=0} C_n\overline{\operatorname{vec}(L_f(A(t), E_{\sigma(i)}))},$$

and therefore the right-hand side of (5.6.3) can be written as

$$\begin{bmatrix}\operatorname{vec}(L_f^{(2)}(A,E_j,E_1^T)^*) & \cdots & \operatorname{vec}(L_f^{(2)}(A,E_j,E_{n^2}^T)^*)\end{bmatrix}$$
$$= \frac{d}{dt}\Big|_{t=0} \begin{bmatrix} e_{\sigma(1)}^T\overline{\operatorname{vec}(L_f(A(t),E_{\sigma(1)}))} & \cdots & e_{\sigma(1)}^T\overline{\operatorname{vec}(L_f(A(t),E_{\sigma(n^2)}))} \\ \vdots & \ddots & \vdots \\ e_{\sigma(n^2)}^T\overline{\operatorname{vec}(L_f(A(t),E_{\sigma(1)}))} & \cdots & e_{\sigma(n^2)}^T\overline{\operatorname{vec}(L_f(A(t),E_{\sigma(n^2)}))} \end{bmatrix}.$$

Suppressing the dependence on t, we need to prove that

$$e_j^T \operatorname{vec}(L_f(A,E_i)) = e_{\sigma(i)}^T \operatorname{vec}(L_f(A,E_{\sigma(j)})),$$

since these are the (i,j) elements of the left and right-hand side of equation (5.6.3) respectively (with the complex conjugation removed from both sides). Beginning from the right-hand side we have

$$\begin{aligned} e_{\sigma(i)}^T \operatorname{vec}(L_f(A,E_{\sigma(j)})) &= e_i^T C_n \operatorname{vec}(L_f(A,E_{\sigma(j)})) \\ &= e_i^T\overline{\operatorname{vec}(L_f(A^*,E_j))} \quad \text{by (5.6.2)} \\ &= e_i^T(e_j^T\otimes I_{n^2})\overline{\operatorname{vec}(K_f(A^*))} \quad \text{by (5.1.1)} \\ &= e_i^T(e_j^T\otimes I_{n^2})\overline{\operatorname{vec}(K_f(A)^*)} \quad \text{by (5.3.2)} \\ &= e_i^T(e_j^T\otimes I_{n^2})C_n \operatorname{vec}(K_f(A)) \\ &= e_i^T(I_{n^2}\otimes e_j^T)\operatorname{vec}(K_f(A)) \quad \text{by (5.3.6)} \\ &= e_j^T(e_i^T\otimes I_{n^2})\operatorname{vec}(K_f(A)) \\ &= e_j^T \operatorname{vec}(L_f(A,E_i)) \quad \text{by (5.1.1)}, \end{aligned}$$

as required, which completes the proof of

$$\operatorname{vec}(K_f^{(2)}(A)^*) = \operatorname{vec}(K_f^{(2)}(A^*)C_n).$$

To complete the result we need to prove (5.6.1). To make the notation slightly easier we will use $X = A^*$ from now on. By [78, Thm. 3.1(i)] we can write

$$C_{n^2} = \sum_{j=1}^{n^2} e_j^T \otimes I_{n^2} \otimes e_j,$$

where $e_k \in \mathbb{C}^{n^2}$, and so the left-hand side of (5.6.1) becomes

$$\begin{aligned}
(C_{n^2} \otimes I_{n^2}) \operatorname{vec}(K_f^{(2)}(X)) &= \left(\sum_{j=1}^{n^2} e_j^T \otimes I_{n^2} \otimes e_j \otimes I_{n^2} \right) \operatorname{vec}(K_f^{(2)}(X)) \\
&= \sum_{j=1}^{n^2} \operatorname{vec}\left((I_{n^2} \otimes e_j \otimes I_{n^2}) K_f^{(2)}(X) e_j \right) \\
&= \sum_{j=1}^{n^2} \operatorname{vec}\left((I_{n^2} \otimes e_j \otimes I_{n^2}) \operatorname{vec}(K_f^{(1)}(X, E_j)) \right) \\
&= \sum_{j=1}^{n^2} \operatorname{vec}\left((e_j \otimes I_{n^2}) K_f^{(1)}(X, E_j) \right) \\
&= \sum_{j=1}^{n^2} \operatorname{vec}\left(e_j \otimes K_f^{(1)}(X, E_j) \right) \\
&= \operatorname{vec}\left(\begin{bmatrix} K_f^{(1)}(X, E_1) \\ \vdots \\ K_f^{(1)}(X, E_{n^2}) \end{bmatrix} \right),
\end{aligned}$$

where $K_f^{(1)}(X, E_i)$ is defined in section 4.4. To show that this is equal to $\operatorname{vec}(K_f^{(2)}(X))$ we can write the two vectors out elementwise. For $\operatorname{vec}(K_f^{(2)}(X))$ we know from Algorithm 4.4.2 that

$$\operatorname{vec}(K_f^{(2)}(X)) = \begin{bmatrix} \operatorname{vec}(L_f^{(2)}(X, E_1, E_1)) \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(X, E_1, E_{n^2})) \\ \operatorname{vec}(L_f^{(2)}(X, E_2, E_1)) \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(X, E_{n^2}, E_{n^2})) \end{bmatrix}, \tag{5.6.4}$$

whereas

$$\operatorname{vec}\left(\begin{bmatrix} K_f^{(1)}(X,E_1) \\ \vdots \\ K_f^{(1)}(X,E_{n^2}) \end{bmatrix}\right) = \operatorname{vec}\left(\begin{bmatrix} K_f^{(1)}(X,E_1)e_1 \\ K_f^{(1)}(X,E_2)e_1 \\ \vdots \\ K_f^{(1)}(X,E_1)e_2 \\ \vdots \\ K_f^{(1)}(X,E_{n^2})e_{n^2} \end{bmatrix}\right)$$

$$= \operatorname{vec}\left(\begin{bmatrix} \operatorname{vec}(L_f^{(2)}(X,E_1,E_1)) \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(X,E_{n^2},E_1)) \\ \operatorname{vec}(L_f^{(2)}(X,E_1,E_2)) \\ \vdots \\ \operatorname{vec}(L_f^{(2)}(X,E_{n^2},E_{n^2})) \end{bmatrix}\right).$$

This is equal to (5.6.4) since $L_f^{(2)}(X,F,G) = L_f^{(2)}(X,G,F)$, by the ordering independence noted in section 5.1.

Chapter 6

Conclusions

In this final chapter we provide a summary of the material contained in the previous chapters and identify a number of open problems for future research.

Our new algorithms in chapter 2 for computing the matrix sine and cosine—both separately and together—are backward stable in exact arithmetic, thereby providing a more rigorous foundation than for previous algorithms, all of which are based on bounding absolute or forward errors of the function of a scaled matrix. Algorithms with this form of backward stability are already available for the matrix exponential and matrix logarithm. A key finding is that Padé approximants of the matrix cosine do not lend themselves to deriving backward stable algorithms, while those for the matrix sine put strong constraints on the spectral radius of the matrix. We therefore introduced new rational approximants obtained from Padé approximants of the exponential, which yield backward stable approximants of the sine and cosine with no a priori limit on the spectral radius. We also gave the first multiple angle formula-based algorithm for the matrix sine, which uses the triple angle formula in order to avoid the cosines that would be needed by the double angle formula. Experiments show that the new algorithms behave in a forward stable manner in floating point arithmetic, have better backward stability properties than their competitors, and are especially effective for triangular matrices.

A remaining open question is why the second double angle formula in (2.6.1) performs better in floating point arithmetic than the first in Algorithm 2.6.2 for simultaneous computation of the sine and cosine.

In chapter 3 we extended the complex Schur form-based inverse scaling and squaring algorithm of Al-Mohy and Higham [6] for computing the matrix logarithm in two ways. First, Algorithm 3.6.1 extends the algorithm to work entirely in real arithmetic for real matrices. It has the advantages over the original version of being twice as fast, requiring less intermediate storage, and yielding generally more accurate results.

Second, Algorithm 3.5.1 extends the algorithm of [6] to compute one or more Fréchet derivatives after computing $\log(A)$, with reuse of information, while Algorithm 3.6.1 does the same but working in real arithmetic for real data. We have shown that the new algorithms for $L_{\log}(A, E)$ are significantly less expensive than existing algorithms (see section 3.7) and are also more accurate in practice (see section 3.8.2).

The fact that our choice of the algorithmic parameters m and s is based on $\alpha_p(A)$, while our backward error bounds for the Fréchet derivative involve the potentially much larger quantity $\|A\|$, does not appear to affect the accuracy of the Fréchet derivative computations: in our experiments the Fréchet derivatives were computed in a forward stable way throughout.

By combining the new algorithms with the block 1-norm estimation algorithm of Higham and Tisseur [66] reliable condition estimates are obtained, whereas we have shown that a general purpose $\mathrm{cond}_{\mathrm{rel}}(f, A)$ estimate based on finite differences can greatly underestimate the condition number (see section 3.8.3).

In chapter 4 we derived sufficient conditions for the existence and continuity of higher order Fréchet derivatives of matrix functions as well as methods for computing the kth Fréchet derivative and its associated Kronecker form. These lay the foundations for further investigation of higher order Fréchet derivatives and their use in applications.

We have also investigated the level-2 condition number for matrix functions, showing that in a number of cases the level-2 condition number can be related to the level-1 condition number, through equality, a functional relationship, or a bound. It is an interesting open question whether stronger results can be proved, but our numerical experiments give some indication that this may be possible.

In chapter 5 we defined, for the first time, the condition number of the Fréchet derivative of a matrix function and derived an algorithm for estimating it (Algorithm 5.4.1) that applies to a wide class of functions containing the exponential, the

logarithm, and real matrix powers. In practice, the algorithm produces estimates within a factor $6n$ of the true 1-norm condition number at a cost of $O(n^3)$ flops, given $O(n^3)$ flops algorithms for computing the function and its Fréchet derivative. The norms being estimated by the algorithm involve $n^4 \times n^2$ matrices, so structure is being exploited. An interesting open question is whether the highly structured nature of the second Fréchet derivative and its associated Kronecker form can be exploited to gain further theoretical insight into the conditioning of the Fréchet derivative.

This latter algorithm is particularly useful for testing the forward stability of algorithms for computing Fréchet derivatives, and for this purpose our experiments show it to be much more reliable than a heuristic estimate used previously.

More generally, there are numerous directions to be explored in future research. Our new results regarding the existence and computation of higher order Fréchet derivatives and their Kronecker forms enable the design of new algorithms for nonlinear matrix problems. For example, second order Fréchet derivatives of matrix functions have recently been used to analyze spatial transformations in computer vision and image analysis [109]. There is also a generalized Halley method utilizing higher order Fréchet derivatives to solve nonlinear equations in Banach space [9, Sec. 3], which can compute the required derivatives using Algorithm 4.3.7. To further facilitate the development of applications using higher order Fréchet derivatives it will be important to design more efficient algorithms than Algorithms 4.3.7 and 4.4.2 for their computation.

It will also be beneficial to compute multiple Fréchet derivatives in parallel, making efficient use of modern computer architectures. Such a parallel algorithm would, for example, dramatically increase the speed of condition number estimation (as described in section 1.3) since $t \geq 2$ Fréchet derivatives are required at each step of the algorithm.

Finally, we might also investigate elementwise and mixed condition numbers of a matrix function, as opposed to the normwise condition numbers considered here. This would, for example, allow us to find the elements of A which, under a small perturbation, cause the largest normwise change to $f(A)$. Such condition numbers are particularly interesting when the elements of A have some physical meaning. For example, in nuclear burnup calculations the elements of A are coefficients of different chemical reactions [91].

Bibliography

[1] Selin D. Ahipasaoglu, Xiaobo Li, and Karthik Natarajan. A convex optimization approach for computing correlated choice probabilities with many alternatives. Preprint 4034, Optimization Online, 2013.

[2] Awad H. Al-Mohy. A more accurate Briggs method for the logarithm. *Numer. Algorithms*, 59(3):393–402, 2012.

[3] Awad H. Al-Mohy and Nicholas J. Higham. Computing the Fréchet derivative of the matrix exponential, with an application to condition number estimation. *SIAM J. Matrix Anal. Appl.*, 30(4):1639–1657, 2009.

[4] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[5] Awad H. Al-Mohy and Nicholas J. Higham. The complex step approximation to the Fréchet derivative of a matrix function. *Numer. Algorithms*, 53(1):133–148, 2010.

[6] Awad H. Al-Mohy and Nicholas J. Higham. Improved inverse scaling and squaring algorithms for the matrix logarithm. *SIAM J. Sci. Comput.*, 34(4):C153–C169, 2012.

[7] Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. *SIAM J. Sci. Comput.*, 35(4):C394–C410, 2013.

[8] Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. New algorithms for computing the matrix sine and cosine separately or simultaneously. MIMS

EPrint 2014.31, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, June 2014. 32 pp.

[9] S. Amat, S. Busquier, and J. M. Gutiérrez. Geometric constructions of iterative functions to solve nonlinear equations. *Journal of Computational and Applied Mathematics*, 157(1):197–205, 2003.

[10] David Amsallem and Charbel Farhat. An online method for interpolating linear parametric reduced-order models. *SIAM J. Scientific Computing*, 33(5):2169–2198, 2011.

[11] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. Third edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. xxvi+407 pp. ISBN 0-89871-447-8.

[12] Vincent Arsigny, Olivier Commowick, Nicholas Ayache, and Xavier Pennec. A fast and log–Euclidean polyaffine framework for locally linear registration. *J. Math. Imaging Vis*, 33:222–238, 2009.

[13] John Ashburner and Gerard R. Ridgway. Symmetric diffeomorphic modelling of longitudinal structural MRI. *Frontiers in Neuroscience*, 6(197), 2013.

[14] M. Athans and F. C. Schweppe. Gradient matrices and matrix calculations. Tech. Note 53, MIT Lincoln Lab, Lexington, MA, November 1965.

[15] George A. Baker. *Essentials of Padé Approximants*. Academic Press, 1975. xi+306 pp. ISBN 978-0120748556.

[16] D. A. Benson, M. M. Meerschaert, and J. Revielle. Fractional calculus in hydrologic modeling: A numerical perspective. *Adv. Water Resour.*, 51:479–497, 2013.

[17] Rajendra Bhatia. *Matrix Analysis*. Springer-Verlag, New York, 1997. xi+347 pp. ISBN 978-0397948461.

[18] Åke Björck and Sven Hammarling. A Schur method for the square root of a matrix. *Linear Algebra Appl.*, 52–53:127–140, 1983.

[19] L. Susan Blackford, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, Michael Heroux, Linda Kaufman, Andrew Lumsdaine, Antoine Petitet, Roldan Pozo, Karin Remington, and R. Clint Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Software*, 28 (2):135–151, 2002.

[20] K. Burrage, N. Hale, and D. Kay. An efficient implicit FEM scheme for fractional-in-space reaction-diffusion equations. *SIAM J. Sci. Comput.*, 34(4):A2145–A2172, 2012.

[21] João R. Cardoso and F. Silva Leite. Theoretical and numerical considerations about Padé approximants for the matrix logarithm. *Linear Algebra Appl.*, 330: 31–42, 2001.

[22] Arthur Cayley. A memoir on the theory of matrices. *Philos. Trans. Roy. Soc. London*, 148:17–37, 1858.

[23] Sheung Hun Cheng, Nicholas J. Higham, Charles S. Kenney, and Alan J. Laub. Approximating the logarithm of a matrix to specified accuracy. *SIAM J. Matrix Anal. Appl.*, 22(4):1112–1125, 2001.

[24] Dennis Cheung and Felipe Cucker. A note on level-2 condition numbers. *Journal of Complexity*, 21:314–319, 2005.

[25] John P. Coleman. Rational approximations for the cosine function; P-acceptability and order. *Numer. Algorithms*, 3:143–158, 1992.

[26] A. R. Collar. The first fifty years of aeroelasticity. *Aerospace (Royal Aeronautical Society Journal)*, 5:12–20, 1978.

[27] E. B. Davies. Approximate diagonalization. *SIAM J. Matrix Anal. Appl.*, 29(4): 1051–1064, 2007.

[28] Philip I. Davies and Nicholas J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[29] Edvin Deadman and Nicholas J. Higham. Testing matrix function algorithms using identities. MIMS EPrint 2014.13, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, March 2014. 15 pp.

[30] Edvin Deadman, Nicholas J. Higham, and Rui Ralha. Blocked Schur algorithms for computing the matrix square root. In *Applied Parallel and Scientific Computing: 11th International Conference, PARA 2012, Helsinki, Finland*, P. Manninen and P. Öster, editors, volume 7782 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2013, pages 171–182.

[31] J. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Numerische Mathematik*, 51:251–289, 1987.

[32] J. Diblk, D. Ya. Khusainov, J. Lukov, and M. Rikov. Control of oscillating systems with a single delay. *Advances in Difference Equations*, 2010(108218): 1–15, 2010.

[33] L. Dieci, B. Morini, and A. Papini. Computational techniques for real logarithms of matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):570–593, 1996.

[34] J. Dieudonné. *Foundations of Modern Analysis*. Academic Press, New York, 1969. xviii+387 pp. ISBN 0-12-215550-5.

[35] Nicholas J. Dingle and Nicholas J. Higham. Reducing the influence of tiny normwise relative errors on performance profiles. *ACM Trans. Math. Software*, 39(4):24:1–24:11, 2013.

[36] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[37] Ernesto Estrada and Naomichi Hatano. Communicability in complex networks. *Phys. Rev. E*, 77(3):036111, 2008.

[38] Ernesto Estrada and Desmond J. Higham. Network properties revealed through matrix functions. *SIAM Review*, 52(4):696–714, 2010.

[39] Ernesto Estrada, Desmond J. Higham, and Naomichi Hatano. Communicability betweeness in complex networks. *Physica A: Statistical Mechanics and its Applications*, 388(5):764–774, 2009.

[40] J. M. Franco. New methods for oscillatory systems based on ARKN methods. *Appl. Numer. Math.*, 56(8):1040–1053, 2006.

[41] R. A. Frazer, W. J. Duncan, and A. R. Collar. *Elementary Matrices and Some Applications to Dynamics and Differential Equations*. Cambridge University Press, 1938. xviii+416 pp. 1963 printing.

[42] F. R. Gantmacher. *The Theory of Matrices*, volume one. Chelsea, New York, 1959. x+374 pp. ISBN 0-8284-0131-4.

[43] B. García-Mora, C. Santamaría, G. Rubio, and J. L. Pontones. Computing survival functions of the sum of two independent Markov processes. An application to bladder carcinoma treatment. *Int. Journal of Computer Mathematics*, 91(2): 209–220, 2014.

[44] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Fourth edition, John Hopkins University Press, Baltimore, MD, USA, 2013. xxi+756 pp. ISBN 978-1-4214-0794-4.

[45] Lawrence M. Graves. Riemann integration and Taylor's theorem in general analysis. *Transactions of the American Mathematical Society*, 29(1):163–177, 1927.

[46] Gareth I. Hargreaves and Nicholas J. Higham. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms*, 40(4):383–400, 2005.

[47] Harold V. Henderson and S. R. Searle. The vec-permutation matrix, the vec operator and Kronecker products: A review. *Linear and Multilinear Algebra*, 9: 271–288, 1981.

[48] Desmond J. Higham. Condition numbers and their condition numbers. *Linear Algebra Appl.*, 214:193–213, 1995.

[49] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005. xxiii+382 pp. ISBN 0-89871-578-4.

[50] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.maths.manchester.ac.uk/~higham/mctoolbox`.

[51] Nicholas J. Higham. The Matrix Function Toolbox. `http://www.maths.manchester.ac.uk/~higham/mftoolbox`.

[52] Nicholas J. Higham. Computing real square roots of a real matrix. *Linear Algebra Appl.*, 88/89:405–430, 1987.

[53] Nicholas J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(2):317–333, 1993.

[54] Nicholas J. Higham. Evaluating Padé approximants of the matrix logarithm. *SIAM J. Matrix Anal. Appl.*, 22(4):1126–1135, 2001.

[55] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[56] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[57] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. xx+425 pp. ISBN 978-0-898716-46-7.

[58] Nicholas J. Higham and Awad H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19:159–208, 2010.

[59] Nicholas J. Higham and Edvin Deadman. A catalogue of software for matrix functions. Version 1.0. MIMS EPrint 2014.8, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, February 2014. 19 pp.

[60] Nicholas J. Higham and Lijing Lin. A Schur–Padé algorithm for fractional powers of a matrix. *SIAM J. Matrix Anal. Appl.*, 32(3):1056–1078, 2011.

[61] Nicholas J. Higham and Lijing Lin. An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives. *SIAM J. Matrix Anal. Appl.*, 34(3):1341–1360, 2013.

[62] Nicholas J. Higham, D. Steven Mackey, Niloufer Mackey, and Françoise Tisseur. Functions preserving matrix groups and iterations for the matrix square root. *SIAM J. Matrix Anal. Appl.*, 26(3):849–877, 2005.

[63] Nicholas J. Higham and Samuel D. Relton. Estimating the condition number of the Fréchet derivative of a matrix function. MIMS EPrint 2013.84, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, December 2013. 18 pp.

[64] Nicholas J. Higham and Samuel D. Relton. Higher order Fréchet derivatives of matrix functions and the level-2 condition number. *SIAM J. Matrix Anal. Appl.*, 35(3):1019–1037, 2014.

[65] Nicholas J. Higham and Matthew I. Smith. Computing the matrix cosine. *Numer. Algorithms*, 34:13–26, 2003.

[66] Nicholas J. Higham and Françoise Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21(4):1185–1201, 2000.

[67] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Second edition, Cambridge University Press, Cambridge, UK, 2013. xviii+643 pp. ISBN 978-0-521-83940-2.

[68] Weiming Hu, Haiqiang Zuo, Ou Wu, Yunfei Chen, Zhongfei Zhang, and David Suter. Recognition of adult images, videos and web page bags. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7S:Article 28, 24 pages, 2011.

[69] Arieh Iserles and S. P. Nørsett. *Order Stars*. Chapman and Hall, London, 1991. xi+248 pp. ISBN 0-411-35260-5.

[70] Ben Jeuris, Raf Vandebril, and Bart Vandereycken. A survey and comparison of contemporary algorithms for computing the matrix geometric mean. *Electron. Trans. Numer. Anal.*, 39:379–402, 2012.

[71] Isak Jonsson and Bo Kågström. Recursive blocked algorithms for solving triangular systems—Part I: One-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Software*, 28(4):392–415, 2002.

[72] L. V. Kantorovich and G. P. Akilov. *Functional Analysis*. Second edition, Pergamon Press, New York, 1982. xiv+589 pp. ISBN 0-08-023036-9.

[73] C. S. Kenney and A. J. Laub. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.*, 10(2):191–209, 1989.

[74] C. S. Kenney and A. J. Laub. A Schur-Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3): 640–663, 1998.

[75] Peter Lancaster and Miron Tismenetsky. *The Theory of Matrices*. Second edition, Academic Press, London, 1985. xv+570 pp. ISBN 0-12-435560-9.

[76] NAG Ltd. NAG Library. `http://www.nag.co.uk`.

[77] A. Magnus and J. Wynn. On the padé table of cos z. *Proc. Amer. Math. Soc.*, 47(2):361–367, 1975.

[78] Jan R. Magnus and Heinz Neudecker. The commutation matrix: Some properties and applications. *Ann. Statist.*, 7(2):381–394, 1979.

[79] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Revised edition, Wiley, Chichester, UK, 1999. xviii+395 pp. ISBN 0-471-98633-X.

[80] Roy Mathias. A chain rule for matrix functions and applications. *SIAM J. Matrix Anal. Appl.*, 17(3):610–620, 1996.

[81] Dominik L. Michels, Gerrit A. Sobottka, and Andreas G. Weber. Exponential integrators for stiff elastodynamic problems. *ACM Trans. Graph.*, 33(1):7:1–7:20, 2014.

[82] Cleve B. Moler and Charles F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20(4):801–836, 1978.

[83] Cleve B. Moler and Charles F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[84] Kalyan Mukherjea. *Differential Calculus in Normed Linear Spaces*. Second edition, Hindustan Book Agency, New Delhi, India, 2007. 274 pp. ISBN 978-81-85931-76-0.

[85] Igor Najfeld and Timothy F. Havel. Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, 16(3):321–375, 1995.

[86] M. Z. Nashed. Some remarks on variations and differentials. *The American Mathematical Monthly*, 73(4):63–76, 1966.

[87] Simon T. Parker, David M. Lorenzetti, and Michael D. Sohn. Implementing state-space methods for mutlizone contaminant transport. *Building and Environment*, 71:131–139, 2014.

[88] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[89] Daniel Petersson. *A Nonlinear Optimization Approach to $\mathcal{H}_2$-Optimal Modeling and Control*. PhD thesis, Department of Electrical Engineering, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 2013. Dissertation No. 1528.

[90] Daniel Petersson and Johan Löfberg. Model reduction using a frequency-limited $\mathcal{H}_2$-cost. *Systems & Control Letters*, 67:32–39, 2014.

[91] Maria Pusa and Jaakko Leppănen. Computing the matrix exponential in burnup calculations. *Nuclear Science and Engineering*, 164(2):140–150, 2010.

[92] John R. Rice. A theory of condition. *SIAM J. Numer. Anal.*, 3(2):287–310, 1966.

[93] Jarek Rossignac and Álvar Vinacua. Steady affine motions and morphs. *ACM Trans. Graph.*, 30(5):Article 116, 16 pages, 2011.

[94] Patrick Sanan. *Geometric Elasticity for Graphics, Simulation and Computation*. PhD thesis, California Institute of Technology, Pasadena, California, USA, 2014.

[95] Jorge Sastre, Javier Ibáñez, Pedro Ruiz, and Emilio Defez. Efficient computation of the matrix cosine. *Appl. Math. Comput.*, 219(14):7575–7585, 2013.

[96] Scipy. `http://www.scipy.org`.

[97] S. M. Serbin. Rational approximations of trigonometric matrices with application to second-order systems of differential equations. *Appl. Mathm. Comput.*, 5(1): 75–92, 1979.

[98] Steven M. Serbin and Sybil A. Blalock. An algorithm for computing the matrix cosine. *SIAM J. Sci. Stat. Comput.*, 1(2):198–204, 1980.

[99] Jaita Pankaj Sharma and Raju K. George. Controllability of matrix second order systems: A trigonometric matrix approach. *Electronic Journal of Differential Equations*, 2007(80):1–14, 2007.

[100] R. B. Sidje. EXPOKIT: A software package for computing matrix exponentials. *ACM Trans. Math. Softw.*, 24(1):130–156, 1998.

[101] J. J. Sylvester. On the equation to the secular inequalities in the planetary theory. *Philosophical Magazine*, 16:267–269, 1883.

[102] Virginia J. Torczon. *Multi-directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Dept. of Mathematical Sciences, Rice Univeristy, Houston, TX, 1989.

[103] Virginia J. Torczon. On the convergence of the multidirectional search algorithm. *SIAM J. Optimization*, 1:123–145, 1991.

[104] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2013. vii+305 pp. ISBN 978-1-611972-39-9.

[105] Charles F. Van Loan. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal.*, 14(6):971–981, 1977.

[106] Bin Wang, Kai Liu, and Xinyuan Wu. A Filon-type asymptotic approach to solving highly oscilliatory second-order initial value problems. *Journal of Computational Physics*, 243:210–223, 2013.

[107] Ding Yuan and Warnick Kernan. Explicit solutions for exit-only radioactive decay chains. *J. Appl. Phys.*, 101(9):094907 1–12, 2007.

[108] Ernesto Zacur, Matias Bossa, and Salvador Olmos. Multivariate tensor-based morphometry with a right-invariant Riemannian distance on $GL^{+}(n)$. *J. Math Imaging Vis.*, 50(1–2):18–31, 2013.

[109] Ernesto Zacur, Matias Bossa, and Salvador Olmos. Left-invariant Riemannian geodesics on spatial transformation groups. *SIAM J. Imaging Sci.*, 7(3):1503–1557, 2014.

Index