ALGORITHMS FOR THE MATRIX EXPONENTIAL AND ITS FRÉCHET DERIVATIVE

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

2010

Awad H. Al-Mohy
School of Mathematics

Contents

List of Tables

List of Figures

Abstract

New algorithms for the matrix exponential and its Fréchet derivative are presented. First, we derive a new scaling and squaring algorithm (denoted $\mathrm{expm}_{\mathrm{new}}$) for computing e^A, where A is any square matrix, that mitigates the overscaling problem. The algorithm is built on the algorithm of Higham [*SIAM J. Matrix Anal. Appl.*, 26 (4):1179–1193, 2005] but improves on it by two key features. The first, specific to triangular matrices, is to compute the diagonal elements in the squaring phase as exponentials instead of powering them. The second is to base the backward error analysis that underlies the algorithm on members of the sequence $\{\|A^k\|^{1/k}\}$ instead of $\|A\|$. The terms $\|A^k\|^{1/k}$ are estimated without computing powers of A by using a matrix 1-norm estimator.

Second, a new algorithm is developed for computing the action of the matrix exponential on a matrix, $e^{tA}B$, where A is an $n \times n$ matrix and B is $n \times n_0$ with $n_0 \ll n$. The algorithm works for any A, its computational cost is dominated by the formation of products of A with $n \times n_0$ matrices, and the only input parameter is a backward error tolerance. The algorithm can return a single matrix $e^{tA}B$ or a sequence $e^{t_k A}B$ on an equally spaced grid of points t_k. It uses the scaling part of the scaling and squaring method together with a truncated Taylor series approximation to the exponential. It determines the amount of scaling and the Taylor degree using the strategy of $\mathrm{expm}_{\mathrm{new}}$. Preprocessing steps are used to reduce the cost of the algorithm. An important application of the algorithm is to exponential integrators for ordinary differential equations. It is shown that the sums of the form $\sum_{k=0}^{p} \varphi_k(A)u_k$ that arise in exponential integrators, where the φ_k are related to the exponential function, can be expressed in terms of a single exponential of a matrix of dimension $n+p$ built by augmenting A with additional rows and columns.

Third, a general framework for simultaneously computing a matrix function, $f(A)$, and its Fréchet derivative in the direction E, $L_f(A,E)$, is established for a wide range of matrix functions. In particular, we extend the algorithm of Higham and $\mathrm{expm}_{\mathrm{new}}$ to two algorithms that intertwine the evaluation of both e^A and $L(A,E)$ at a cost about three times that for computing e^A alone. These two extended algorithms are then adapted to algorithms that simultaneously calculate e^A together with an estimate of its condition number.

Finally, we show that $L_f(A,E)$, where f is a real-valued matrix function and A and E are real matrices, can be approximated by $\operatorname{Im} f(A+ihE)/h$ for some suitably small h. This approximation generalizes the complex step approximation known in the scalar case, and is proved to be of second order in h for analytic functions f and also for the matrix sign function. It is shown that it does not suffer the inherent cancellation that limits the accuracy of finite difference approximations in floating point arithmetic. However, cancellation does nevertheless vitiate the approximation

when the underlying method for evaluating f employs complex arithmetic. The complex step approximation is attractive when specialized methods for evaluating the Fréchet derivative are not available.

Declaration

No portion of the work referred to in this 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

Publications

This thesis is based on five publications detailed as follows.

- Some of the material in Chapter 1 and Section 2.7 are based on the paper: N. J. Higham and A. H. Al-Mohy. Computing Matrix Functions. *Acta Numerica*, 19: 159–208, 2010.

- The material in Chapter 2 is based on the paper: A. H. Al-Mohy and N. J. Higham. A New Scaling and Squaring Algorithm for the Matrix Exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.
  (This paper won the 2009 SIAM Student Paper Prize).

- The material in Chapter 3 is based on the paper: A. H. Al-Mohy and N. J. Higham. Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators. MIMS EPrint 2010.30, March 2010. Submitted to *SIAM J. Sci. Comput.*

- The material in Chapter 4 is based on the paper: A. H. Al-Mohy and N. 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.

- The material in Chapter 5 is based on the paper: A. H. Al-Mohy and N. J. Higham. The Complex Step Approximation to the Fréchet Derivative of a Matrix Function. *Numer. Algorithms*, 53(1):133–148, 2010.

Acknowledgements

I have unbounded pleasure and am very proud that my work towards the PhD degree has met the standards of Professor Nick Higham. I owe Nick a debt of gratitude for his great supervision, continuous support, and sharing knowledge and expertise. Words cannot express or carry my heartfelt gratitude, appreciation, and thanks for all the support, guidance and time he had provided during my stay in Manchester.

My gratitude extends to Professor Sven Hammarling of NAG Ltd, Oxford, and Dr. Françoise Tisseur of the University of Manchester for their valuable comments on my thesis. I highly appreciate their time they spent on reading my work.

Many thanks go to the Ministry of Higher Education in Saudi Arabia for financially sponsoring my postgraduate studies and life expenses (for me and my family) during our stay in both the US and the UK. I would also like to thank SIAM for awarding me the *2009 SIAM Student Paper Prize.*

The debt that can never be repaid is owed to my parents and my wife for their continuous love, support, and praying. I dedicate this thesis to them.

Dedication

To My Parents & My Wife

Chapter 1

Background

1.1 Introduction

Matrix functions are as old as matrix algebra itself. In his memoir that initiated the study of matrix algebra, Cayley [9] treated matrix square roots. The theory of matrix functions was subsequently developed by many mathematicians over the ensuing 100 years. Today, functions of matrices are widely used in science and engineering and are of growing interest, due to the succinct way they allow solutions to be expressed and recent advances in numerical algorithms for computing them. New applications are regularly being found, but the archetypal application of matrix functions is in the solution of differential equations. Early recognition of the important role of the matrix exponential in this regard can be found in the book *Elementary Matrices and Some Applications to Dynamics and Differential Equations* by aerospace engineers [21], which was “the first book to treat matrices as a branch of applied mathematics” [11].

This thesis concerns new algorithms for the matrix exponential and its Fréchet derivative and is organized as follows. The present chapter reviews basic definitions and fundamental properties for matrix functions and Fréchet derivatives. It also shows the relation between the Fréchet derivative and the condition number of the matrix function and describes algorithms for exact calculation or estimation of the condition number. This chapter supplies the necessary background for the rest of the thesis.

Chapter 2 presents a new algorithm for computing the matrix exponential for a general complex matrix. The algorithm implements the scaling and squaring method for the matrix exponential, which is based on the approximation $e^A \approx (r_m(2^{-s}A))^{2^s}$, where $r_m(x)$ is the $[m/m]$ Padé approximant to e^x and the integers m and s are to be chosen. Several authors have identified a weakness of existing scaling and squaring algorithms termed overscaling, in which a value of s much larger than necessary is chosen, causing a loss of accuracy in floating point arithmetic. Building on the scaling and squaring algorithm of Higham [30], [32], which is used by the MATLAB function `expm`, we derive a new algorithm that alleviates the overscaling problem. Two key ideas are employed. The first, specific to triangular matrices, is to compute the diagonal elements in the squaring phase as exponentials instead of from powers of r_m. The second idea is to base the backward error analysis that underlies the algorithm on members of the sequence $\{\|A^k\|^{1/k}\}$ instead of $\|A\|$, since for non-normal matrices it is possible that $\|A^k\|^{1/k}$ is much smaller than $\|A\|$, and indeed this is likely when overscaling occurs in existing algorithms. The terms $\|A^k\|^{1/k}$ are estimated without computing powers of A by using a matrix 1-norm estimator in conjunction with a bound of the form $\|A^k\|^{1/k} \le \max\bigl(\|A^p\|^{1/p}, \|A^q\|^{1/q}\bigr)$ that holds for certain fixed p and q less than k. The improvements to the truncation error bounds have to be balanced by the potential for a large $\|A\|$ to cause inaccurate evaluation of r_m in floating point arithmetic. We employ rigorous error bounds along with some heuristics to ensure that rounding errors are kept under control. Our numerical experiments show that the new algorithm generally provides accuracy at least as good as the existing algorithm of Higham at no higher cost, while for matrices that are triangular or cause overscaling it usually yields significant improvements in accuracy, cost, or both. At the end of the chapter, we compare the efficiency of diagonal Padé approximant and Taylor series within the scaling and squaring method.

In Chapter 3, a new algorithm is developed for computing the action of the matrix exponential on a matrix, $e^{tA}B$, where A is an $n \times n$ matrix and B is $n \times n_0$ with $n_0 \ll n$. The algorithm works for any A, its computational cost is dominated by the formation of products of A with $n \times n_0$ matrices, and the only input parameter is a backward error tolerance. The algorithm can return a single matrix $e^{tA}B$ or a sequence $e^{t_kA}B$ on an equally spaced grid of points t_k. It uses the scaling part of the scaling and squaring method together with a truncated Taylor series approximation to the exponential. It determines the amount of scaling and the Taylor degree using the analysis developed in Chapter 2, which provides sharp truncation error bounds expressed in terms of the quantities $\|A^k\|^{1/k}$ for a few values of k, where the norms are estimated using a matrix norm estimator. Shifting and balancing are used as preprocessing steps to reduce the cost of the algorithm. Numerical experiments show that the algorithm performs in a numerically stable fashion across a wide range of problems, and analysis of rounding errors and of the conditioning of the problem provides theoretical support. Experimental comparisons with two Krylov-based MATLAB codes show the new algorithm to be sometimes much superior in terms of computational cost and accuracy. An important application of the algorithm is to exponential integrators for ordinary differential equations. It is shown that the sums of the form $\sum_{k=0}^{p} \varphi_k(A)u_k$ that arise in exponential integrators, where the φ_k are related to the exponential function, can be expressed in terms of a single exponential of a matrix of dimension $n + p$ built by augmenting A with additional rows and columns, and the algorithm developed here can therefore be employed.

In Chapter 4, we show that the implementation of the scaling and squaring method in Chapter 2 can be extended to compute both e^A and the Fréchet derivative at A in the direction E, denoted by $L_{\exp}(A, E)$, at a cost about three times that for computing e^A alone. The algorithm is derived from the scaling and squaring method by differentiating the Padé approximants and the squaring recurrence, re-using quantities computed during the evaluation of the Padé approximant, and intertwining the recurrences in the squaring phase. To guide the choice of algorithmic parameters an extension of the existing backward error analysis for the scaling and squaring method is developed which shows that, modulo rounding errors, the approximations obtained are $e^{A+\Delta A}$ and $L(A + \Delta A, E + \Delta E)$, with the same ΔA in both cases, and with computable bounds on $\|\Delta A\|$ and $\|\Delta E\|$. The algorithm for $L_{\exp}(A, E)$ is used to develop an algorithm that computes e^A together with an estimate of its condition number. In addition to results specific to the exponential, we develop some results and techniques for arbitrary functions. We show how a matrix iteration for $f(A)$ yields an iteration for the Fréchet derivative and show how to efficiently compute the Fréchet derivative of a power series. We also show that a matrix polynomial and its Fréchet derivative can be evaluated at a cost at most three times that of computing the polynomial itself and give a general framework for evaluating a matrix function and its Fréchet derivative via Padé approximation.

In Chapter 5, we show that the Fréchet derivative of a matrix function $f : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ at A in the direction E, where A and E are real matrices, can be approximated by $\operatorname{Im} f(A + ihE)/h$ for some suitably small h. This approximation, requiring a single function evaluation at a complex argument, generalizes the complex step approximation known in the scalar case. The approximation is proved to be of second order in h for analytic functions f and also for the matrix sign function. It is shown that it does not suffer the inherent cancellation that limits the accuracy of finite difference approximations in floating point arithmetic. However, cancellation does nevertheless vitiate the approximation when the underlying method for evaluating f employs complex arithmetic. The ease of implementation of the approximation, and its superiority over finite differences, make it attractive when specialized methods for evaluating the Fréchet derivative are not available, and in particular for condition number estimation when used in conjunction with a block 1-norm estimation algorithm.

Finally, Chapter 6 presents our concluding remarks and discussions.

1.2 Definitions

The concept of *matrix function* has various meanings in the matrix analysis literature. Matrix norms, determinants, traces, and condition numbers are examples of matrix functions of the form $f : \mathbb{C}^{n\times n} \to \mathbb{C}$. Despite the importance of such functions, the scope of this thesis is on matrix functions $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ defined in terms of an underlying scalar function f.

As the definition of the elementary operations addition, subtraction, multiplication, and division is readily extended to matrices, we have the tools to obtain a value of f, where f is a polynomial or rational function, at a matrix argument by just replacing the scalar by the matrix and employing matrix operations. For example, take

$$f(x) = \frac{1 + x + x^3}{1 - x^2} = (1 + x + x^3)(1 - x^2)^{-1}, \qquad x \neq \pm 1.$$

Then for $A \in \mathbb{C}^{n\times n}$,

$$f(A) = (I + A + A^3)(I - A^2)^{-1}$$

is well-defined if ±1 are not eigenvalues of A. The theory of matrix functions of this type is based on the *Jordan canonical form* (JCF), which exists for any matrix

$A \in \mathbb{C}^{n\times n}$:

$$Z^{-1}AZ = J = \operatorname{diag}(J_1, J_2, \ldots, J_p), \tag{1.1a}$$

$$J_k = J_k(\lambda_k) = \begin{bmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{bmatrix} \in \mathbb{C}^{m_k\times m_k}, \tag{1.1b}$$

where Z is nonsingular and $\sum_{k=1}^{p} m_k = n$.

Definition 1.2.1 *Denote by $\lambda_1, \ldots, \lambda_s$ the distinct eigenvalues of $A \in \mathbb{C}^{n\times n}$ and let n_i be the index of λ_i, that is, the order of the largest Jordan block in which λ_i appears. The function f is said to be defined on the spectrum of A if the values*

$$f^{(j)}(\lambda_k), \quad j = 0\colon n_i - 1, \quad k = 1\colon s$$

exist.

Definition 1.2.2 (matrix function via JCF) *Let f be defined on the spectrum of $A \in \mathbb{C}^{n\times n}$ and let A have the Jordan canonical form* (1.1). *Then*

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

where

$$f(J_k) := \begin{bmatrix} f(\lambda_k) & f'(\lambda_k) & \cdots & \dfrac{f^{(n_k-1)}(\lambda_k)}{(n_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & f'(\lambda_k) \\ & & & f(\lambda_k) \end{bmatrix}. \tag{1.2}$$

When the function f is multivalued and A has a repeated eigenvalue occurring in more than one Jordan block (i.e., A is derogatory), we will take the same branch for f and its derivatives in each Jordan block. This gives a *primary matrix function.* If different branches are taken for the same eigenvalue in two different Jordan blocks then a *nonprimary matrix function* is obtained. We will be concerned here only with primary matrix functions, and it is these that are needed in most applications. For more on nonprimary matrix functions see [31, Sec. 1.4].

An alternative way to define $f(A)$ is as follows.

Definition 1.2.3 (polynomial interpolation definition of $f(A)$) *Let f be defined on the spectrum of $A \in \mathbb{C}^{n\times n}$ with the distinct eigenvalues, $\lambda_1, \ldots, \lambda_s$, and let n_i be the index of λ_i. Then $f(A) := r(A)$, where r is the unique Hermite interpolating polynomial of degree less than $\sum_{i=1}^{s} n_i$ that satisfies the interpolation conditions*

$$r^{(j)}(\lambda_i) = f^{(j)}(\lambda_i), \qquad j = 0\colon n_i - 1, \quad i = 1\colon s. \tag{1.3}$$

A proof of the equivalence of these two definitions can be found in [31, Thm. 1.12]. The equivalence is easily demonstrated for the $m_k \times m_k$ Jordan block $J_k(\lambda_k)$ in (1.1b). The polynomial satisfying the interpolation conditions (1.3) is then

$$r(t) = f(\lambda_k) + (t-\lambda_k)f'(\lambda_k) + \frac{(t-\lambda_k)^2}{2!}f''(\lambda_k) + \cdots + \frac{(t-\lambda_k)^{m_k-1}}{(m_k-1)!}f^{(m_k-1)}(\lambda_k),$$

which is just the first m_k terms of the Taylor series of f about λ_k (assuming the Taylor series exists). Hence, from Definition 1.2.3,

$$\begin{aligned} f(J_k(\lambda_k)) &= r(J_k(\lambda_k)) \\ &= f(\lambda_k)I + (J_k(\lambda_k) - \lambda_k I)f'(\lambda_k) + \frac{(J_k(\lambda_k)-\lambda_k I)^2}{2!}f''(\lambda_k) + \cdots \\ &\quad + \frac{(J_k(\lambda_k)-\lambda_k I)^{m_k-1}}{(m_k-1)!}f^{(m_k-1)}(\lambda_k). \end{aligned}$$

The matrix $(J_k(\lambda_k) - \lambda_k I)^j$ is zero except for 1s on the jth superdiagonal. This expression for $f(J_k(\lambda_k))$ is therefore equal to that in (1.2).

1.3 Properties

One of the most important basic properties is that $f(A)$ is a polynomial in $A \in \mathbb{C}^{n\times n}$, which is immediate from Definition 1.2.3. However, the coefficients of that polynomial depend on A. This property is not surprising in view of the Cayley–Hamilton theorem, which says that any matrix satisfies its own characteristic equation: $q(A) = 0$, where $q(t) = \det(tI - A)$ is the characteristic polynomial. The theorem implies that the nth power of A, and inductively all higher powers, are expressible as a linear combination of I, A, $\dots$, A^{n-1}. Thus any power series in A can be reduced to a polynomial in A of degree at most $n-1$ (with coefficients depending on A).

Other important properties are collected in the next result.

Theorem 1.3.1 ([31, Thm. 1.13]) *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) *$f(A^T) = f(A)^T$;*

(3) *$f(XAX^{-1}) = Xf(A)X^{-1}$;*

(4) *the eigenvalues of $f(A)$ are $f(\lambda_i)$, where the λ_i are the eigenvalues of A;*

(5) *if $A = (A_{ij})$ is block triangular then $F = f(A)$ is block triangular with the same block structure as A, and $F_{ii} = f(A_{ii})$;*

(6) *if $A = \mathrm{diag}(A_{11}, A_{22}, \dots, A_{mm})$ is block diagonal then*

$$f(A) = \mathrm{diag}\big(f(A_{11}), f(A_{22}), \dots, f(A_{mm})\big).$$

(7) *$f(I_m \otimes A) = I_m \otimes f(A)$, where $\otimes$ is the Kronecker product defined in Subsection* 1.6.1*;*

(8) *$f(A \otimes I_m) = f(A) \otimes I_m$.*

It is often convenient to represent a matrix function as a power series or Taylor series. The next result explains when such a series converges [31, Thm. 4.7].

Theorem 1.3.2 (convergence of matrix Taylor series) *Suppose f has a Taylor series expansion*

$$f(z) = \sum_{k=0}^{\infty} a_k (z-\alpha)^k \qquad \left(a_k = \frac{f^{(k)}(\alpha)}{k!} \right) \tag{1.4}$$

with radius of convergence r. If $A \in \mathbb{C}^{n\times n}$ then $f(A)$ is defined and is given by

$$f(A) = \sum_{k=0}^{\infty} a_k (A-\alpha I)^k \tag{1.5}$$

if and only if each of the distinct eigenvalues $\lambda_1, \ldots, \lambda_s$ of A satisfies one of the conditions

(1) $|\lambda_i - \alpha| < r$,

(2) $|\lambda_i - \alpha| = r$ *and the series for $f^{(n_i-1)}(\lambda)$ (where n_i is the index of λ_i) is convergent at the point $\lambda = \lambda_i$, $i = 1\colon s$.*

The matrix function therefore can be defined everywhere from the power series for functions having a power series with an infinite radius of convergence. Thus the matrix exponential, cosine, and sine are, respectively, given by

$$\begin{aligned} e^A &= I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots, \\ \cos(A) &= I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \cdots, \\ \sin(A) &= A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!} + \cdots. \end{aligned}$$

1.4 Floating point arithmetic

A floating point number system F is a bounded subset of the real numbers whose elements have the representation

$$y = \pm m \times \beta^{e-t},$$

where m, β, e, and t are integers known as *significand, base, exponent*, and *precision*, respectively, with $0 \le m \le \beta^t - 1$. The representation of any nonzero element of F is unique if m is selected so that $\beta^{t-1} \le m \le \beta^t - 1$. The quantity $u := \frac{1}{2}\beta^{1-t}$ is called the *unit roundoff.* The boundness of F implies the existence of two integers $e_{\min}$ and $e_{\max}$ with $e_{\min} \le e \le e_{\max}$. Thus

$$F \subset [-\beta^{e_{\max}}(1-\beta^{-t}), \beta^{e_{\max}}(1-\beta^{-t})] := R \subset \mathbb{R}.$$

The set R is called the *range* of F, and an element of F closest to $x \in R$ is denoted by $fl(x)$. The following theorem draws the relation between x and $fl(x)$.

Theorem 1.4.1 ([29, Thm. 2.2]) *If $x \in \mathbb{R}$ lies in the range of F then there exists $\delta \in \mathbb{R}$ such that*

$$fl(x) = x(1+\delta), \quad |\delta| < u. \qquad \square$$

Obviously by the definition of $fl(\cdot)$, $fl(x) = x$ if and only if $x \in F$. The *floating point arithmetic* or *finite precision arithmetic* is the calculation that involves the elementary operations: addition, multiplication, subtraction, and division on the elements of F. These operations are known as *floating point operations* (flops), and the counting of them is a very important measure of the complexity of numerical algorithms.

The standard model of the floating point arithmetic that underlies rounding error analysis is

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| < u, \quad \text{op} = +, -, *, /,$$

where $x, y \in F$ and $fl(x \text{ op } y)$ lies in the range of F. In addition, the following variant of this model can also be used

$$fl(x \text{ op } y) = \frac{x \text{ op } y}{1+\delta}, \quad |\delta| < u.$$

The IEEE binary arithmetic standard specifies two basic floating point formats: double and single. The IEEE double format has a significand precision of 53 bits and occupies 64 bits overall while the IEEE single format has a significand precision of 24 bits and occupies 32 bits. The unit roundoff in the IEEE double and single precisions are $u = 2^{-53} \approx 1.11 \times 10^{-16}$ and $u = 2^{-24} \approx 5.96 \times 10^{-8}$, respectively, [29, Sec. 2.3], [59, Chap. 4].

1.5 Forward and backward errors

Error analysis in numerical linear algebra and matrix functions exploits two important types of errors known as *forward* and *backward errors*. Given a numerical algorithm that approximates the matrix function f at a point X by $\widehat{Y}$, the quantities $\|f(X)-\widehat{Y}\|$ and $\|f(X)-\widehat{Y}\|/\|f(X)\|$ are *absolute* and *relative forward errors*, respectively. If a matrix $\mathit{\Delta} X$ exists so that $\widehat{Y} = f(X + \mathit{\Delta} X)$ then $\|\mathit{\Delta} X\|$ and $\|\mathit{\Delta} X\|/\|X\|$ are *absolute* and *relative backward errors*, respectively. If in floating point arithmetic either the forward or backward error is always "small", usually of the order of the unit roundoff, the numerical algorithm is said to be *forward stable* or *backward stable*, respectively. A result of the form

$$\widehat{Y} + \mathit{\Delta} Y = f(X + \mathit{\Delta} X), \qquad \|\mathit{\Delta} Y\| \le \epsilon_1 \|Y\|, \quad \|\mathit{\Delta} X\| \le \epsilon_2 \|X\|$$

is known as a *mixed-forward-backward error* result. If a numerical algorithm produces sufficiently small ϵ_1 and ϵ_2, it is said to be *mixed-forward-backward stable* or, in short, *numerically stable.*

When backward error, forward error, and the condition number are defined in a consistent fashion, they are linked by the *rule of thumb*:

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

In the next section, we review fundamental definitions, properties, and algorithms for the Fréchet derivative and its applications in condition number measurement.

1.6 Fréchet derivative and conditioning of a matrix function

The condition number is a measure of stability or sensitivity of a problem. In this context, the problem is said to be *well-conditioned* if the condition number is small and *ill-conditioned* otherwise, where the definition of *large* or *small* condition number is problem-dependent. We are interested in the condition number of a matrix function.

Definition 1.6.1 *The condition number of the matrix function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ at a point $X \in \mathbb{C}^{n\times n}$ is defined as*

$$\operatorname{cond}(f, X) := \lim_{\epsilon\to 0} \sup_{\|E\|\le\epsilon\|X\|} \frac{\|f(X+E) - f(X)\|}{\epsilon\|f(X)\|}, \tag{1.6}$$

where the norm is any matrix norm.

Definition 1.6.2 (Fréchet derivative of a matrix function) *The Fréchet derivative of a matrix function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ at a point $X \in \mathbb{C}^{n\times n}$ is a linear operator*

$$\begin{array}{ccc} \mathbb{C}^{n\times n} & \xrightarrow{L_f(X)} & \mathbb{C}^{n\times n} \\ E & \longmapsto & L_f(X, E) \end{array}$$

such that

$$f(X+E) - f(X) - L_f(X, E) = o(\|E\|) \tag{1.7}$$

for all $E \in \mathbb{C}^{n\times n}$. $L_f(X, E)$ is read as "the Fréchet derivative of f at X in the direction E". If such an operator exists, f is said to be Fréchet differentiable. The norm of $L_f(X)$ is defined as

$$\|L_f(X)\| := \max_{Z\neq 0} \frac{\|L_f(X, Z)\|}{\|Z\|} \tag{1.8}$$

The following theorem gives a necessary condition for existence of the Fréchet derivative of the function f.

Theorem 1.6.3 ([31, Thm. 3.8]) *Let f be $2n-1$ times continuously differentiable on $\mathcal{D}$. For $X \in \mathbb{C}^{n\times n}$ with spectrum in $\mathcal{D}$ the Fréchet derivative $L_f(X, E)$ exists and is continuous in the variables X and E.*

The condition number of f at X can be expressed in terms of the Fréchet derivative.

Theorem 1.6.4 ([31, Thm. 3.1]) *Suppose the function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ is Fréchet differentiable at $X \in \mathbb{C}^{n\times n}$. Then*

$$\operatorname{cond}(f, X) = \frac{\|L_f(X)\|\,\|X\|}{\|f(X)\|}. \qquad \square$$

The Fréchet derivative satisfies the following sum, product, and chain rules.

Theorem 1.6.5 ([31, Thms. 3.2–3.4]) *Let $g, h : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ be Fréchet differentiable at X. Then for all $E \in \mathbb{C}^{n\times n}$*

(a) $L_{\alpha g+\beta h}(X, E) = \alpha L_g(X, E) + \beta L_h(X, E)$, $\forall\ \alpha, \beta \in \mathbb{C}$.

(b) $L_{gh}(X, E) = L_g(X, E)h(X) + g(X)L_h(X, E)$.

(c) $L_{g\circ h}(X, E) = L_g(h(X), L_h(X, E))$.

1.6.1 Kronecker form of the Fréchet derivative

Linear matrix equations can be conveniently represented by the Kronecker product. Given $A = [a_{ij}] \in \mathbb{C}^{m\times n}$ and $B \in \mathbb{C}^{p\times q}$, the Kronecker product is $A \otimes B = [a_{ij}B] \in \mathbb{C}^{mp\times nq}$. The operator $\mathrm{vec} : \mathbb{C}^{m\times n} \to \mathbb{C}^{mn}$, defined by

$$\mathrm{vec}(A) = [a_{11}, \dots, a_{m1}, a_{12}, \dots, a_{m2}, \dots, a_{1n}, \dots, a_{mn}]^T,$$

is an essential component in Kronecker forms of linear matrix equations; vec stacks the columns of a matrix into one long vector. It is straightforward to see that vec is linear and a bijection, so the inverse operator vec^{-1} is well-defined, linear, and a bijection. The operator vec^{-1} rebuilds the matrix whose columns vec has stacked into a long vector. That is, for any $v \in \mathbb{C}^{mn}$, $\mathrm{vec}^{-1}(v) = V \in \mathbb{C}^{m\times n}$, where $\mathrm{vec}(V) = v$. A key result is the following lemma.

Lemma 1.6.6 ([37, Lemma 4.3.1]) *For $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$, and $X \in \mathbb{C}^{n\times p}$,*

$$\mathrm{vec}(AXB) = (B^T \otimes A)\,\mathrm{vec}(X). \qquad \square$$

Define the operator $T_X : \mathbb{C}^{n^2} \to \mathbb{C}^{n^2}$ by $T_X(v) = \mathrm{vec}(L_f(X, \mathrm{vec}^{-1}(v)))$. By linearity of the operator $L_f(X)$ and the operators vec and vec^{-1}, T_X is linear. As it operates on a finite dimensional vector space, T_X can be represented by a matrix $K \in \mathbb{C}^{n^2\times n^2}$ that depends on the function f at the matrix X, so we write $K := K_f(X)$. Thus $\mathrm{vec}(L_f(X, \mathrm{vec}^{-1}(v))) = K_f(X)v$. Hence for any $E \in \mathbb{C}^{n\times n}$ with $v = \mathrm{vec}(E)$, we obtain

$$\mathrm{vec}(L_f(X, E)) = K_f(X)\,\mathrm{vec}(E). \tag{1.9}$$

The matrix $K_f(X)$ can be seen to be exactly the Jacobian of the operator $F_f : \mathbb{C}^{n^2} \to \mathbb{C}^{n^2}$, defined by $F_f(v) = \mathrm{vec}(f(\mathrm{vec}^{-1}(v)))$, at $v = \mathrm{vec}(X)$.

Now we write the Kronecker forms of the rules of the Fréchet derivative stated in Theorem 1.6.5. The next theorem appears to be new.

Theorem 1.6.7 *Let $g, h : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ be Fréchet differentiable at X. Then*

(a) $K_{\alpha g+\beta h}(X) = \alpha K_g(X) + \beta K_h(X)$.

(b) $K_{gh}(X) = \big(h(X^T) \otimes I_n\big)K_g(X) + \big(I_n \otimes g(X)\big)K_h(X)$.

(c) $K_{g\circ h}(X) = K_g(h(X))K_h(X)$.

Proof. Part (a) follows immediately from (1.9) by applying the vec operator to both sides of Theorem 1.6.5 (a). For part (b), we apply the vec on both sides of Theorem 1.6.5 (b) and use Lemma 1.6.6 and (1.9).

$$\begin{aligned} K_{gh}(X)\,\mathrm{vec}(E) &= \mathrm{vec}(L_{gh}(X, E)) \\ &= \mathrm{vec}\big(I_n\, L_g(X, E)\, h(X)\big) + \mathrm{vec}\big(g(X)\, L_h(X, E)\, I_n\big) \\ &= \big(h(X)^T \otimes I_n\big)\,\mathrm{vec}(L_g(X, E)) + \big(I_n \otimes g(X)\big)\,\mathrm{vec}(L_h(X, E)) \\ &= \big[\big(h(X^T) \otimes I_n\big)K_g(X) + \big(I_n \otimes g(X)\big)K_h(X)\big]\,\mathrm{vec}(E). \end{aligned}$$

Part (c) follows by applying the vec operator on both sides of Theorem 1.6.5 (c) and using (1.9).

$$\begin{aligned} K_{g\circ h}(X)\,\mathrm{vec}(E) &= \mathrm{vec}(L_{g\circ h}(X, E)) \\ &= \mathrm{vec}(L_g(h(X), L_h(X, E))) \\ &= K_g(h(X))\,\mathrm{vec}(L_h(X, E)) \\ &= K_g(h(X))K_h(X)\,\mathrm{vec}(E). \qquad \square \end{aligned}$$

1.6.2 Computing or estimating the condition number

In view of Theorem 1.6.4, the key component of computing or estimating the condition number is computing or estimating the norm of the Fréchet derivative, $\|L_f(X)\|$. Using the fact that $\|v\|_2 = \|\operatorname{vec}^{-1}(v)\|_F$ for every $v \in \mathbb{C}^{n^2}$ and (1.9), we have

$$\|K_f(X)\|_2 = \max_{v\neq 0} \frac{\|K_f(X)v\|_2}{\|v\|_2} = \max_{v\neq 0} \frac{\|L_f(X, \operatorname{vec}^{-1}(v))\|_F}{\|\operatorname{vec}^{-1}(v)\|_F} = \|L_f(X)\|_F. \qquad (1.10)$$

Thus, the condition number can be obtained exactly in the Frobenius norm via the relation:

$$\operatorname{cond}(f, X) = \frac{\|K_f(X)\|_2 \, \|X\|_F}{\|f(X)\|_F}.$$

Having a method for computing $L_f(X, E)$, we can therefore compute the jth column of $K_f(X)$ explicitly via $K_f(X)e_j = \operatorname{vec}(L_f(X, \operatorname{vec}^{-1}(e_j)))$, where $\{e_j : j = 1\colon n^2\}$ is the standard basis for $\mathbb{C}^{n^2}$.

Algorithm 1.6.8 (exact condition number) *Given a function f and its Fréchet derivative and $X \in \mathbb{C}^{n\times n}$, this algorithm computes* $\operatorname{cond}(f, X)$ *in the Frobenius norm.*

```
for $j = 1 : n^2$
    Compute $Y = L_f(X, \operatorname{vec}^{-1}(e_j))$
    $K(:, j) = \operatorname{vec}(Y)$
end
$\operatorname{cond}(f, X) = \|K\|_2 \|X\|_F / \|f(X)\|_F$
```

Cost: $O(n^5)$ flops if $f(X)$ and $L_f(X, E)$ cost $O(n^3)$ flops.

This algorithm is prohibitively expensive for large n in practice, so the condition number needs to be estimated. That is, we need to estimate $\|K_f(X)\|_2$. A candidate is the power method to estimate the square root of the modulus of the largest eigenvalue of $K_f(X)^* K_f(X)$ since

$$\|K_f(X)\|_2 = \|K_f(X)^* K_f(X)\|_2^{1/2} = \lambda_{\max}(K_f(X)^* K_f(X))^{1/2}.$$

Of course we do not intend to form the matrix $K_f(X)$ explicitly as this is very costly. What is required by the power method is the action of the matrices $K_f(X)$ and $K_f(X)^*$ on vectors z and w, respectively, which can be achieved using

$$\operatorname{vec}^{-1}(K_f(X)z) = L_f(X, \operatorname{vec}^{-1}(z)), \quad \operatorname{vec}^{-1}(K_f(X)^* w) = L_f^\star(X, \operatorname{vec}^{-1}(w)). \qquad (1.11)$$

Here, $L_f^\star(X) = L_{\overline{f}}(X^*)$, the adjoint operator of $L_f(X)$ with respect to the inner product $\langle X, Y\rangle = \operatorname{trace}(Y^* X)$ on $\mathbb{C}^{n\times n}$, and $\overline{f}(z) = \overline{f(\overline{z})}$. If $f : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$, $\overline{f} = f$ and hence $L_f^\star(X) = L_f(X^*)$.

The following algorithm is essentially the usual power method applied to $K_f(X)$ implicitly by exploiting the relations (1.10) and (1.11) [31, Sec. 3.4], [42].

Algorithm 1.6.9 (power method on Fréchet derivative) *Given $X \in \mathbb{C}^{n\times n}$ and the Fréchet derivative L of a function f, this algorithm uses the power method to produce an estimate $\eta \le \|L_f(X)\|_F$.*

```
Choose a nonzero starting matrix $Z_0 \in \mathbb{C}^{n\times n}$.
for $k = 0\colon\infty$
    $W_{k+1} = L_f(X, Z_k)$
    $Z_{k+1} = L_f^\star(X, W_{k+1})$
    $\eta_{k+1} = \|Z_{k+1}\|_F / \|W_{k+1}\|_F$
    if converged, $\eta = \eta_{k+1}$, quit, end
end
```

This is an elegant approach to estimate the norm of the Fréchet derivative. However, it lacks a starting matrix and a convergence test and because of its linear convergence rate the number of iteration required is unpredictable. The LAPACK matrix norm estimator applying a 1-norm variant of the power method overcomes these difficulties. It has advantages over the usual power method underlying Algorithm 1.6.9 in terms of having a "built-in" starting matrix, convergence test, and a more predictable number of iterations. The next algorithm forms the basis of all the condition number estimation in LAPACK and is used in the MATLAB function `rcond` [29, Sec. 26.3].

Algorithm 1.6.10 ([31, Alg. 3.21]) *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes γ and $v = Aw$ such that $\gamma \le \|A\|_1$ with $\gamma = \|v\|_1/\|w\|_1$. For $z \in \mathbb{C}$, define* sign *as* $\mathrm{sign}(z) = z/|z|$ *and* $\mathrm{sign}(0) = 1$.

```
 1  $v = A(n^{-1}e)$
 2  if $n = 1$, quit with $\gamma = |v_1|$, end
 3  $\gamma = \|v\|_1, \quad \xi = \mathrm{sign}(v), \quad x = A^*\xi$
 4  $k = 2$
 5  repeat
 6     $j = \min\{\, i\colon |x_i| = \|x\|_\infty \,\}$
 7     $v = Ae_j, \quad \overline{\gamma} = \gamma, \quad \gamma = \|v\|_1$
 8     if ($A$ is real and $\mathrm{sign}(v) = \pm\xi$) or $\gamma \le \overline{\gamma}$, goto line 12, end
 9     $\xi = \mathrm{sign}(v), \quad x = A^*\xi$
10     $k \leftarrow k + 1$
11  until ($\|x\|_\infty = x_j$ or $k > 5$)
12  $x_i = (-1)^{i+1}(1 + \frac{i-1}{n-1}), \quad i = 1 : n$
13  $x = Ax$
14  if $2\|x\|_1/(3n) > \gamma$
15     $v = x, \quad \gamma = 2\|x\|_1/(3n)$
16  end
```

For increased reliability and efficiency, Higham and Tisseur [34] generalize Algorithm 1.6.10 to a block algorithm iterating with an $n \times t$ matrix, where $t \ge 1$; for $t = 1$, Algorithm 1.6.10 (without lines 12–16) is recovered. The MATLAB function `normest1` implements [34, Alg. 2.4] and we use it for estimating matrix 1-norms throughout this thesis. In spite of the advantages of 1-norm estimators, there is no analogue of (1.10) for the 1-norm, but the next lemma shows how $\|K_f(X)\|_1$ compares with $\|L_f(X)\|_1$.

Lemma 1.6.11 ([31, Lemma 3.18]) *For $X \in \mathbb{C}^{n\times n}$ and any Fréchet differentiable function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$,*

$$\frac{\|L_f(X)\|_1}{n} \le \|K_f(X)\|_1 \le n\|L_f(X)\|_1. \qquad \square \tag{1.12}$$

We can apply [34, Alg. 2.4] implicitly on $\|K_f(X)\|_1$ using (1.11).

Algorithm 1.6.12 (block 1-norm estimator for Fréchet derivative) *Given a matrix $X \in \mathbb{C}^{n\times n}$ this algorithm uses a block 1-norm estimator to produce an estimate η of $\|L_f(X)\|_1$, given the ability to compute $L_f(X,E)$ and $L_f^\star(X,E)$ for any E. More precisely, $\eta \le \|K_f(X)\|_1$, where $\|K_f(X)\|_1$ satisfies (1.12).*

```
Apply Algorithm 2.4 from Higham and Tisseur [34] with parameter $t = 2$ to
  the Kronecker matrix representation $K_f(X)$ of $L_f(X)$, making use of
  the relations in (1.11).
```

Key properties of Algorithm 1.6.12 are that it typically requires about $4t$ Fréchet derivative evaluations and it almost invariably produces an estimate of $\|K_f(X)\|_1$ correct to within a factor 3. A factor n of uncertainty is added when we take η as an estimate of $\|L_f(X)\|_1$. Overall, the algorithm is a very reliable means of estimating $\|L_f(X)\|_1$ to within a factor $3n$.

Chapter 2

A New Scaling and Squaring algorithm for the Matrix Exponential

2.1 Introduction

The scaling and squaring method is the most popular method for computing the matrix exponential. It is used, for example, in Mathematica (function `MatrixExp`), MATLAB (function `expm`), SLICOT (subroutine MB05OD) [65], and the Expokit package [63]. It is also used in more general contexts, such as for computing the group exponential of a diffeomorphism [8]. The method is based on the approximation

$$e^A = (e^{2^{-s}A})^{2^s} \approx r_m(2^{-s}A)^{2^s}, \tag{2.1}$$

where r_m is the $[m/m]$ Padé approximant to e^x and the nonnegative integers m and s are chosen in a prescribed way that aims to achieve full machine accuracy at minimal cost. The $[k/m]$ Padé approximants $r_{km}(x) = p_{km}(x)/q_{km}(x)$ to the exponential function are known explicitly for all k and m:

$$p_{km}(x) = \sum_{j=0}^{k} \frac{(k+m-j)!\,k!}{(k+m)!\,(k-j)!}\frac{x^j}{j!}, \quad q_{km}(x) = \sum_{j=0}^{m} \frac{(k+m-j)!\,m!}{(k+m)!\,(m-j)!}\frac{(-x)^j}{j!}. \tag{2.2}$$

Note that $p_{km}(x) = q_{mk}(-x)$, which reflects the property $(e^x)^{-1} = e^{-x}$ of the exponential function [30].

The method behaves reliably in floating point arithmetic across a wide range of matrices, but does have a weakness manifested in a subtle phenomenon known as *overscaling*, in which a large $\|A\|$ causes a larger than necessary s to be chosen, with a harmful effect on accuracy. We illustrate the phenomenon with the matrix

$$A = \begin{bmatrix} 1 & b \\ 0 & -1 \end{bmatrix}, \qquad e^A = \begin{bmatrix} e & \frac{b}{2}(e-e^{-1}) \\ 0 & e^{-1} \end{bmatrix}. \tag{2.3}$$

Our computations are carried out in MATLAB 7.10 (R2010a), which uses IEEE double precision arithmetic with unit roundoff $u = 2^{-53} \approx 1.1 \times 10^{-16}$. We computed the exponential of A using `expm`, which implements the algorithm of Higham [30],

Table 2.1: Errors and condition numbers for A in (2.3) and $B = Q^*AQ$. The columns headed "s" show the values of s used by `expm` to produce the results in the previous column. The superscripts † and ‡ denote that a particular choice of s was forced: $s = 0$ for † and the $s \in [0, 25]$ giving the most accurate result for ‡.

b	`expm`(A)	s	`expm`$(A)^\dagger$	`funm`(A)	`expm`(B)	s	`expm`$(B)^\ddagger$	s	`funm`(B)	$\kappa_{\exp}(A)$
10^3	1.7e-15	8	1.9e-16	1.9e-16	2.8e-12	8	2.6e-13	4	2.9e-14	1.6e5
10^4	1.8e-13	11	7.6e-20	3.8e-20	4.0e-8	12	1.9e-10	1	4.1e-10	1.6e7
10^5	7.5e-13	15	1.2e-16	1.2e-16	2.2e-5	15	5.0e-7	4	1.3e-8	1.6e9
10^6	1.3e-11	18	2.0e-16	2.0e-16	8.3e-4	18	7.5e-6	13	7.5e-8	1.6e11
10^7	7.2e-11	21	1.6e-16	1.6e-16	1.2e2	22	6.9e-1	14	6.2e-4	1.6e13
10^8	3.0e-12	25	1.3e-16	1.3e-16	4.4e37	25	1.0e0	3	6.3e-2	1.6e15

and `funm`, which is applicable to general matrix functions and implements the Schur–Parlett method of Davies and Higham [13], [31, Sec. 10.4.3]. For b ranging from 10^3 to 10^8, the normwise relative errors in the Frobenius norm are shown in the columns of Table 2.1 headed "`expm`(A)" and "`funm`(A)". We see that while `funm` provides full accuracy in every case, the accuracy for `expm` deteriorates with increasing b. As b increases so does the chosen s in (2.1), with m equal to 13 in each case, which is the maximum value that `expm` allows. For $b = 10^8$ the diagonal elements of e^A are approximated by $r_m(x)^{2^{25}} \approx \big((1 + x/2)/(1 - x/2)\big)^{2^{25}}$ with $x = \pm 2^{-25} \approx \pm 10^{-8}$. Approximately half the digits of x are lost in forming $1 \pm x$ and the repeated squarings can only amplify the loss. The essential problem is loss of significance due to too large a choice of s. If we force `expm` to take smaller values of s (still with $m = 13$) we find that the accuracy of the computed exponential increases as s decreases, until a result correct to full machine precision is obtained for $s = 0$, as shown in the column of the table headed "`expm`$(A)^\dagger$". Note that $s = 0$ corresponds to disregarding the large values of a_{12} completely.

To gain some more insight we note the following expression for the exponential of a block 2×2 block triangular matrix (see, e.g., [31, Prob. 10.12], [70]):

$$\exp\left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}\right) = \begin{bmatrix} e^{A_{11}} & \displaystyle\int_0^1 e^{A_{11}(1-s)} A_{12} e^{A_{22}s}\, ds \\ 0 & e^{A_{22}} \end{bmatrix}. \tag{2.4}$$

Note that A_{12} appears only in the (1,2) block of e^A, where it enters *linearly*. This suggests that the approximation procedure for e^A should be unaffected by $\|A_{12}\|$ and should depend only on $\|A_{11}\|$ and $\|A_{22}\|$. And if A_{11} and A_{22} are upper triangular this argument can be recurred to reason that only the diagonal elements of A should influence the parameters m and s. The need for exponentials of triangular matrices does arise in practice, for example in the solution of radioactive decay equations [55], [74].

In practice A is most often full, rather than (block) triangular. Of course a full matrix A can be reduced by unitary similarities to a triangular matrix T via a Schur decomposition, but applying the scaling and squaring method to T is not numerically equivalent to applying it to A. To investigate possible overscaling for full matrices we repeated the experiment above using $B = Q^*AQ$, where A is as in (2.3) and Q

is a random orthogonal matrix, different for each b. The relative normwise errors are shown in the second group of columns of Table 2.1. We see that both expm and funm produce errors increasing with b, those from expm being somewhat larger. The column headed " expm$(B)^{\ddagger}$" shows that by forcing an optimal choice of s, expm can be made significantly more accurate, so expm is again suffering from overscaling.

To determine whether the computed results from expm are acceptable we need to know the condition number of the problem, which is

$$\kappa_{\exp}(A) := \lim_{\epsilon \to 0} \sup_{\|E\| \le \epsilon \|A\|} \frac{\|e^{A+E} - e^A\|}{\epsilon \|e^A\|}. \tag{2.5}$$

We evaluated this condition number in the Frobenius norm (for which $\kappa_{\exp}(A) = \kappa_{\exp}(B)$) using a combination of Algorithm 1.6.8 and Algorithm 4.6.3, implemented in a modified version of expm_cond from the Matrix Function Toolbox [26]. The results are shown in the final column of Table 2.1. For a stable method we expect an error bounded by a modest multiple of $\kappa_{\exp}(A)u$. Thus funm is performing stably but expm is behaving unstably, especially for $b = 10^7, 10^8$.

For the original A, the errors for expm are all substantially less than $\kappa_{\exp}(A)u$, but of course this condition number allows arbitrary perturbations and so is not appropriate for this triangular A. For structured condition numbers for (block) triangular matrices see Dieci and Papini [18].

Our simple 2×2 example reveals two things. First, that for triangular matrices overscaling can happen because of large off-diagonal elements. Second, that for full matrices overscaling is also possible and may cause unstable behavior of the scaling and squaring method.

The goal of this work is to modify the scaling and squaring method in order to overcome the overscaling problem. To this end we employ two novel ideas, one specific to triangular matrices and one applying to general matrices.

Triangular matrices. For the triangular matrix (2.3) we noted that the diagonal elements are calculated inaccurately by expm for large $|b|$. A simple solution is to replace the diagonal elements of the computed exponential by $e^{a_{ii}}$. To benefit the off-diagonal elements as well, we can replace the values $r_m(2^{-s}a_{ii})^{2^j}$ in the squaring phase by $e^{2^{j-s}a_{ii}}$, thereby attenuating the propagation of errors.

Full matrices. For full matrices we introduce a new way of sharpening the truncation error bounds that are used in the derivation of the method. This allows the method to take a potentially smaller s, and hence evaluate the Padé approximant at a larger-normed matrix and require fewer squarings. We will argue that the sharpening is likely to have a particularly beneficial effect when overscaling is possible.

Our key idea is to exploit the sequence $\{\|A^k\|^{1/k}\}$. It is easy to see that

$$\rho(A) \le \|A^k\|^{1/k} \le \|A\|, \qquad k = 1\colon \infty, \tag{2.6}$$

where ρ is the spectral radius, and moreover $\|A^k\|^{1/k} \to \rho(A)$ as $k \to \infty$ [36, Cor. 5.6.14]. Figure 2.1 plots the sequence $\{\|A^k\|_2^{1/k}\}_{k=1}^{20}$ for 54 nonnormal 16×16 matrices A, normalized (without loss of generality) so that $\|A\|_2 = 1$, drawn from the MATLAB function gallery function, from the Matrix Computation Toolbox [25], and from the e^A literature. We see that typically the sequence is decreasing, although very non-monotonic behavior is possible. It is this decrease that we will exploit.

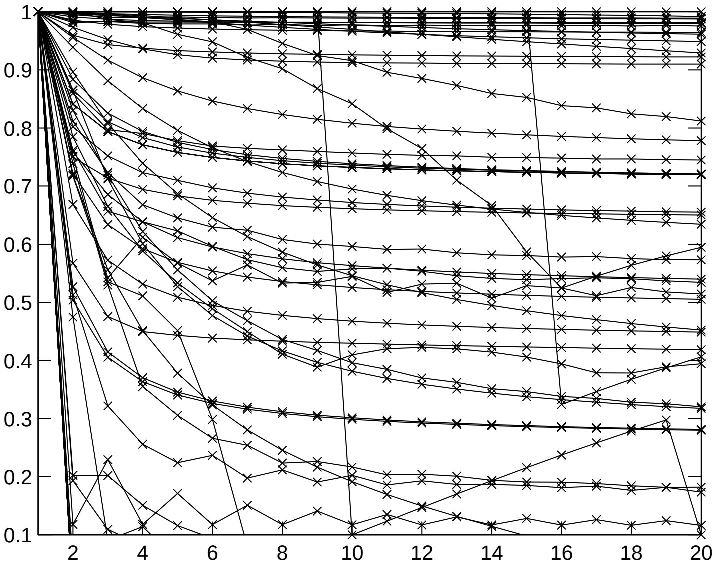

Figure 2.1: $\{\|A^k\|_2^{1/k}\}_{k=1}^{20}$ for 54 16×16 matrices A with $\|A\|_2 = 1$.

In the derivation of the scaling and squaring algorithm of [30] a power series

$$h_\ell(x) = \sum_{k=\ell}^{\infty} c_k\, x^k$$

has to be bounded at the matrix argument A (or, more precisely, $2^{-s}A$, but we drop the scale factor for now), where $\ell = 2m+1$. The bound used previously is [30], [31, Sec. 10.3]

$$\|h_\ell(A)\| \le \sum_{k=\ell}^{\infty} |c_k|\, \|A\|^k. \tag{2.7}$$

The following theorem provides a sharper bound.

Theorem 2.1.1 *Let $h_\ell(x) = \sum_{k=\ell}^{\infty} c_k\, x^k$ be a power series with radius of convergence ω, and let $\widetilde{h}_\ell(x) = \sum_{k=\ell}^{\infty} |c_k| x^k$. For any $A \in \mathbb{C}^{n\times n}$ with $\rho(A) < \omega$ we have*

$$\|h_\ell(A)\| \le \widetilde{h}_\ell(\|A^t\|^{1/t}), \tag{2.8}$$

where $\|A^t\|^{1/t} = \max\{\|A^k\|^{1/k} : k \ge \ell \text{ and } c_k \ne 0\}$.

Proof. The existence of t is guaranteed since the sequence $\{\|A^k\|^{1/k}\}$ is bounded above and convergent, as noted above. We have

$$\begin{aligned}\|h_\ell(A)\| \le \sum_{k=\ell}^{\infty} |c_k|\, \|A^k\| &= \sum_{k=\ell}^{\infty} |c_k|\, \left(\|A^k\|^{1/k}\right)^k \\ &\le \sum_{k=\ell}^{\infty} |c_k|\, \left(\|A^t\|^{1/t}\right)^k = \widetilde{h}_\ell(\|A^t\|^{1/t}). \qquad \square\end{aligned}$$

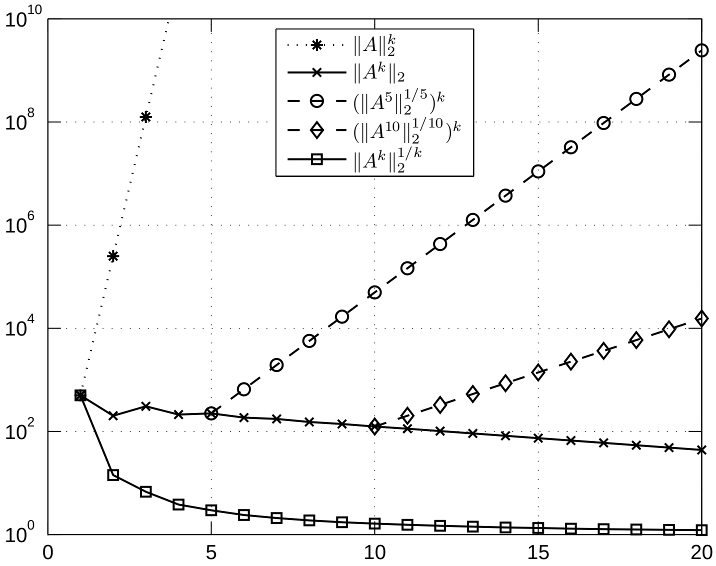

Figure 2.2: For the 2×2 matrix A in (2.10), $\|A^k\|_2$ and various bounds for $k = 1\colon 20$.

The bound (2.8) is clearly sharper than (2.7) since $\|A^t\|^{1/t} \le \|A\|$, and it can be arbitrarily smaller. In particular, if $A \ne 0$ is nilpotent and $\ell \ge n$ then the bound (2.8) is zero while (2.7) is nonzero unless $h_\ell(x) \equiv 0$. Note that as essentially a special case of Theorem 2.1.1, if the sequence $\{\|A^k\|\}_{k \ge i}$ is nonincreasing then

$$\|A^k\| \le \left(\|A^i\|^{1/i}\right)^k, \qquad k \ge i. \tag{2.9}$$

To see why (2.8) may help to avoid overscaling, consider the matrix

$$A = \begin{bmatrix} 0.9 & 500 \\ 0 & -0.5 \end{bmatrix}, \tag{2.10}$$

for which the 2-norms of the powers of A decay monotonically for $k \ge 5$, despite the large (1,2) element. Figure 2.2 plots $\|A^k\|_2$, the crude bound $\|A\|_2^k$, and the more refined bounds $(\|A^5\|_2^{1/5})^k$ valid for $k \ge 5$ and $(\|A^{10}\|_2^{1/10})^k$ valid for $k \ge 10$, by (2.9). The crude bound is an extreme overestimate and the refined bounds are a significant improvement. The reason for the improvement is that when A is powered the large (1,2) element multiplies the diagonal elements and there is both multiplicative and subtractive cancellation, resulting in little or no growth. The refined bounds take advantage of this. For the power series h_ℓ, such a reduction in the bound for $\|A^k\|$ translates into a reduction in the bound for $h_\ell(A)$, and this in turn can lead to a much smaller s being chosen in the scaling and squaring method.

In essence, what we are doing is using the behavior of the first few powers of A to extract information about the non-normality of A. In the scaling and squaring method we will exploit the powers that must be computed anyway during the course of the algorithm, thereby gaining potentially improved accuracy and reduced cost. This idea has already been used in an ad hoc way by Hargreaves and Higham [24], who in the context of computing the matrix cosine express error bounds in terms of $\|A^2\|^{1/2}$ instead of $\|A\|$, but here we are exploiting the idea more systematically.

This chapter is organized as follows. We begin, in Section 2.2, by showing how to improve the squaring phase of the scaling and squaring method for triangular matrices. In Section 2.3 we summarize the scaling and squaring algorithm of Higham. Section 2.4 presents new bounds for norms of matrix powers that are then exploited in Section 2.5 to produce a new algorithm that is often more accurate, more efficient, or both. Numerical experiments that illustrate the benefits of the new algorithm are given in Section 2.6. Section 2.7 presents a comparison between the efficiency of diagonal Padé approximants and Taylor approximants within the scaling and squaring method for the matrix exponential.

Finally, we make connections with earlier work. The overscaling phenomenon was first identified by Kenney and Laub [44]. It was later analyzed by Dieci and Papini [17] for the case of block 2×2 block upper triangular matrices $(A_{ij})_{i,j=1}^{2}$. The latter analysis suggests that if the scaling and squaring method is used with s determined so that $2^{-s}\|A_{11}\|$ and $2^{-s}\|A_{22}\|$ are appropriately bounded, without consideration of $\|A_{12}\|$, then an accurate approximation to e^{A} will still be obtained. However, no algorithm for general A was proposed in [17].

2.2 Squaring phase for triangular matrices

Our new approach for triangular matrices was inspired by the practical observation that the scaling and squaring method seems to be immune to overscaling for nilpotent triangular matrices T—those with zero diagonal elements. Indeed, the diagonal entries of $r_m(2^{-s}T)$ are correctly computed as ones and remain as ones through the squaring process. Now for a general upper triangular matrix T, the scaling and squaring method computes $r_m(2^{-s}T) =: D_s + F_s$, where D_s is diagonal and F_s is strictly upper triangular, and then forms

$$D_{i-1} + F_{i-1} = (D_i + F_i)^2, \qquad i = s : -1 : 1,$$

after which $D_0 + F_0 \approx e^{T}$. Hence we have the recurrence

$$\left.\begin{aligned} D_{i-1} &= D_i^2 \\ F_{i-1} &= D_i F_i + F_i D_i + F_i^2 \end{aligned}\right\} \quad i = s : -1 : 1. \tag{2.11}$$

Clearly, errors in the computation of the D_i propagate into the off-diagonals contained in F_{i-1}. Indeed a single error ϵI (say) in D_i transmits into F_{i-1} as $2\epsilon F_i$ and into D_{i-1} as $2\epsilon D_i$, so there is potential exponential error growth. We can virtually remove errors in the diagonal and thereby attenuate the overall error growth by computing $D_i = \exp(2^{-i}\mathrm{diag}(T))$ at each stage instead of computing $D_s = r_m(2^{-s}\mathrm{diag}(T))$ and then repeatedly squaring. Thus the final steps of the scaling and squaring method are rewritten as follows.

Code Fragment 2.2.1

```
Form $X = r_m(2^{-s}T)$. % First phase of method (unchanged).
Replace $\mathrm{diag}(X)$ by $\exp(2^{-s}\mathrm{diag}(T))$.
for $i = s - 1 \colon -1 \colon 0$
    $X \leftarrow X^2$
```

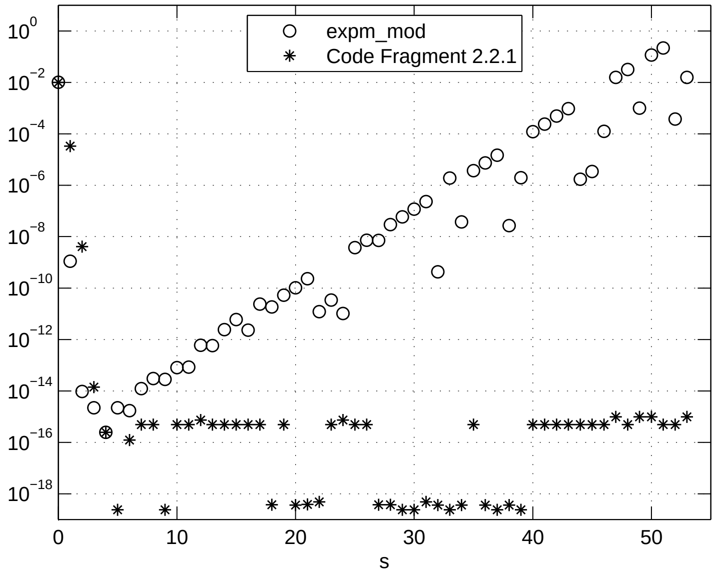

Figure 2.3: Relative errors from Code Fragment 2.2.1 and `expm_mod` for a single 8×8 matrix with $s = 0\colon 53$.

```
5     Replace diag($X$) by exp($2^{-i}$diag($T$)).
6     Replace (first) superdiagonal of $X$ by explicit formula
      for superdiagonal of exp($2^{-i}T$) from [31, eq. (10.42)].
7  end
```

Note that we have gone further in line 6 by computing the correct superdiagonal as well, since it is available from an accurate formula at negligible cost.

We give a numerical example to illustrate the benefits of this approach. We take the matrix formed by the MATLAB code, with $n = 8$,

```
T = gallery('triw',n,-1);  T(1,n) = 1e4; T(1:n+1:n^2) = -(1:n).^2
```

The MATLAB function `expm` chooses $s = 11$ and $m = 13$ for this matrix and produces a relative error 8.4×10^{-14}. For s from 0 to 53 we compare Code Fragment 2.2.1, using $m = 13$, with a modified version `expm_mod` of `expm` that accepts a user-specified choice of s. The normwise relative errors for both methods are plotted in Figure 2.3. The optimum value of s for `expm_mod` is 4, for which a relative error 4.9×10^{-16} is achieved; as s increases the relative error deteriorates rapidly until it reaches 1 at $s = 53$. However, Code Fragment 2.2.1 remains fully accurate for all $s \geq 4$, showing the effectiveness of the strategy of injecting the correct diagonal into the recurrence.

This approach can be extended to quasi-triangular matrices T, which are block triangular matrices whose diagonal blocks T_{ii} are 1×1 or 2×2. Such T arise in the real Schur decomposition of a real matrix, in which case the 2×2 blocks T_{ii} have distinct eigenvalues that are nonreal complex conjugates. We need to compute the exponentials of the diagonal blocks

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

which we assume have distinct eigenvalues λ_1, λ_2. From the general formula $f(A) = f(\lambda_1)I + f[\lambda_1, \lambda_2](A - \lambda_2 I)$, where $f[\lambda_1, \lambda_2]$ is a divided difference [31, Prob. 1.9], we obtain

$$e^A = \frac{e^{\lambda_1} - e^{\lambda_2}}{\lambda_1 - \lambda_2} A + \left(e^{\lambda_1} - \lambda_2 \frac{e^{\lambda_1} - e^{\lambda_2}}{\lambda_1 - \lambda_2} \right) I.$$

The eigenvalues of A are $(a+d)/2 \pm \mu$, where $\mu = \frac{1}{2}\sqrt{(a-d)^2 + 4bc}$. After some manipulation we obtain

$$e^A = e^{(a+d)/2} \begin{bmatrix} \cosh(\mu) + \frac{1}{2}(a-d)\,\mathrm{sinch}(\mu) & b\,\mathrm{sinch}(\mu) \\ c\,\mathrm{sinch}(\mu) & \cosh(\mu) - \frac{1}{2}(a-d)\,\mathrm{sinch}(\mu) \end{bmatrix}, \qquad (2.12)$$

where

$$\mathrm{sinch}(x) = \begin{cases} \sinh(x)/x, & x \neq 0, \\ 1, & x = 0. \end{cases}$$

This formula is not always evaluated to high relative accuracy, so the use of extra precision in its evaluation might be justified.

Combining this formula with an initial real Schur decomposition we have the following outline applicable to any $A \in \mathbb{R}^{n \times n}$.

Code Fragment 2.2.2

```
Compute the real Schur decomposition, $A = QTQ^*$,
with block $q \times q$ upper quasi-triangular $T = (T_{ij})$.
Form $X = r_m(2^{-s}T)$.
Replace diag($X_{ii}$) by exp($2^{-s}$diag($T_{ii}$)), $i = 1\colon q$, using (2.12).
for $i = s-1\colon -1\colon 0$
    $X \leftarrow X^2$
    Replace diag($X_{ii}$) by exp($2^{-i}$diag($T_{ii}$)), $i = 1\colon q$, using (2.12).
end
$X \leftarrow QXQ^*$
```

Note that this approach has the advantage that it works entirely in real arithmetic, in contrast to the Schur–Parlett method specialized to the exponential, which necessarily uses the complex Schur form [13], [31, Sec. 10.4.3].

Our main interest is in improving the scaling and squaring method for full matrices without using a Schur decomposition. In the next section we summarize the derivation of the existing algorithm on which we will work.

2.3 The existing scaling and squaring algorithm

In this section we review the scaling and squaring algorithm of Higham [30]. Write the $[m/m]$ Padé approximant of e^x as $r_m(x) = p_m(x)/q_m(x)$. We will later need the properties that p_m has positive coefficients and $q_m(x) = p_m(-x)$.

Let $\nu_m = \min\{\, |t| : q_m(t) = 0 \,\}$ and

$$\Omega_m = \{\, X \in \mathbb{C}^{n \times n} : \rho(e^{-X} r_m(X) - I) < 1 \text{ and } \rho(X) < \nu_m \,\}. \qquad (2.13)$$

Table 2.2: Parameters θ_m needed in Algorithm 2.3.1 and Algorithm 2.5.1 and upper bounds for $\kappa_A(q_m(A))$.

m	θ_m	$\kappa_A(q_m(A))$
3	1.495585217958292e-2	1.0e0
5	2.539398330063230e-1	1.3e0
7	9.504178996162932e-1	2.6e0
9	2.097847961257068e0	8.2e0
13 (Alg. 2.3.1)	5.371920351148152e0	2.2e2
13 (Alg. 2.5.1)	4.25	7.1e1

The functions

$$h_{2m+1}(X) = \log(e^{-X} r_m(X)) \tag{2.14}$$

are defined for $X \in \Omega_m$, where log denotes the principal logarithm, and so for $X \in \Omega_m$ we have $r_m(X) = e^{X+h_{2m+1}(X)}$. Now choose s so that $2^{-s}A \in \Omega_m$. Then

$$r_m(2^{-s}A)^{2^s} = e^{A+2^s h_{2m+1}(2^{-s}A)} =: e^{A+\Delta A} \tag{2.15}$$

and the matrix $\Delta A = 2^s h_{2m+1}(2^{-s}A)$ represents the backward error resulting from the approximation of e^A by the scaling and squaring method. Over Ω_m, the functions h_{2m+1} have a power series expansion

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

Higham [30] employs a variant of the bound

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{2m+1}(2^{-s}A)\|}{\|2^{-s}A\|} \leq \frac{\widetilde{h}_{2m+1}(\|2^{-s}A\|)}{\|2^{-s}A\|}, \tag{2.16}$$

where $\widetilde{h}_{2m+1}(x) = \sum_{k=2m+1}^{\infty} |c_k| x^k$. For $m = 1\colon 21$, he uses high precision arithmetic to compute the values

$$\theta_m = \max\{\, \theta : \widetilde{h}_{2m+1}(\theta)/\theta \leq u \,\}, \tag{2.17}$$

some of which are listed in Table 2.2. He finds that $\theta_m < \nu_m$. Thus in *exact* arithmetic, $\|\Delta A\| \leq u\|A\|$ if s is chosen so that $\|2^{-s}A\| \leq \theta_m$ and $r_m(2^{-s}A)^{2^s}$ is used to approximate e^A. Higham's cost analysis eliminates even degrees of Padé approximants and reveals that $m = 13$ is the optimal degree to use when scaling is required. When $\|A\| \leq \theta_{13}$, his algorithm chooses the first $m \in \{3, 5, 7, 9, 13\}$ such that $\|A\| \leq \theta_m$.

For $m = 3, 5, 7, 9$, Higham uses the evaluation scheme

$$p_m(A) = A \sum_{k=0}^{(m-1)/2} b_{2k+1} A_{2k} + \sum_{k=0}^{(m-1)/2} b_{2k} A_{2k} =: (u_m + v_m)(A), \tag{2.18}$$

where $A_{2k} = A^{2k}$. For $m = 13$, he uses the scheme

$$\begin{aligned} p_{13}(A) &= A\big[A_6(b_{13}A_6 + b_{11}A_4 + b_9A_2) + b_7A_6 + b_5A_4 + b_3A_2 + b_1I\big] \qquad (2.19)\\ &\quad + A_6(b_{12}A_6 + b_{10}A_4 + b_8A_2) + b_6A_6 + b_4A_4 + b_2A_2 + b_0I \\ &=: (u_{13} + v_{13})(A), \end{aligned}$$

where $A_2 = A^2$, $A_4 = A_2^2$, and $A_6 = A_2A_4$. Since $q_m(A) = p_m(-A)$, we have $q_m(A) = (-u_m + v_m)(A)$ and $r_m(A)$ is obtained from the equation

$$(-u_m + v_m)(A)r_m(A) = (u_m + v_m)(A). \tag{2.20}$$

The algorithm of Higham [30], which forms the basis of the MATLAB function `expm`, can be summarized as follows.

Algorithm 2.3.1 *This algorithm evaluates the matrix exponential $X = e^A$ of $A \in \mathbb{C}^{n\times n}$ by the scaling and squaring method. It uses the parameters θ_m given in Table* 2.2. *The algorithm is intended for IEEE double precision arithmetic.*

```
for $m = [3\ 5\ 7\ 9]$
    if $\|A\|_1 \le \theta_m$, evaluate $X = r_m(A)$ using (2.18) and (2.20), quit, end
end
$A \leftarrow 2^{-s}A$ with $s = \lceil \log_2(\|A\|_1/\theta_{13}) \rceil$
Evaluate $r_{13}(A)$ using (2.19) and (2.20).
$X = r_{13}(A)^{2^s}$ by repeated squaring.
```

Cost: $(\pi_m + s)M + D$, where m is the degree of Padé approximant used and π_m (tabulated in [30, Table 2.2]) is the cost of evaluating p_m and q_m.

2.4 Practical bounds for norm of matrix power series

The bound for $\|h_\ell(A)\|$ in Theorem 2.1.1 is not readily usable in the form stated since it employs $\|A^t\|^{1/t}$ and we will rarely know the value of t. We now develop a more convenient form of bound that will be used in the next section to improve Algorithm 2.3.1. We denote by $\mathbb{N}$ the set of positive integers.

Lemma 2.4.1 *For any $k \ge 1$ such that $k = pm_1 + qm_2$ with $p, q \in \mathbb{N}$ and $m_1, m_2 \in \mathbb{N} \cup \{0\}$,*

$$\|A^k\|^{1/k} \le \max\big(\|A^p\|^{1/p}, \|A^q\|^{1/q}\big).$$

Proof. Let $\delta = \max(\|A^p\|^{1/p}, \|A^q\|^{1/q})$. The bound follows from the inequality

$$\begin{aligned}\|A^k\| &\le \|A^{pm_1}\|\,\|A^{qm_2}\| \\ &\le \big(\|A^p\|^{1/p}\big)^{pm_1}\big(\|A^q\|^{1/q}\big)^{qm_2} \\ &\le \delta^{pm_1}\delta^{qm_2} = \delta^k. \qquad \square\end{aligned}$$

Theorem 2.4.2 *Define h_ℓ and $\widetilde{h}_\ell$ as in Theorem* 2.1.1 *and suppose $\rho(A) < \omega$ and $p \in \mathbb{N}$. Then*

(a) $\|h_\ell(A)\| \le \widetilde{h}_\ell\big(\max(\|A^p\|^{1/p}, \|A^{p+1}\|^{1/(p+1)})\big)$ *if $\ell \ge p(p-1)$.*

(b) $\|h_\ell(A)\| \le \widetilde{h}_\ell\big(\max(\|A^{2p}\|^{1/(2p)}, \|A^{2p+2}\|^{1/(2p+2)})\big)$ *if $\ell \ge 2p(p-1)$ and h_ℓ is even.*

Proof. For the first part, let $X_p = \{\, m \in \mathbb{N} : m \ge p(p-1) \,\}$ and $Y_p = \{\, (p+1)m_1 + pm_2 : m_1, m_2 \in \mathbb{N} \cup \{0\} \,\}$. We have $X_p \subset Y_p$ since any element $k \in X_p$ can be written as $k = pq + r$ with $q \ge p-1$ and $0 \le r < p$. Then $k = (p+1)r + p(q-r) \in Y_p$, since $q - r \ge p - r - 1 \ge 0$. Hence by Lemma 2.4.1 we have

$$\|A^k\|^{1/k} \le \max\{\, \|A^{p+1}\|^{1/(p+1)}, \|A^p\|^{1/p} \,\}, \quad k \ge p(p-1),$$

and the result follows from Theorem 2.1.1. Part (b) follows similarly from the relations

$$\begin{aligned} \{\, \ell : \ell \text{ even}, \ \ell \ge 2p(p-1) \,\} &= 2X_p \subset 2Y_p \\ &= \{\, (2p+2)m_1 + 2pm_2 : m_1, m_2 \in \mathbb{N} \cup \{0\} \,\}. \end{aligned}$$ □

To illustrate Theorem 2.4.2, suppose $\ell = 12$. In view of the inequality $\|A^{2k}\|^{1/(2k)} \le \|A^k\|^{1/k}$, the p that minimizes $\max(\|A^p\|^{1/p}, \|A^{p+1}\|^{1/(p+1)})$ subject to $\ell \ge p(p-1)$ is either $p = 3$ or $p = 4$. So the first part of the theorem gives

$$\|h_{12}(A)\| \le \widetilde{h}_{12}\bigl(\min\bigl(\max(\|A^3\|^{1/3}, \|A^4\|^{1/4}), \max(\|A^4\|^{1/4}, \|A^5\|^{1/5})\bigr)\bigr). \tag{2.21}$$

If h_{12} is even we can apply the second part of the theorem, for which the optimal choice of p is 3, which gives

$$\|h_{12}(A)\| \le \widetilde{h}_{12}\bigl(\max(\|A^6\|^{1/6}, \|A^8\|^{1/8})\bigr). \tag{2.22}$$

For a normal matrix and the 2-norm these upper bounds are both identical to the bound $h_{12}(\|A\|_2)$, but for nonnormal matrices they can be significantly smaller. For A in (2.10), we have $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(\|A\|_2) \approx \widetilde{h}_{12}(500)$, but $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(3.82)$ from (2.21) and $\|h_{12}(A)\|_2 \le \widetilde{h}_{12}(2.39)$ from (2.22), demonstrating the benefit of exploiting the structure of h_{12} as an even function. Figure 2.1 suggests that it will typically be the case that (2.22) is sharper than (2.21).

2.5 New algorithm

We now derive a new algorithm that builds on Algorithm 2.3.1. Our development focuses on increasing the sharpness of the inequality in (2.16) by using the bounds of the previous section.

The function h_{2m+1} in (2.14) is odd, which follows from the fact that Padé approximants to the exponential function satisfy $r_m(-x) = (r_m(x))^{-1}$ [31, Sec. 10.3]:

$$\begin{aligned} h_{2m+1}(-x) &= \log(e^x r_m(-x)) \\ &= \log((e^{-x} r_m(x))^{-1}) \\ &= -\log(e^{-x} r_m(x)) = -h_{2m+1}(x). \end{aligned}$$

Therefore for $X \in \Omega_m$ we can write

$$h_{2m+1}(X) = X \sum_{k=2m}^{\infty} c_{k+1} X^k =: X\phi_{2m}(X),$$

where the ϕ_{2m} are even functions. Let $\widetilde{\phi}_{2m}(x) = \sum_{k=2m}^{\infty} |c_{k+1}| x^k$. We can now refine the bound in (2.16) using Theorem 2.4.2(b):

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{2m+1}(2^{-s}A)\|}{\|2^{-s}A\|} = \frac{\|2^{-s}A\phi_{2m}(2^{-s}A)\|}{\|2^{-s}A\|} \tag{2.23a}$$
$$\le \|\phi_{2m}(2^{-s}A)\| \le \widetilde{\phi}_{2m}(2^{-s}\alpha_p(A)),$$

where

$$\alpha_p(A) = \max\bigl(\|A^{2p}\|^{1/(2p)}, \|A^{2p+2}\|^{1/(2p+2)}\bigr) \tag{2.23b}$$

and we choose p to minimize $\alpha_p(A)$ subject to $2m \ge 2p(p-1)$. As we have $\widetilde{\phi}_{2m}(\theta) = \widetilde{h}_{2m+1}(\theta)/\theta$, clearly this analysis does not affect the calculation of the values θ_m in (2.17), but it does affect the choice of scaling parameter s. Whereas before, the pair (s, m) could be used if $2^{-s}\|A\| \le \theta_m$, now the requirement is only

$$2^{-s}\alpha_p(A) \le \theta_m, \tag{2.24}$$

and for a given m this is potentially satisfied with a much smaller s when A is nonnormal. A significant computational saving could therefore accrue.

At this point, we have an improved way to choose the parameters m and s, but we have not yet considered the effect of rounding errors. The analysis in [30] shows that Algorithm 2.3.1 is not unduly affected by rounding errors except, possibly, in the squaring phase. But with our more liberal choice of parameters numerical stability needs further analysis. We will combine rigorous error bounds with some heuristics in order to arrive at our final algorithm.

The main aim is to check that the evaluation of r_m is accurate in floating point arithmetic. Let $A \leftarrow 2^{-s}A$, so that A denotes the scaled matrix, and consider the evaluation of $p_m(A)$ by the schemes described in the previous section. It is shown in [30] that the computed matrix $\widehat{p}_m(A)$ satisfies

$$\|p_m(A) - \widehat{p}_m(A)\|_1 \le \widetilde{\gamma}_{mn} p_m(\|A\|_1) \lesssim \widetilde{\gamma}_{mn} \|p_m(A)\|_1 \, e^{\|A\|_1}, \tag{2.25}$$

where $\widetilde{\gamma}_k = cku/(1 - cku)$ with c a small integer constant. While this is a satisfactory bound for the values of $\|A\|$ allowed by Algorithm 2.3.1, it is not so for an algorithm based on (2.24), because $\|A\|$ can be arbitrarily large. Therefore we will use the sharper error bound [31, Thm. 4.5]

$$\max(\|p_m - \widehat{p}_m\|_1, \|q_m - \widehat{q}_m\|_1) \le \widetilde{\gamma}_{mn} \|p_m(|A|)\|_1 = \widetilde{\gamma}_{mn} \|p_m(|A|)^T e\|_\infty, \tag{2.26}$$

where $e = [1, 1, \dots, 1]^T$ and we have used the properties of p_m and q_m mentioned at the start of Section 2.3 together with the relations

$$\|B\|_1 = \|\,|B|\,\|_1 = \|\,|B|^T e\,\|_\infty. \tag{2.27}$$

The bound (2.26) can be computed in just $O(n^2)$ operations.

The a priori bound (2.26) applies to several different evaluation schemes and does not exploit the particular properties of our scheme. In particular, it contains $|A|^m$, which is clearly pessimistic since our schemes for p_m and q_m do not explicitly evaluate the mth power of A. However, it turns out that the bound is surprisingly sharp when used within our algorithm in practice. We have found that if the inequality

$$\|\widehat{p}_m(|A|)^T e\|_\infty / \min(\|\widehat{p}_m\|_1, \|\widehat{q}_m\|_1) \le ce^{\theta_m} \tag{2.28}$$

is satisfied for a suitable integer constant c then this is a reliable indicator that $\widehat{p}_m$ and $\widehat{q}_m$ have been computed to close to full precision (in other words, we can ignore the term $\widetilde{\gamma}_{mn}$ in (2.26)). We have tried the alternative of computing a running error bound (an a posteriori bound that is the smallest possible) [29, Sec. 3.3], but found that it brings no benefits.

Suppose that (2.28) is not satisfied, which suggests that the evaluation of p_m or q_m may have been insufficiently accurate. Since the weaker bound (2.25) is satisfactory for Algorithm 2.3.1, this means that $s < s_{\max}$, where $s_{\max}$ is the scaling parameter selected by Algorithm 2.3.1. We could simply revert to $s_{\max}$ and execute Algorithm 2.3.1, but instead we will use a strategy that in certain cases increases s based on the available information. One approach is to increase s so that (2.28) is satisfied. However, we have found a more heuristic approach to perform better in practice. Let A denote the original, unscaled matrix. Returning to the bound (2.23a), using $|h_{2m+1}(A)| \le \widetilde{h}_{2m+1}(|A|)$ we have

$$\begin{aligned}\frac{\|\varDelta A\|_1}{\|A\|_1} = \frac{\|h_{2m+1}(2^{-s}A)\|_1}{\|2^{-s}A\|_1} &\le \frac{\|\widetilde{h}_{2m+1}(2^{-s}|A|)\|_1}{\|2^{-s}A\|_1} \\ &\le |c_{2m+1}| \frac{\|\,|2^{-s}A|^{2m+1}\,\|_1}{\|2^{-s}A\|_1} + \sum_{k=2m+2}^{\infty} |c_k| \frac{\|\,|2^{-s}A|^{k}\,\|_1}{\|2^{-s}A\|_1} \\ &\le \widetilde{\phi}_{2m}(\|2^{-s}A\|_1).\end{aligned} \tag{2.29}$$

We select the smallest integer $\ell_m \ge 0$ so that $|c_{2m+1}|\,\|\,|2^{-s-\ell_m}A|^{2m+1}\|_1 / \|2^{-s-\ell_m}A\|_1 \le u$, that is,

$$\ell_m = \max\left(\left\lceil \log_2\left(|c_{2m+1}| \frac{\|\,|2^{-s}A|^{2m+1}\|_1}{u\|2^{-s}A\|_1}\right) /(2m)\right\rceil, 0\right). \tag{2.30}$$

If $\ell_m > 0$ then we increase s to $s + \ell_m$. The value $s + \ell_m$ cannot exceed $s_{\max}$ as long as $s \le s_{\max}$. To see why, write $s_{\max} = s + t$ and note that by the definition of $s_{\max}$ we have $\widetilde{\phi}_{2m}(\|2^{-s_{\max}}A\|_1) \le u$, which yields from (2.29) that

$$|c_{2m+1}| \frac{\|\,|2^{-s-t}A|^{2m+1}\,\|_1}{\|2^{-s-t}A\|_1} \le u.$$

As ℓ_m is chosen to be the smallest nonnegative integer such that this relation holds, we have $t \ge \ell_m$, that is, $s + \ell_m \le s_{\max}$. Note that we can evaluate $\|\,|2^{-s}A|^{2m+1}\,\|_1$ in $O(n^2)$ operations by repeated matrix–vector products, as $\|\,|2^{-s}A|^{2m+1}e\,\|_\infty$. Also, while $\|\,|2^{-s}A|^{2m+1}\,\|_1$ can be large, we have $|c_{2m+1}| \ll 1$, so ℓ_m should not typically be large. Experiments show that this heuristic choice of ℓ_m has the ability usually to increase s just when needed. If (2.28) is satisfied we proceed with this s; otherwise we revert to Algorithm 2.3.1, reusing as many of the computed quantities as possible.

To obtain $r_m(A)$, with A once again denoting the scaled matrix $2^{-s}A$, we solve the multiple right-hand side linear system (2.20) with the coefficient matrix $q_m(A) =$

$-U+V$, where $U = u_m(A)$, $V = v_m(A)$. Since $\rho(A) \le \alpha_p(A) \le \theta_m < \nu_m$ by (2.6) and (2.24), the matrix $q_m(A)$ is nonsingular and the series $q_m(A)^{-1} = \sum_{k=0}^{\infty} a_k A^k$ converges absolutely. But in addition we want $q_m(A)$ to be well conditioned, so that the system (2.20) can be solved accurately in floating point arithmetic. For any $\epsilon > 0$, there exists a consistent matrix norm $\|\cdot\|_A$ such that $\|A\|_A \le \rho(A) + \epsilon \le \alpha_p(A) + \epsilon$. The corresponding condition number is

$$\begin{aligned}\kappa_A(q_m(A)) &= \|q_m(A)\|_A \|q_m(A)^{-1}\|_A \\ &\le p_m(\alpha_p(A)+\epsilon) \sum_{k=0}^{\infty} |a_k|\,(\alpha_p(A)+\epsilon)^k \le p_m(\theta_m+\epsilon) \sum_{k=0}^{\infty} |a_k|\,(\theta_m+\epsilon)^k,\end{aligned}$$

where we have used the properties of p_m and q_m mentioned at the start of this section. We choose $\epsilon = u$ and list these upper bounds for $\kappa_A(q_m(A))$ in Table 2.2. Since the norm $\|\cdot\|_A$ can be very badly scaled the practical value of these bounds for a particular A is difficult to judge. However, we can compute an a posteriori forward error bound for (2.20). This bound is, with X denoting r_m and the residual matrix $\widehat{R} = fl(U+V-(-U+V)\widehat{X})$ for the computed U and V,

$$\frac{\|X-\widehat{X}\|_M}{\|\widehat{X}\|_M} \le \frac{\|\,|(-U+V)^{-1}|(|\widehat{R}| + \gamma_{n+1}(|-U+V||\widehat{X}| + |U+V|)\|_M}{\|\widehat{X}\|_M}, \tag{2.31}$$

where $\|X\|_M = \max_{i,j} |x_{ij}|$ and $\gamma_k = ku/(1-ku)$. This bound is from [29, eq. (7.31)] and is essentially the best possible forward error bound. Given that we already have an LU factorization of $-U+V$ from solving the linear system, this bound can be cheaply estimated without computing $(-U+V)^{-1}$, as described in [29, Sec. 15.1]. The cost of forming the bound (2.31) is therefore essentially one matrix multiplication—that needed for R.

We are now in a position to design the new algorithm. Higham's cost analysis and evaluation schemes stated above remain applicable. From (2.23b) it is easily seen that $\alpha_3(A) \le \alpha_2(A) \le \alpha_1(A)$. Thus, for $m = 3, 5, 7, 9$ the optimal values of p subject to the constraint $2m \ge 2p(p-1)$ are $p = 2, 2, 3, 3$, respectively, and for $m = 13$ the optimal value of p is 3 or 4 (either of $\alpha_3(A)$ and $\alpha_4(A)$ can be the smaller). Thus, using the 1-norm, we need to compute the quantities

$$\alpha_p = \max\bigl(d_{2p}, d_{2p+2}\bigr), \quad p = 2\colon 4, \qquad d_{2j} := \|A^{2j}\|_1^{1/(2j)}, \quad j = 2\colon 5.$$

However, computing the d_{2j} requires computing powers of A that are not needed to evaluate r_m, for which the highest explicitly computed power is, depending on A, the eighth or lower. We will use the powers of A that are evaluated in the schemes (2.18) and (2.19), and for other powers compute norm *estimates* using the block 1-norm estimation algorithm of Higham and Tisseur [34], which for an $n \times n$ matrix carries out a 1-norm power iteration whose iterates are $n \times t$ matrices, where t is a parameter that we take to be 2. This algorithm typically requires the evaluation of about $4t$ matrix–vector products and almost invariably produces a norm estimate (which is, in fact, a lower bound on the norm) correct to within a factor 3.

Now we describe the details of how to choose m and s, with s minimal, so that the bound in (2.23a) is no larger than u.

1. Compute $A_2 = A^2$ and set $s = 0$ and $m = 3$, so $p = 2$ is the optimal value such that $2m \geq 2p(p-1)$, as explained above. We need $\eta_1 = \max(d_4, d_6)$. Since A^4 and A^6 are not needed by r_3, use estimates of d_4 and d_6 obtained by applying the norm estimator to A_2^2 and A_2^3 (a product $A_2^3 x$ required by the estimator is computed by three matrix–vector multiplications with A_2, for example). If $\eta_1 \leq \theta_3$ quit, otherwise continue to step 2.

2. Compute $A_4 = A_2^2$ and set $m = 5$, for which $p = 2$ is again the optimal value such that $2m \geq 2p(p-1)$. Now we have d_4 and can reuse the estimate of d_6, setting $\eta_2 = \max(d_4, d_6)$. If $\eta_2 \leq \theta_5$ quit, otherwise continue to step 3.

3. Compute $A_6 = A_4 A_2$. For $m \in \{7, 9\}$, $p = 3$ is the optimal value such that $2m \geq 2p(p-1)$. We compute $\eta_3 = \max(d_6, d_8)$, in which we estimate d_8 by applying the norm estimator to A_4^2. If $\eta_3 \leq \theta_7$ set $m = 7$, else if $\eta_3 \leq \theta_9$ set $m = 9$, else continue to step 4.

4. Set $m = 13$, for which either $p = 3$ or $p = 4$ is the optimal value such that $2m \geq 2p(p-1)$. The highest power of A that we compute to evaluate r_{13} by (2.19) is A^6, so we use the norm estimator to estimate d_{10} and set $\eta_4 = \max(d_8, d_{10})$. Choose the smallest $s \geq 0$ such that $2^{-s}\eta_5 \leq \theta_{13}$, where $\eta_5 = \min(\eta_3, \eta_4)$.

We introduce two more algorithmic refinements. First, we use $\theta_{13} = 4.25$ in place of the value $\theta_{13} = 5.37$ used in Algorithm 2.3.1 (see Table 2.2). The reason is that this produces a slightly better conditioned denominator polynomial q_{13} and our experiments show that in the context of our more liberal choice of s this is beneficial to the accuracy of the computed exponential. This refinement can lead to s exceeding $s_{\max}$, but only by 1, and in this case $\ell_m = 0$ as can be seen from (2.29). The second refinement is that for each putative m we compute the correction (2.30) *before* evaluating p_m and q_m and checking the inequality (2.28) and, if the correction is nonzero, we proceed to the next larger choice of m. This is simply another means for trying to avoid an inaccurate evaluation (or, put another way, wasted computation).

Now we write the new scaling and squaring algorithm for the matrix exponential.

Algorithm 2.5.1 *This algorithm evaluates the matrix exponential $X = e^A$ of $A \in \mathbb{C}^{n\times n}$ by the scaling and squaring method. It is intended for IEEE double precision arithmetic. It uses the parameters θ_m given in Table* 2.2 *and the following functions*

- `normest`, *which when invoked as* `normest`$(A_1, A_2, \ldots, A_k)$ *produces an estimate of* $\|A_1 A_2 \ldots A_k\|_1$ *and when invoked as* `normest`(A, m) *produces an estimate of* $\|A^m\|_1$*;*

- `ell`(A, m)*, which returns the integer* $\max(\lceil(\log_2(\alpha/u)/(2m)\rceil), 0)$*,*
  where $\alpha = |c_{2m+1}|$`normest`$(|A|, 2m+1)/\|A\|_1$.

```
$A_2 = A^2$
$d_6 = \texttt{normest}(A_2, 3)^{1/6}$, $\eta_1 = \max\big(\texttt{normest}(A_2, 2)^{1/4}, d_6\big)$
if $\eta_1 \leq \theta_3$ and $\texttt{ell}(A, 3) = 0$
    Evaluate $p_3(A)$ and $q_3(A)$ using (2.18).
    if $\|p_3(|A|)^T e\|_\infty / \min(\|p_3\|_1, \|q_3\|_1) \leq 10e^{\theta_3}$
```

```
6        Evaluate $r_3$ using (2.20), quit.
7      end
8    end
9    $A_4 = A_2^2$, $d_4 = \|A_4\|_1^{1/4}$
10   $\eta_2 = \max\big(d_4, d_6\big)$
11   if $\eta_2 \le \theta_5$ and $\mathtt{ell}(A, 5) = 0$
12     Evaluate $p_5(A)$ and $q_5(A)$ using (2.18).
13     if $\|p_5(|A|)^T e\|_\infty / \min(\|p_5\|_1, \|q_5\|_1) \le 10e^{\theta_5}$
14       Evaluate $r_5$ using (2.20), quit.
15     end
16   end
17   $A_6 = A_2 A_4$, $d_6 = \|A_6\|_1^{1/6}$
18   $d_8 = \mathtt{normest}(A_4, 2)^{1/8}$, $\eta_3 = \max(d_6, d_8)$
19   for $m = [7, 9]$
20       if $\eta_3 \le \theta_m$ and $\mathtt{ell}(A, m) = 0$
21         Evaluate $p_m(A)$ and $q_m(A)$ using (2.18).
22         if $\|p_m(|A|)^T e\|_\infty / \min(\|p_m\|_1, \|q_m\|_1) \le 10e^{\theta_m}$
23           Evaluate $r_m$ using (2.20), quit.
24         end
25       end
26   end
27   $\eta_4 = \max\big(d_8, \mathtt{normest}(A_4, A_6)^{1/10}\big)$
28   $\eta_5 = \min(\eta_3, \eta_4)$
29   $s = \max\big(\lceil \log_2(\eta_5/\theta_{13}) \rceil, 0\big)$
30   $s = s + \mathtt{ell}(2^{-s}A, 13)$
31   $A \leftarrow 2^{-s}A$, $A_2 \leftarrow 2^{-2s}A_2$, $A_4 \leftarrow 2^{-4s}A_4$, $A_6 \leftarrow 2^{-6s}A_6$
32   Evaluate $p_{13}(A)$ and $q_{13}(A)$ using (2.19).
33   if $\|p_{13}(|A|)^T e\|_\infty / \min(\|p_{13}\|_1, \|q_{13}\|_1) \le (10 + s_{\max})e^{\theta_{13}}$
34     Evaluate $r_{13}$ using (2.20), quit.
35   else
36     $s_1 = s_{\max} - s$, $s = s_{\max}$
37     $A \leftarrow 2^{-s_1}A$, $A_2 \leftarrow 2^{-2s_1}A_2$, $A_4 \leftarrow 2^{-4s_1}A_4$, $A_6 \leftarrow 2^{-6s_1}A_6$
38     Evaluate $r_{13}$ using (2.19) and (2.20).
39   end
40   if $A$ is triangular
41     Invoke Code Fragment 2.2.1.
42   else
43     $X = r_{13}(A)^{2^s}$ by repeated squaring.
44   end
```

Cost: $(\pi_m + s)M + D$, where m is the degree of Padé approximant used and π_m (tabulated in [30, Table 2.2]) is the cost of evaluating p_m and q_m. If line 36 is executed then the cost is $(\pi_{13} + s + 3)M + D$. If any of the tests at lines 5, 13, and 22 fail then there is some wasted work in evaluating lower degree p_m and q_m that are not used.

Note that we have not included the bound (2.31) because it would require an extra matrix multiplication and the algorithm performs well in practice without the use of it. It is easy to check that if line 33 is reached then Algorithm 2.3.1 would choose $m = 13$, so at line 36 the algorithm is reverting to Algorithm 2.3.1.

2.6 Numerical experiments

We now compare Algorithm 2.3.1, as implemented in `expm`, and Algorithm 2.5.1 experimentally. We will use four sets of test matrices.

Set 1 Matrices from the literature on developing methods for e^A (including (2.3) with $b = 10^7$), mostly intended to be difficult for the scaling and squaring method. All are of dimension 10 or less.

Set 2 10×10 matrices from MATLAB (in particular, from the `gallery` function), and from the Matrix Computation Toolbox [25].

Set 3 The upper triangular Schur factors of the matrices from Set 2.

Set 4 Test matrices provided with EigTool [73], which are mainly discretizations of operators. The matrices are of variable dimension, which we have taken to be as close as possible to $n = 50$.

The tests in [30] and [31] used Sets 1 and 2 together.

Our tests were done in MATLAB 7.10 (R2010a). We compute normwise relative errors $\|\widehat{X} - e^A\|_F / \|e^A\|_F$ of the computed $\widehat{X}$ by approximating e^A by the result computed at 100 digit precision using Symbolic Math Toolbox. For Sets 1 and 3, Algorithm 2.5.1 produces many errors of zero, but to facilitate the plots we replace a zero error for this algorithm by 10^{-18}.

For each set we present the results as four plots in a 2×2 grid; see Figures 2.4–2.7. The (1,1) plot shows the relative errors for the two algorithms, where the matrices are sorted by decreasing value of the condition number $\kappa_{\text{exp}}(A)$ in (2.5), and $\kappa_{\text{exp}}(A)u$ is shown as a solid line. The (1,2) plot shows the $\log_{10}$ of the ratio of relative errors, sorted in increasing order, and this same ordering is also used by the (2,1) and (2,2) plots. The (2,1) plot shows the values of s chosen by each method. The (2,2) plot shows the ratio of the costs of the algorithms, where cost is measured as described after the statement of each algorithm and we regard M and D as equal. Note that this measure of cost is appropriate only for $n \gg 10$, but since the choice of s and m depends only on $\|A\|_1$ for Algorithm 2.3.1 and on $\|A^k\|_1^{1/k}$ for certain k for Algorithm 2.5.1, these results are indicative of the relative costs for much larger matrices. We note that the cost ratio cannot exceed $8/7 \approx 1.14$, and can be greater than 1 only because of the differing values for θ_{13} for the two algorithms (see Table 2.2).

The main points to note are as follows.

(1) Algorithm 2.5.1 did not revert to Algorithm 2.3.1 (on line 36) for any of the test matrices. Moreover, the tests at lines 5, 13, and 22 never failed to be satisfied. The correction (2.30) was nonzero at line 30 on 6, 10, 0, and 2 occasions on the four test sets, respectively. If we remove line 30 then there are 6 reversions in Test 1, 1 in Test 2, and none in Tests 3 and 4. The value of `ell` at lines 3, 11, and 20 was nonzero once for Test 1, 5 times for Test 2, and not at all for Tests 3 and 4.

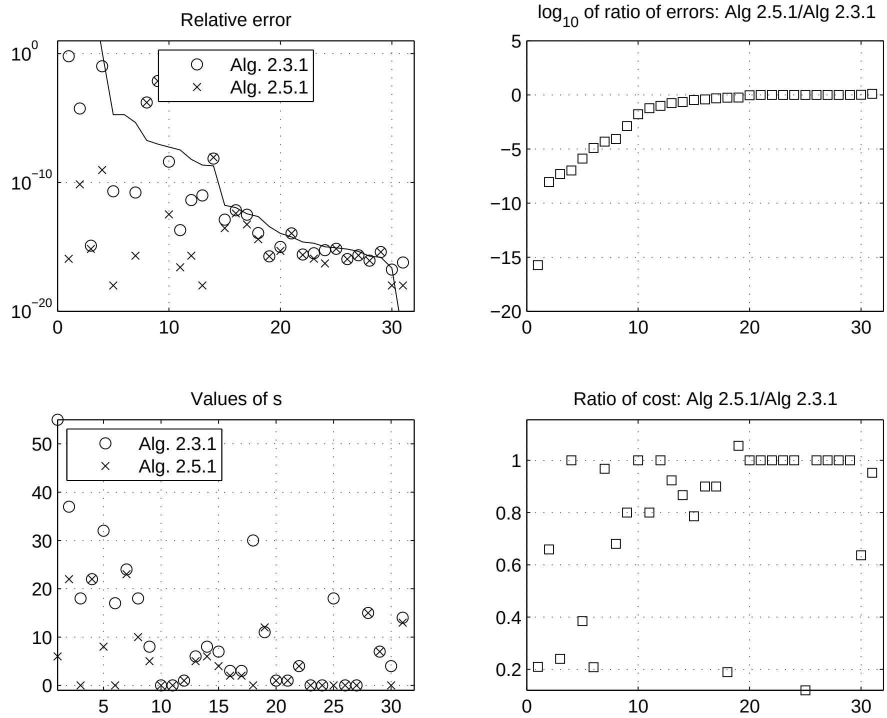

Figure 2.4: Results for test matrix Set 1.

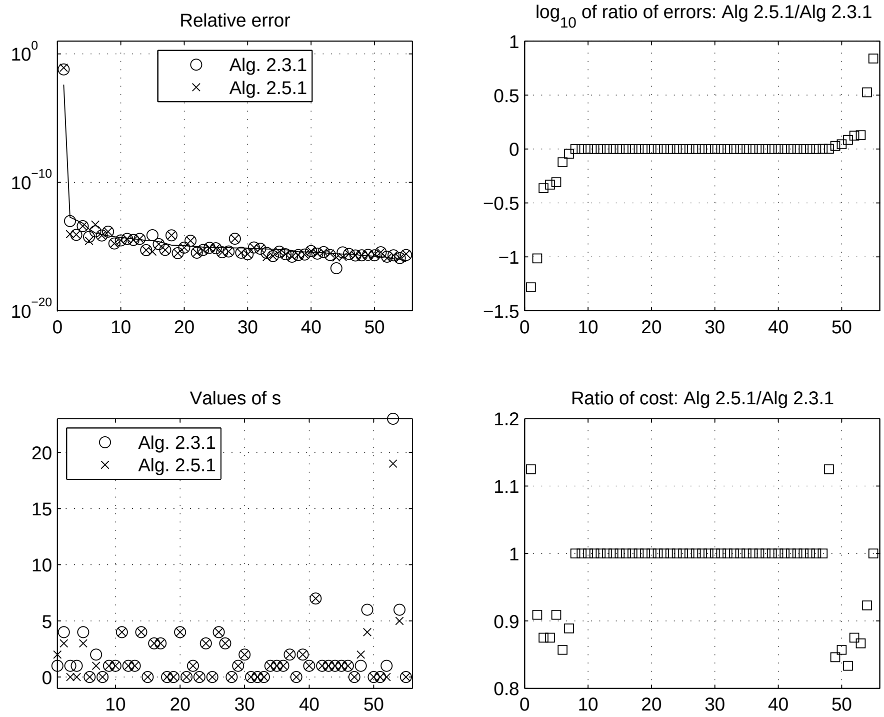

Figure 2.5: Results for test matrix Set 2.

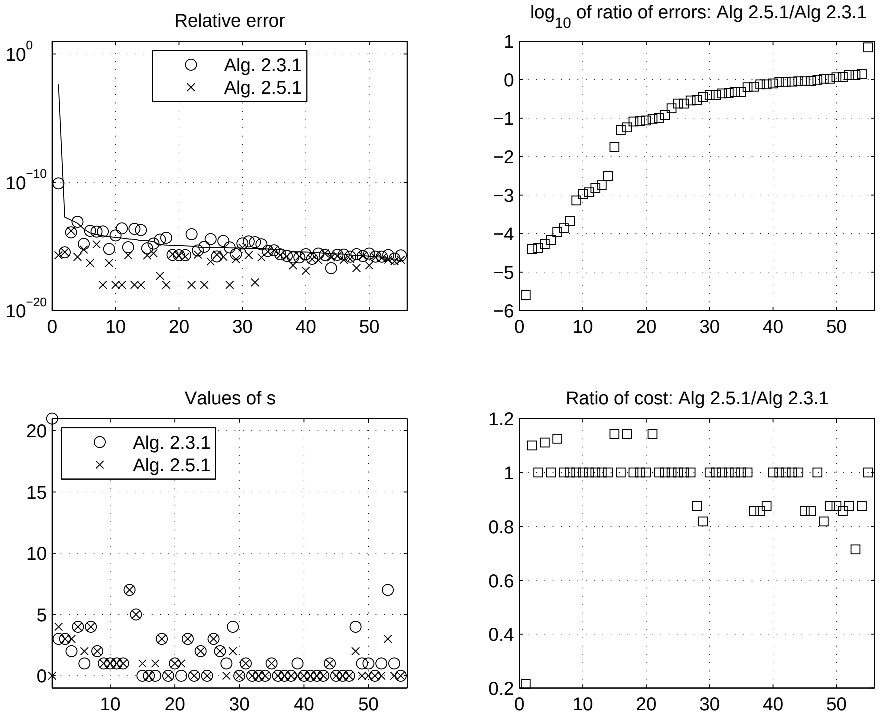

Figure 2.6: Results for test matrix Set 3.

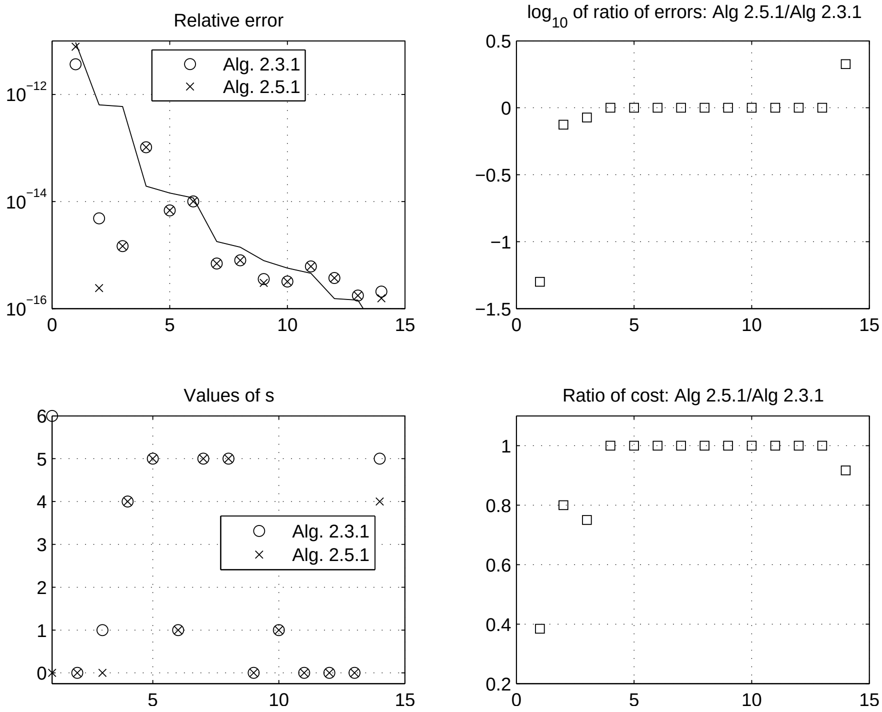

Figure 2.7: Results for test matrix Set 4.

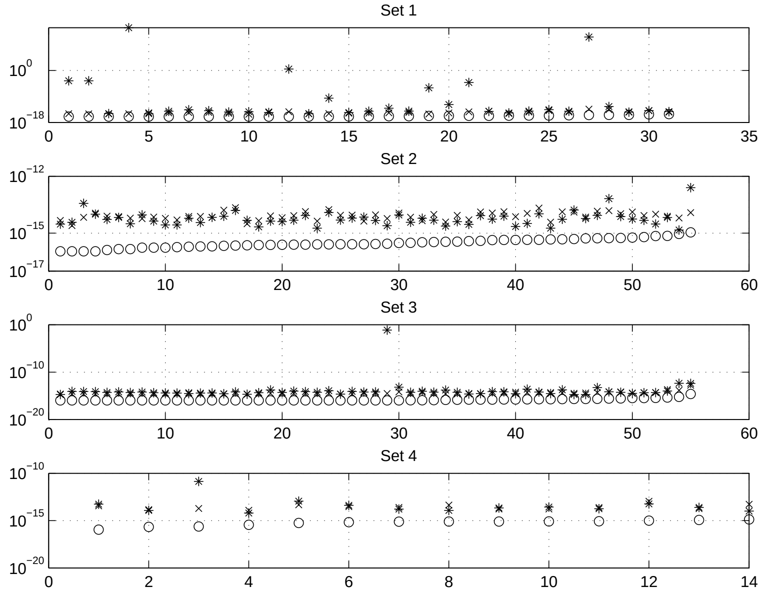

Figure 2.8: Quantities associated with the computed $\widehat{r}_m \approx r_m(2^{-s}A)$ for Algorithm 2.5.1: relative error in $\widehat{r}_m$ ("∘"), a posteriori forward error bound (2.31) ("×"), and $n\kappa_1(q_m)u$ ("∗")—an approximate a priori bound for the error.

(2) For Set 1, Algorithm 2.5.1 has a cost up to about 5 times smaller than Algorithm 2.3.1 while achieving error barely any larger and sometimes orders of magnitude smaller. This is due to Algorithm 2.5.1 frequently choosing a smaller s.

(3) For Set 2 there is no significant difference in the accuracy of the two algorithms. But in 22% of the cases Algorithm 2.5.1 is less expensive than Algorithm 2.3.1, by up to 17% while in just two cases it is more expensive, by 12%.

(4) For Set 3, Algorithm 2.5.1 is more accurate than Algorithm 2.3.1 in almost every case, often by orders of magnitude. This is mainly due to exploiting triangularity in the squaring phase.

(5) For Set 4, Algorithm 2.5.1 is superior to Algorithm 2.3.1 in speed and accuracy for three of the matrices and performs equivalently to it for the rest except for one case.

(6) Figure 2.8 provides information about the linear systems (2.20) that are solved to obtain $r_m(2^{-s}A)$. It shows the relative error $\|r_m - \widehat{r}_m\|_1/\|r_m\|_1$ along with the a posteriori bound (2.31) and the approximate a priori bound $n\kappa_1(q_m)u$ for this error. The results show that (a) the relative error is reasonably small in every case, (b) the system is sometimes solved to much better accuracy than the condition number $\kappa_1(q_m)$ would suggest (see Set 1), and (c) the a posteriori bound is a surprisingly good predictor of the actual error.

These experiments and our additional investigations lead us to conclude that no benefit is gained from using the test (2.28) to gauge whether the evaluation of p_m and q_m has been sufficiently accurate. Therefore we recommend the following simplification of Algorithm 2.5.1, which we emphasize performs identically to Algorithm 2.5.1 on the tests reported here.

Algorithm 2.6.1 *This algorithm is identical to Algorithm* 2.5.1 *except that lines* 5, 7, 13, 15, 22, 24, 33, *and* 35–39 *are removed.*

```
$A_2 = A^2$
$d_6 = \texttt{normest}(A_2, 3)^{1/6}$, $\eta_1 = \max\bigl(\texttt{normest}(A_2, 2)^{1/4}, d_6\bigr)$
if $\eta_1 \le \theta_3$ and $\texttt{ell}(A, 3) = 0$
  Evaluate $r_3(A)$ using (2.18) and (2.20)
  quit
end
$A_4 = A_2^2$, $d_4 = \|A_4\|_1^{1/4}$
$\eta_2 = \max\bigl(d_4, d_6\bigr)$
if $\eta_2 \le \theta_5$ and $\texttt{ell}(A, 5) = 0$
  Evaluate $r_5(A)$ using (2.18) and (2.20)
  quit
end
$A_6 = A_2 A_4$, $d_6 = \|A_6\|_1^{1/6}$
$d_8 = \texttt{normest}(A_4, 2)^{1/8}$, $\eta_3 = \max(d_6, d_8)$
for $m = [7, 9]$
    if $\eta_3 \le \theta_m$ and $\texttt{ell}(A, m) = 0$
      Evaluate $r_m(A)$ using (2.18) and (2.20)
    quit
    end
end
$\eta_4 = \max\bigl(d_8, \texttt{normest}(A_4, A_6)^{1/10}\bigr)$
$\eta_5 = \min(\eta_3, \eta_4)$
$s = \max\bigl(\lceil \log_2(\eta_5/\theta_{13}) \rceil, 0\bigr)$
$s = s + \texttt{ell}(2^{-s}A, 13)$
$A \leftarrow 2^{-s}A$, $A_2 \leftarrow 2^{-2s}A_2$, $A_4 \leftarrow 2^{-4s}A_4$, $A_6 \leftarrow 2^{-6s}A_6$
Evaluate $r_{13}(A)$ using (2.19) and (2.20)
if $A$ is triangular
  Invoke Code Fragment 2.2.1.
else
  $X = r_{13}(A)^{2^s}$ by repeated squaring.
end
```

2.7 Cost of Padé versus Taylor approximants within the scaling and squaring method

In this section we compare the efficiency of diagonal Padé approximants and Taylor approximants within the scaling and squaring method for the matrix exponential, based on the use of refined backward error bounds in both cases.

For $A \in \mathbb{C}^{n \times n}$ we use the Paterson-Stockmeyer scheme [60], [31, Sec. 4.2] to evaluate $T_m(A) = \sum_{k=0}^{m} A^k/k!$ as

$$T_m(A) = \sum_{k=0}^{\ell} g_k(A)(A^\tau)^k, \quad \ell = \lfloor m/\tau \rfloor, \tag{2.32}$$

where $1 \le \tau \le m$ is an integer and

$$g_k(A) = \begin{cases} \sum_{i=1}^{\tau} A^{\tau-i}/(\tau k + \tau - i)!, & k = 0 : \ell - 1, \\ \sum_{i=\ell\tau}^{m} A^{i-\ell\tau}/i!, & k = \ell. \end{cases}$$

Horner's rule is used on (2.32). This scheme evaluates $T_m(A)$ at a number of matrix multiplications equal to

$$\widetilde{\pi}_m = \ell + \tau - 1 - \phi(m, \tau), \qquad \phi(m, \tau) = \begin{cases} 1, & \text{if } \tau \mid m, \\ 0, & \text{otherwise.} \end{cases} \tag{2.33}$$

The choice $\tau = \sqrt{m}$ approximately minimizes this quantity [31, Sec. 4.2], so we take τ either $\lfloor \sqrt{m} \rfloor$ or $\lceil \sqrt{m}\ \rceil$ since both yield the same operation count [23, Thm. 1.7.4].

The analysis in Section 2.3 is applicable to any rational approximation to e^x, not just diagonal Padé approximants, so we can replace r_m therein by T_m. Thus with $h_{m+1}(x) = \log(e^{-x}T_m(x)) = \sum_{k=m+1}^{\infty} c_k x^k$ we calculate the parameters

$$\widetilde{\theta}_m = \max\{\, \theta : \widetilde{h}_{m+1}(\theta)/\theta \le u = 2^{-53} \,\}, \tag{2.34}$$

where $\widetilde{h}_{m+1}(x) = \sum_{k=m+1}^{\infty} |c_k| x^k$, using the same techniques described as in Section 2.3. Then we know that $T_m(2^{-s}A)^{2^s}$ has backward error at most u for $\|2^{-s}A\| \le \widetilde{\theta}_m$. We select s as the smallest nonnegative integer such that $2^{-s}\|A\| \le \widetilde{\theta}_m$, which is given by $s = \max(\lceil \log_2(\|A\|/\widetilde{\theta}_m) \rceil, 0)$. Then the number of matrix multiplications required to evaluate $T_m(2^{-s}A)^{2^s}$ is

$$\underbrace{\lfloor m/\lceil \sqrt{m}\ \rceil \rfloor + \lceil \sqrt{m}\ \rceil - 1 - \phi(m, \lceil \sqrt{m}\ \rceil)}_{\widetilde{\pi}_m} + \max(\lceil \log_2(\|A\|/\widetilde{\theta}_m) \rceil, 0). \tag{2.35}$$

When $s > 0$ we are interested in m that minimizes the cost. To obtain a suitable measure of the cost we ignore the constant terms in (2.35) (since they are common to each m) and consider

$$C_m = \lfloor m/\lceil \sqrt{m}\ \rceil \rfloor + \lceil \sqrt{m}\ \rceil - \phi(m, \lceil \sqrt{m}\ \rceil) - \log_2(\widetilde{\theta}_m). \tag{2.36}$$

We tabulate $\widetilde{\pi}_m$ and $\widetilde{\theta}_m$ for $m = 1\colon 30$ in Table 2.3 and find that $m = 16$ is the global minimizer of C_m, which suggests using $T_{16}(2^{-s}A)^{2^s}$ to approximate e^A when $\|A\| \ge \widetilde{\theta}_{16} \approx 0.78$. Corresponding analysis was done for Padé approximants by [30] and we use the number of matrix multiplications π_m from [30, Table 2.2] below.

Now we analyze the cost of Taylor and Padé approximants with scaling and squaring when applying the two methods simultaneously on the matrix A. Assume first that $\|A\| \ge \theta_{13} \approx 5.3$. Computing $T_{16}(2^{-s}A)$ requires six matrix multiplications and so the overall cost from (2.35) of approximating e^A is $c_T := 6 + s$ while Algorithm 2.3.1, which chooses a nonnegative integer t so that $\frac{1}{2}\theta_{13} < \|2^{-t}A\| \le \theta_{13}$, computes $r_{13}(2^{-t}A)^{2^t}$ with cost $c_P := 6 + 4/3 + t$, where the term $4/3$ accounts for the solution of the multiple right-hand side linear system for the Padé approximant. Since $\frac{1}{2}\theta_{13} < 4\widetilde{\theta}_{16}$ there are two cases to consider. First, when $\|2^{-t}A\| \in (4\widetilde{\theta}_{16}, \theta_{13}]$ we have $2^{-t-3}\|A\| \le \frac{1}{8}\theta_{13} < \widetilde{\theta}_{16}$ and hence $s = t+3$. Therefore, $1 < c_T/c_P = (9+t)/(7\frac{1}{3}+t) \le 27/22$. Secondly, when $\|2^{-t}A\| \in (\frac{1}{2}\theta_{13}, 4\widetilde{\theta}_{16}]$ we have $2^{-t-2}\|A\| \le \widetilde{\theta}_{16}$ and hence

Table 2.3: The number of matrix products $\widetilde{\pi}_m$ in (2.33) needed for the Paterson-Stockmeyer scheme, $\widetilde{\theta}_m$ defined by (2.34), and C_m from (2.36).

m	$\widetilde{\theta}_m$	$\widetilde{\pi}_m$	C_m	m	$\widetilde{\theta}_m$	$\widetilde{\pi}_m$	C_m	m	$\widetilde{\theta}_m$	$\widetilde{\pi}_m$	C_m
1	1.5e-8	0	26.99	11	2.14e-1	5	8.22	21	1.62	8	8.30
2	2.6e-8	1	27.21	12	3.00e-1	5	7.74	22	1.82	8	8.14
3	1.4e-5	2	19.14	13	4.00e-1	6	8.32	23	2.01	8	7.99
4	3.4e-4	2	14.52	14	5.14e-1	6	7.96	24	2.22	8	7.85
5	2.4e-3	3	12.70	15	6.41e-1	6	7.64	25	2.43	8	7.72
6	9.1e-3	3	10.79	16	7.81e-1	6	7.36	26	2.64	9	8.60
7	2.4e-2	4	10.39	17	9.31e-1	7	8.10	27	2.86	9	8.48
8	5.0e-2	4	9.32	18	1.09	7	7.87	28	3.08	9	8.38
9	9.0e-2	4	8.48	19	1.26	7	7.67	29	3.31	9	8.27
10	1.44e-1	5	8.79	20	1.44	7	7.48	30	3.54	9	8.18

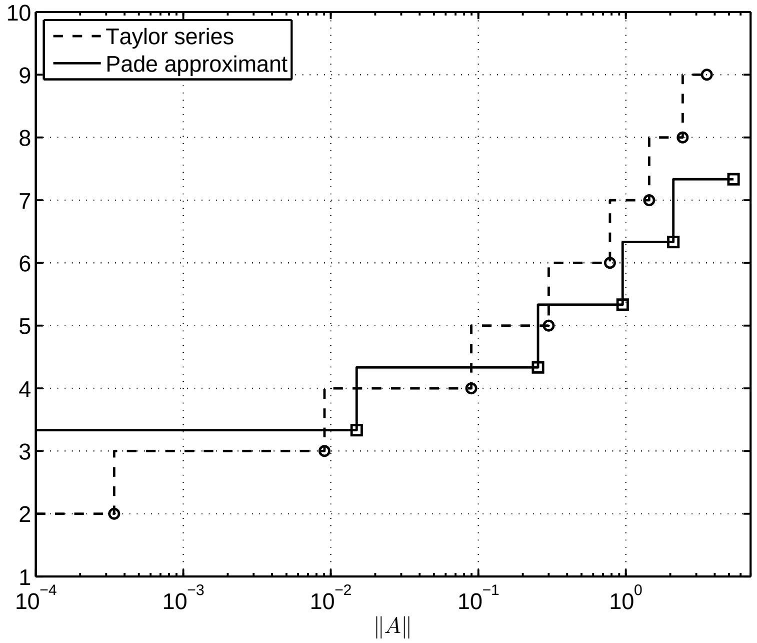

Figure 2.9: $\|A\|$ versus cost in equivalent matrix multiplications of evaluating Taylor and Padé approximants to e^A in double precision.

$s = t + 2$. Therefore, $1 < c_T/c_P = (8+t)/(7\frac{1}{3}+t) \le 12/11$. Thus any algorithm based on Taylor series will cost up to 23% more than the Padé approximant-based Algorithm 2.3.1 and cannot have a lower cost for $\|A\| > 4\widetilde{\theta}_{16}$. Moreover the Taylor series requires a larger amount of scaling (since we are scaling to reduce $\|A\|$ below 0.78 instead of 5.4), which is undesirable from the point of view of possible numerical instability in the squaring phase.

A remaining question is whether when $\|A\| < \theta_{13}$ a Taylor series can be more efficient than a Padé approximant. The answer can be seen from Figure 2.9, where "∘" indicates the points $(\widetilde{\theta}_m, \widetilde{\pi}_m)$, $m = 4, 6, 9, 12, 16, 20, 25, 30$, and "□" indicates the points $(\theta_m, \pi_m{+}4/3)$, $m = 3, 5, 7, 9, 13$. Notice that the dotted curve, which represents the cost of Taylor series, lies below the solid curve in three intervals: $[0, \widetilde{\theta}_6]$, $(\theta_3, \widetilde{\theta}_9]$, and $(\theta_5, \widetilde{\theta}_{12}]$. Therefore, it is more efficient to use $T_m(A)$ rather Algorithm 2.3.1 if $\|A\|$ lies in any of these intervals. Precisely, in view of Figure 2.9, $T_m(A)$, $m = 4, 6, 9, 12$, is more efficient if $\|A\|$ lies, respectively, in the intervals $[0, \widetilde{\theta}_4]$, $(\widetilde{\theta}_4, \widetilde{\theta}_6]$, $(\theta_3, \widetilde{\theta}_9]$, or $(\theta_5, \widetilde{\theta}_{12}]$.

The above analysis is for computations in double precision. For single precision we reach similar conclusion.

2.7.1 Single precision

We repeat the analysis above (with the same notations) for $u = 2^{-24} \approx 6 \times 10^{-8}$, the unit roundoff for IEEE single precision arithmetic, and recalculate the parameters $\widetilde{\theta}_m$ in (2.34). We find that $m = 9$ minimizes C_m in (2.36). Thus $T_9(2^{-s}A)^{2^s}$, where $s \ge 0$ is the optimal integer such that $2^{-s}\|A\| \le \widetilde{\theta}_9 \approx 0.78$, is the candidate for approximating e^A in single precision arithmetic. [30] shows that [7/7] Padé approximant $r_7(2^{-t}A)^{2^t}$ is the one to be chosen for the approximation, where $t \ge 0$ is the optimal integer such that $2^{-t}\|A\| \le \theta_7 \approx 3.9$. Suppose now that $\|A\| \ge \theta_7$. Thus computing $T_9(2^{-s}A)^{2^s}$ costs $c_T := 4 + s$ by Table 2.3 while the cost of Padé approximant is $c_P := 4{+}4/3{+}t$. Now we write s in terms of t and compare the costs for both methods. Notice that $\frac{1}{2}\theta_7 < 4\widetilde{\theta}_9 < \theta_7$. Since the optimal choice of t implies that $2^{-t}\|A\| > \frac{1}{2}\theta_7$, there are two cases to consider. First, if $2^{-t}\|A\| \in (4\widetilde{\theta}_9, \theta_7]$ then $2^{-t-3}\|A\| \le \frac{1}{8}\theta_7 < \widetilde{\theta}_9$ and hence $s = t + 3$. In this case $1 < c_T/c_P = (7+t)/(5\frac{1}{3}+t) \le 21/16$. Secondly, if $2^{-t}\|A\| \in (\frac{1}{2}\theta_7, 4\widetilde{\theta}_9]$ then $2^{-t-2}\|A\| \le \widetilde{\theta}_9$ and hence $s = t + 2$. Similarly, we obtain $1 < c_T/c_P \le 9/8$. Therefore, Padé approximant is more efficient that Taylor series for all matrices A with $\|A\| \ge \theta_7$. When $\|A\| < \theta_7$ Figure 2.10 shows the comparison. Respectively, "∘" indicates the points $(\widetilde{\theta}_m, \widetilde{\pi}_m)$, $m = 4, 6, 9, 12, 16, 20, 25$, and "□" indicates the points $(\theta_m, \pi_m + 4/3)$, $m = 3, 5, 7$. It shows that $T_m(A)$, $m = 4, 6, 9$ are more efficient that $r_m(A)$, $m = 3, 3, 5$, respectively, over the intervals $[0, \widetilde{\theta}_4]$, $(\widetilde{\theta}_4, \widetilde{\theta}_6]$, and $(\theta_3, \widetilde{\theta}_9]$.

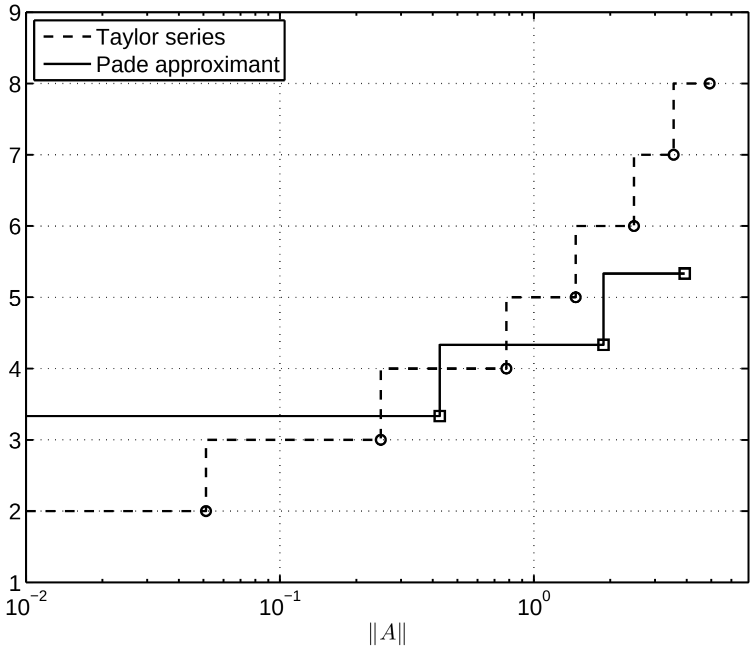

Figure 2.10: $\|A\|$ versus cost in equivalent matrix multiplications of evaluating Taylor and Padé approximants to e^A in single precision.

Chapter 3

Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators

3.1 Introduction

The most popular method for computing the matrix exponential is the scaling and squaring method that we investigate profoundly in Chapter 2. For a matrix $A \in \mathbb{C}^{n\times n}$ it exploits the relation $e^A = (e^{2^{-i}A})^{2^i} \approx (r_m(2^{-i}A))^{2^i}$, where r_m is an $[m/m]$ Padé approximant of the exponential. The parameters m and i can be determined so that truncation errors correspond to a backward error no larger than a specified tolerance (for example, the unit roundoff). In some applications, notably in the numerical solution of ordinary differential equations (ODEs) and in the approximation of dynamical systems [6, Chap. 4], it is not e^A that is required but the action of e^A on a matrix, e^AB, where $B \in \mathbb{C}^{n\times n_0}$ with $n_0 \ll n$, and often $n_0 = 1$, so that B is a vector. The exponential of a sparse matrix is generally full, so when A is large and sparse it is not practical to form e^A and then multiply it into B. Our first contribution in this work is to derive a new algorithm for computing e^AB without explicitly forming e^A. The algorithm allows one to work directly with sparse matrices, and hence take advantage of sparsity. We adapt the scaling and squaring method by computing $e^AB \approx (T_m(s^{-1}A))^sB$, where T_m is a truncated Taylor series rather than a rational approximation (thus avoiding linear system solves) and s multiplications of $n \times n$ by $n \times n_0$ matrices are carried out instead of $\log_2 s$ squarings of $n \times n$ matrices. We employ several key ideas:

(1) Careful choice of the parameters m and s, exploiting estimates of $\|A^p\|^{1/p}$ for several p, in order to keep the backward error suitably bounded while minimizing the computational cost.

(2) Shifting, and optional balancing, to reduce the norm of A.

(3) Premature termination of the truncated Taylor series evaluations.

This basic approach is equivalent to applying a Runge–Kutta or Taylor series method with fixed stepsize to the underlying ODE $y'(t) = Ay(t)$, $y(0) = B$, for which $y(t) = e^{tA}B$, which is the sixth of Moler and Van Loan's "19 dubious ways" [54, Sec. 4], [72]. However, in (1)–(3) we are fully exploiting the linear nature of the problem in a way that a general purpose ODE integrator cannot. Moreover, our algorithmic parameters are determined by backward error considerations, whereas standard local error control for an ODE solver is forward error-based. We also adapt our method to compute approximations of $e^{t_k A}B$, for t_k equally spaced on an interval $[t_0, t_q]$, in such a way that overscaling is avoided no matter how small the stepsize.

Our second contribution concerns the numerical solution of systems of n nonlinear ODEs by exponential integrators. These methods integrate the linear part of the system exactly and approximate the nonlinear part, making use of a set of φ functions closely related to the exponential, evaluated at an $n \times n$ matrix. We show that these methods can be implemented by evaluating a single exponential of an augmented matrix of order $n+p$, where $p-1$ is the degree of the polynomial used to approximate the nonlinear part of the system, thus avoiding the need to compute any φ functions. In fact, on each integration step the integrator is shown to produce the exact solution of an augmented linear system of ODEs of dimension $n + p$. The replacement of φ functions with the exponential is important because algorithms for the φ functions are much less well developed (though see [46], [64], for example) than those for the exponential itself.

The organization of this chapter is as follows. In the next section we derive a theorem that shows how to rewrite linear combinations of φ functions of the form required in exponential integrators in terms of a single exponential of a slightly larger matrix. In Section 3.3 we derive our algorithm for computing $e^A B$ and discuss preprocessing to increase its efficiency. Analysis of the behavior of the algorithm in floating point arithmetic is given in Section 3.4, where a condition number for the $e^A B$ problem is derived. We extend the algorithm in Section 3.5 to compute $e^{tA}B$ on an equally spaced grid of t values, in such a way that the phenomenon of overscaling that has previously afflicted the scaling and squaring method is avoided. Detailed numerical experiments are given in Section 3.6, including comparison with two Krylov-based codes.

3.2 Exponential integrators: avoiding the φ functions

Exponential integrators are a class of time integration methods for solving initial value problems written in the form

$$u'(t) = Au(t) + g(t, u(t)), \quad u(t_0) = u_0, \quad t \geq t_0, \tag{3.1}$$

where $u(t) \in \mathbb{C}^n$, $A \in \mathbb{C}^{n \times n}$, and g is a nonlinear function. Spatial semidiscretization of partial differential equations (PDEs) leads to systems in this form. The matrix A usually represents the Jacobian of a certain function or an approximation of it, and it is usually large and sparse. The solution of (3.1) satisfies the nonlinear integral equation

$$u(t) = e^{(t-t_0)A}u_0 + \int_{t_0}^{t} e^{(t-\tau)A} g(\tau, u(\tau))\, d\tau. \tag{3.2}$$

By expanding g in a Taylor series about t_0, the solution can be written as [53, Lem. 5.1]

$$u(t) = e^{(t-t_0)A}u_0 + \sum_{k=1}^{\infty} \varphi_k\big((t-t_0)A\big)(t-t_0)^k\, u_k, \tag{3.3}$$

where

$$u_k = \frac{d^{k-1}}{dt^{k-1}} g(t, u(t))\mid_{t=t_0}, \quad \varphi_k(z) = \frac{1}{(k-1)!}\int_0^1 e^{(1-\theta)z}\theta^{k-1}\, d\theta, \quad k \geq 1.$$

By suitably truncating the series in (3.3), we obtain the approximation

$$u(t) \approx \widehat{u}(t) = e^{(t-t_0)A}u_0 + \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k\, u_k. \tag{3.4}$$

The functions $\varphi_\ell(z)$ satisfy the recurrence relation

$$\varphi_\ell(z) = z\varphi_{\ell+1}(z) + \frac{1}{\ell!}, \quad \varphi_0(z) = e^z,$$

and have the Taylor expansion

$$\varphi_\ell(z) = \sum_{k=0}^{\infty} \frac{z^k}{(k+\ell)!}. \tag{3.5}$$

A wide class of exponential integrator methods is obtained by employing suitable approximations to the vectors u_k in (3.4), and further methods can be obtained by the use of different approximations to g in (3.2). See Hochbruck and Ostermann [35] for a survey of the state of the art in exponential integrators.

We will show that the right-hand side of (3.4) can be represented in terms of the *single* exponential of an $(n+p)\times(n+p)$ matrix, with no need to explicitly evaluate φ functions. The following theorem is our key result. In fact we will only need the special case of the theorem with $\ell = 0$.

Theorem 3.2.1 *Let $A \in \mathbb{C}^{n\times n}$, $W = [w_1, w_2, \dots, w_p] \in \mathbb{C}^{n\times p}$, $\tau \in \mathbb{C}$, and*

$$\widetilde{A} = \begin{bmatrix} A & W \\ 0 & J \end{bmatrix} \in \mathbb{C}^{(n+p)\times(n+p)}, \quad J = \begin{bmatrix} 0 & I_{p-1} \\ 0 & 0 \end{bmatrix} \in \mathbb{C}^{p\times p}. \tag{3.6}$$

Then for $X = \varphi_\ell(\tau\widetilde{A})$ with $\ell \geq 0$ we have

$$X(1:n, n+j) = \sum_{k=1}^{j} \tau^k \varphi_{\ell+k}(\tau A)\, w_{j-k+1}, \quad j = 1\colon p. \tag{3.7}$$

Proof. It is easy to show that, for $k \geq 0$,

$$\widetilde{A}^k = \begin{bmatrix} A^k & M_k \\ 0 & J^k \end{bmatrix}, \tag{3.8}$$

where $M_k = A^{k-1}W + M_{k-1}J$ and $M_1 = W$, $M_0 = 0$. For $1 \le j \le p$ we have $WJ(:,j) = w_{j-1}$ and $JJ(:,j) = J(:,j-1)$, where we define both right-hand sides to be zero when $j = 1$. Thus

$$\begin{aligned} M_k(:,j) &= A^{k-1}w_j + (A^{k-2}W + M_{k-2}J)J(:,j) \\ &= A^{k-1}w_j + A^{k-2}w_{j-1} + M_{k-2}J(:,j-1) \\ &= \cdots = \sum_{i=1}^{\min(k,j)} A^{k-i}w_{j-i+1}. \end{aligned}$$

We will write $M_k(:,j) = \sum_{i=1}^{j} A^{k-i}w_{j-i+1}$ on the understanding that when $k < j$ we set to zero the terms in the summation where $i > k$ (i.e., those terms with a negative power of A). From (3.5) and (3.8) we see that the (1,2) block of $X = \varphi_\ell(\tau\widetilde{A})$ is

$$X(1:n, n+1:n+p) = \sum_{k=1}^{\infty} \frac{\tau^k M_k}{(k+\ell)!}.$$

Therefore, the $(n+j)$th column of X is given by

$$\begin{aligned} X(1:n, n+j) &= \sum_{k=1}^{\infty} \frac{\tau^k M_k(:,j)}{(k+\ell)!} = \sum_{k=1}^{\infty} \frac{1}{(k+\ell)!}\left(\sum_{i=1}^{j} \tau^i(\tau A)^{k-i}w_{j-i+1}\right) \\ &= \sum_{i=1}^{j} \tau^i \left(\sum_{k=1}^{\infty} \frac{(\tau A)^{k-i}}{(k+\ell)!}\right) w_{j-i+1} \\ &= \sum_{i=1}^{j} \tau^i \left(\sum_{k=0}^{\infty} \frac{(\tau A)^{k}}{(\ell+k+i)!}\right) w_{j-i+1} = \sum_{i=1}^{j} \tau^i \varphi_{\ell+i}(\tau A)w_{j-i+1}. \qquad \square \end{aligned}$$

With $\tau = 1$, $j = p$, and $\ell = 0$, Theorem 3.2.1 shows that, for arbitrary vectors w_k, the sum of matrix–vector products $\sum_{k=1}^{p} \varphi_k(A)w_{j-k+1}$ can be obtained from the last column of the exponential of a matrix of dimension $n+p$. A special case of the theorem is worth noting. On taking $\ell = 0$ and $W = [c\ 0] \in \mathbb{C}^{n\times p}$, where $c \in \mathbb{C}^n$, we obtain $X(1\colon n, n+j) = \tau^j\varphi_j(\tau A)c$, which is a relation useful for Krylov methods that was derived by Sidje [63, Thm. 1]. This in turn generalizes the expression

$$\exp\left(\begin{bmatrix} A & c \\ 0 & 0 \end{bmatrix}\right) = \begin{bmatrix} e^A & \varphi_1(A)c \\ 0 & 1 \end{bmatrix}$$

obtained by Saad [61, Prop. 1].

We now use the theorem to obtain an expression for (3.4) involving only the matrix exponential. Let $W(:,p-k+1) = u_k$, $k = 1\colon p$, form the matrix $\widetilde{A}$ in (3.6), and set $\ell = 0$ and $\tau = t - t_0$. Then

$$X = \varphi_0\big((t-t_0)\widetilde{A}\big) = e^{(t-t_0)\widetilde{A}} = \begin{bmatrix} e^{(t-t_0)A} & X_{12} \\ 0 & e^{(t-t_0)J} \end{bmatrix}, \tag{3.9}$$

where the columns of X_{12} are given by (3.7), and, in particular, the last column of X_{12} is

$$X(1:n, n+p) = \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k u_k.$$

Hence, by (3.4) and (3.9),

$$\begin{aligned}\widehat{u}(t) &= e^{(t-t_0)A}u_0 + \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k\, u_k \\ &= e^{(t-t_0)A}u_0 + X(1:n, n+p) \\ &= \left[\begin{array}{cc} I_n & 0 \end{array}\right] e^{(t-t_0)\widetilde{A}} \left[\begin{array}{c} u_0 \\ e_p \end{array}\right]. \qquad (3.10)\end{aligned}$$

Thus we are approximating the nonlinear system (3.1) by a subspace of a slightly larger linear system

$$y'(t) = \widetilde{A}y(t), \quad y(t_0) = \left[\begin{array}{c} u_0 \\ e_p \end{array}\right].$$

To evaluate (3.10) we need to compute the action of the matrix exponential on a vector. We focus on this problem in the rest of this chapter.

An important practical matter concerns the scaling of $\widetilde{A}$. If we replace W by ηW we see from (3.7) that the only effect on $X = e^{\widetilde{A}}$ is to replace $X(1:n, n+1:n+p)$ by $\eta X(1:n, n+1:n+p)$. This linear relationship can also be seen using properties of the Fréchet derivative [31, Thm. 4.12]. For methods employing a scaling and squaring strategy a large $\|W\|$ can cause overscaling, resulting in numerical instability. To avoid overscaling a suitable normalization of W is necessary. In the 1-norm we have

$$\|A\|_1 \le \|\widetilde{A}\|_1 \le \max\big(\|A\|_1, \eta\|W\|_1 + 1\big),$$

since $\|J\|_1 = 1$. We choose $\eta = 2^{-\lceil \log_2(\|W\|_1)\rceil}$, which is defined as a power of 2 to avoid the introduction of rounding errors. The variant of the expression (3.10) that we should evaluate is

$$\widehat{u}(t) = \left[\begin{array}{cc} I_n & 0 \end{array}\right] \exp\left((t-t_0)\left[\begin{array}{cc} A & \eta W \\ 0 & J \end{array}\right]\right) \left[\begin{array}{c} u_0 \\ \eta^{-1}e_p \end{array}\right]. \qquad (3.11)$$

Experiment 8 in Section 3.6 illustrates the importance of normalizing W.

3.3 Computing $e^A B$

Let r_m be a rational approximation to the exponential function, which we assume to be good near the origin, and let $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times n_0}$ with $n_0 \ll n$. Choose an integer $s \ge 1$ so that $e^{s^{-1}A}$ is well-approximated by $r_m(s^{-1}A)$. Exploiting the relation

$$e^A B = (e^{s^{-1}A})^s B = \underbrace{e^{s^{-1}A} e^{s^{-1}A} \cdots e^{s^{-1}A}}_{s \text{ times}} B, \qquad (3.12)$$

the recurrence

$$B_{i+1} = r_m(s^{-1}A)B_i, \quad i = 0\colon s-1, \qquad B_0 = B \qquad (3.13)$$

yields the approximation $B_s \approx e^A B$. Since A is possibly large and sparse and we wish to assume only the capability to evaluate matrix products with A, we choose for r_m a truncated Taylor series

$$T_m(s^{-1}A) = \sum_{j=0}^{m} \frac{(s^{-1}A)^j}{j!}. \qquad (3.14)$$

Note that throughout this chapter, "matrix product" refers to the product of an $n \times n$ matrix with an $n \times n_0$ matrix, and this reduces to a matrix–vector product when $n_0 = 1$. We will exploit the backward error analysis of Higham [30], [32], as refined in Chapter 2, for determining the scaling parameter s for a given m. Let

$$\Omega_m = \{\, X \in \mathbb{C}^{n\times n} : \rho(e^{-X}T_m(X) - I) < 1 \,\},$$

where ρ is the spectral radius. Then the function

$$h_{m+1}(X) = \log(e^{-X}T_m(X))$$

is defined for $X \in \Omega_m$, where log denotes the principal logarithm [31, Thm. 1.31], and it commutes with X. Hence for $X \in \Omega_m$ we have $T_m(X) = e^{X+h_{m+1}(X)}$. Now choose s so that $s^{-1}A \in \Omega_m$. Then

$$T_m(s^{-1}A)^s = e^{A+sh_{m+1}(s^{-1}A)} =: e^{A+\Delta A},$$

where the matrix $\Delta A = sh_{m+1}(s^{-1}A)$ represents the backward error resulting from the truncation errors in approximating e^A by $T_m(s^{-1}A)^s$. Over Ω_m, the functions h_{m+1} have a power series expansion

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

We want to ensure that

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{m+1}(s^{-1}A)\|}{\|s^{-1}A\|} \le \mathrm{tol},$$

for any matrix norm and a given tolerance, tol. By Theorem 2.4.2(a) we have

$$\frac{\|\Delta A\|}{\|A\|} = \frac{\|h_{m+1}(s^{-1}A)\|}{\|s^{-1}A\|} \le \frac{\widetilde{h}_{m+1}(s^{-1}\alpha_p(A))}{s^{-1}\alpha_p(A)}, \tag{3.15}$$

where $\widetilde{h}_{m+1}(x) = \sum_{k=m+1}^{\infty} |c_k|\, x^k$ and

$$\alpha_p(A) = \max\bigl(d_p, d_{p+1}\bigr), \qquad d_p = \|A^p\|^{1/p}, \tag{3.16}$$

with p arbitrary subject to $m+1 \ge p(p-1)$. The reason for working with $\alpha_p(A)$ instead of $\|A\|$ is that $\alpha_p(A) \ll \|A\|$ is possible for nonnormal A, so (3.15) is sharper than the bound $\widetilde{h}_{m+1}(s^{-1}\|A\|)/(s^{-1}\|A\|)$. For example, consider

$$A = \begin{bmatrix} 1 & a \\ 0 & -1 \end{bmatrix}, \quad |a| \gg 1, \qquad \|A^{2k}\|_1 = 1, \quad \|A^{2k+1}\|_1 = 1 + |a|, \tag{3.17}$$

for which $d_j = \|A^j\|_1^{1/j} \ll \|A\|_1$ for $j \ge 2$.

Define

$$\theta_m = \max\{\, \theta : \widetilde{h}_{m+1}(\theta)/\theta \le \mathrm{tol} \,\}. \tag{3.18}$$

Then for any m and p with $m+1 \ge p(p-1)$ we have $\|\Delta A\| \le \mathrm{tol}\|A\|$ provided that $s \ge 1$ is chosen so that $s^{-1}\alpha_p(A) \le \theta_m$. For each m, the optimal value of the integer s is given by $s = \max\bigl(\lceil \alpha_p(A)/\theta_m \rceil, 1\bigr)$.

The computational cost of evaluating $B_s \approx e^A B$ by the recurrence (3.13) with $r_m = T_m$ is $C_m(A)$ products of an $n \times n$ matrix with an $n \times n_0$ matrix, where

$$C_m(A) := sm = m \max\bigl(\lceil \alpha_p(A)/\theta_m \rceil, 1\bigr) \tag{3.19}$$

and n_0 is the number of columns of B. Here, we are assuming that (3.13) is evaluated by explicit formation of the matrices $A^k B_i$ [31, Alg. 4.3].

Note that this approach, based on the recurrence (3.13), is related to the scaling and squaring method for computing e^A in that it shares the same form of approximation and backward error analysis, but it does not exploit repeated squaring in the final phase. In the case where $s = 2^k$ and $n_0 = 1$, (3.13) employs $2^k m$ matrix–vector products whereas the scaling and squaring method uses k matrix–*matrix* products in the squaring phase.

The sequence $\{C_m(A)\}$ is found to be generally decreasing, though it is not necessarily monotonic. Indeed the sequence $\{\alpha_p(A)\}$ has a generally nonincreasing trend for any A, and with tol in (3.18) corresponding to single or double precision we find that $\{m/\theta_m\}$ is strictly decreasing. Thus the larger is m, the less the cost. However, a large value of m is generally unsuitable in floating point arithmetic because it leads to the evaluation of $T_m(A)B$ with a large $\|A\|$, and as the analysis in the next section explains, numerical instability may result. Thus we impose a limit $m_{\max}$ on m and obtain the minimizer m_* over all p such that $p(p-1) \le m_{\max} + 1$. For the moment we drop the max in (3.19), whose purpose is simply to cater for nilpotent A with $A^j = 0$ for $j \ge p$. Thus we have

$$C_m(A) = m\lceil \alpha_p(A)/\theta_m \rceil.$$

Note that $d_1 \ge d_k$ in (3.16) for all $k \ge 1$ and so $\alpha_1(A) \ge \alpha_2(A)$. Hence we do not need to consider $p = 1$. Let $p_{\max}$ denote the largest positive integer p such that $p(p-1) \le m_{\max} + 1$. Then the optimal cost is

$$C_{m_*}(A) = \min\bigl\{\, m\lceil \alpha_p(A)/\theta_m \rceil : 2 \le p \le p_{\max},\ \ p(p-1) - 1 \le m \le m_{\max} \,\bigr\}, \tag{3.20}$$

where m_* denotes the smallest value of m at which the minimum is attained. The optimal scaling parameter is $s = C_{m_*}(A)/m_*$, by (3.19). Our experience indicates that $p_{\max} = 8$ and $m_{\max} = 55$ are appropriate choices. The above error and cost analysis are valid for any matrix norm, but it is most convenient to use the 1-norm. As we did in Chapter 2, we will use the block 1-norm estimation algorithm of Higham and Tisseur [34] to approximate the quantities $d_p = \|A^p\|_1^{1/p}$ needed to evaluate $\alpha_p(A)$. This algorithm estimates $\|G\|_1$ via about 2 actions of G and 2 actions of G^*, all on matrices of ℓ columns, where the positive integer ℓ is a parameter that we set to 2. Therefore computing $\alpha_p(A)$ for $p = 2\colon p_{\max}$, and thus d_p for $p = 2\colon p_{\max} + 1$, costs approximately

$$4\ell \sum_{p=2}^{p_{\max}+1} p = 2\ell p_{\max}(p_{\max} + 3) \tag{3.21}$$

matrix–vector products. If

$$\|A\|_1 \le 2 \frac{\ell}{n_0} \frac{\theta_{m_{\max}}}{m_{\max}} p_{\max}(p_{\max} + 3) \tag{3.22}$$

Table 3.1: Selected constants θ_m for tol $= 2^{-24}$ (single precision) and tol $= 2^{-53}$ (double).

m	5	10	15	20	25	30	35	40	45	50	55
single	1.3e-1	1.0e0	2.2e0	3.6e0	4.9e0	6.3e0	7.7e0	9.1e0	1.1e1	1.2e1	1.3e1
double	2.4e-3	1.4e-1	6.4e-1	1.4e0	2.4e0	3.5e0	4.7e0	6.0e0	7.2e0	8.5e0	9.9e0

then the cost—namely $n_0 m_{\max}\|A\|_1/\theta_{m_{\max}}$ matrix–vector products—of evaluating B_s with m determined by using $\|A\|_1$ in place of $\alpha_p(A)$ in (3.19) is no larger than the cost (3.21) of computing the $\alpha_p(A)$, and so we should certainly use $\|A\|_1$ in place of the $\alpha_p(A)$. This observation leads to a significant reduction in cost for some matrices. See Experiments 1 and 2 in Section 3.6 for examples where (3.22) is satisfied. Thus m and s are determined as follows.

Code Fragment 3.3.1 ($[m_*, s] =$ **parameters**(A, tol)**)** *This code determines m_* and s given A, tol, $m_{\max}$, and $p_{\max}$.*

```
if (3.22) is satisfied
    $m_* = \mathrm{argmin}_{1\le m\le m_{\max}} m\lceil \|A\|_1/\theta_m\rceil$
    $s = \lceil \|A\|_1/\theta_m\rceil$
else
    Let $m_*$ be the smallest $m$ achieving the minimum in (3.20).
    $s = \max\bigl(C_{m_*}(A)/m_*, 1\bigr)$
end
```

If we wish to exponentiate the matrix tA for several values of t then, since $\alpha_p(tA) = |t|\alpha_p(A)$, we can precompute the matrix $S \in \mathbb{R}^{(p_{\max}-1)\times m_{\max}}$ given by

$$s_{pm} = \begin{cases} \dfrac{\alpha_p(A)}{\theta_m}, & 2 \le p \le p_{\max},\;\; p(p-1)-1 \le m \le m_{\max}, \\ 0, & \text{otherwise} \end{cases} \tag{3.23}$$

and then for each t obtain $C_{m_*}(tA)$ as the smallest nonzero element in the matrix $\lceil |t|S\rceil \mathrm{diag}(1, 2, \ldots, m_{\max})$, where m_* is the column index of the smallest element. Table 3.1 lists some of the θ_m values corresponding to $u_s = \mathrm{tol} = 2^{-24} \approx 6.0 \times 10^{-8}$ (single precision) and $u_d = \mathrm{tol} = 2^{-53} \approx 1.1 \times 10^{-16}$ (double precision). These values were determined as described in Section 2.7.

3.3.1 Preprocessing and termination criterion

Further reduction of the scaling parameter s can be achieved by choosing an appropriate point about which to expand the Taylor series of the exponential function. For any $\mu \in \mathbb{C}$, both the series $\sum_{k=0}^{\infty} A^k/k!$ and $e^{\mu}\sum_{k=0}^{\infty}(A-\mu I)^k/k!$ yield e^A, but the convergence of the second can be faster if μ is selected so that $\|A-\mu I\| \le \|A\|$. Two different ways to approximate e^A via the matrix $A - \mu I$ are from the expressions

$$e^{\mu}\bigl[T_m(s^{-1}(A-\mu I))\bigr]^s, \qquad \bigl[e^{\mu/s}T_m(s^{-1}(A-\mu I))\bigr]^s.$$

These two expressions are not equivalent numerically. The first is prone to overflow when A has an eigenvalue with large negative real part [31, Sec. 10.7.3], since it explicitly approximates $e^{A-\mu I}$. The second expression avoids this problem and is therefore preferred.

Since we will base our algorithm on the 1-norm, the most natural choice of μ is the one that minimizes $\|A-\mu I\|_1$, for which an explicit expression is given in [31, Thm. 4.21]. However, it is the values $d_p(A) = \|A^p\|_1^{1/p}$ in (3.16), not $\|A\|_1$, that govern the construction of our approximation, and choosing the shift to minimize $\|A-\mu I\|_1$ does not necessarily produce the smallest values of $d_p(A-\mu I)$. Indeed we have found empirically that the shift that minimizes the Frobenius norm $\|A-\mu I\|_F$, namely $\mu = \mathrm{trace}(A)/n$, leads to smaller values of the d_p for the 1-norm. A partial explanation follows from the observation that if $A = QTQ^*$ is a Schur decomposition then $(A-\mu I)^p = Q(T-\mu I)^p Q^*$. Hence if $A-\mu I$ has zero trace then $T-\mu I$ has diagonal elements with both positive and negative real parts, and this tends to result in cancellation of any large off-diagonal elements when the matrix is powered, as illustrated by (3.17).

Importantly, incorporating shifts does not vitiate the backward error analysis above: if we choose m_* based on the $\alpha_p(A-\mu I)$ values, the same backward error bounds can be shown to hold.

Another way to reduce the norm is by balancing. Balancing is a heuristic that attempts to equalize the norms of the ith row and ith column of A, for each i, by a diagonal similarity transformation, $\widetilde{A} = D^{-1}AD$. The balancing algorithm available in LAPACK and MATLAB uses the 1-norm and also attempts to permute the matrix to block upper triangular form, something that is important for the eigenvalue problem but not relevant to the computation of $e^A B$. With balancing, we compute $e^A B = De^{\widetilde{A}}D^{-1}B$. There is no guarantee that $\|\widetilde{A}\|_1 < \|A\|_1$, or that the $\alpha_p(A)$ values are reduced; we would certainly not use balancing if $\|\widetilde{A}\|_1 > \|A\|_1$. Balancing affects the backward error analysis: the best backward error bound for A involves an extra factor $\kappa(D)$, though in practice this factor is not seen (see Experiment 1 in Section 3.6). In the context of the eigenvalue problem it is known that balancing can seriously degrade accuracy in special cases [71]. We regard balancing as an option to be used with care and not always to be automatically applied.

The derivation of the θ_m takes no account of the matrix B, so our choice of m is likely to be larger than necessary for some B. We know that our procedure returns $e^{A+\Delta A}B$ with normwise relative backward error $\|\Delta A\|/\|A\| \le \mathrm{tol}$. We now consider truncating the evaluation of $T_m(A)B_i$ in (3.13), and in so doing allow a normwise relative forward error of at most tol to be introduced. With A denoting the scaled and shifted matrix, we will accept $T_k(A)B_i$ for the first k such that

$$\frac{\|A^{k-1}B_i\|}{(k-1)!} + \frac{\|A^k B_i\|}{k!} \le \mathrm{tol}\,\|T_k(A)B_i\|. \tag{3.24}$$

The left-hand side of (3.24) is meant to approximate the norm of the tail of the series, $\sum_{j=k+1}^{m} A^j B_i/j!$. Taking two terms rather than one better captures the behavior of the tail, as illustrated by (3.17); we have found empirically that two terms gives reliable performance.

Our algorithm for computing $e^{tA}B$, where the scalar parameter t is now included for convenience, is summarized as follows. The algorithm is intended for use with tol $= u_s$ or tol $= u_d$, for which we know the corresponding θ_m values (see Table 3.1). However, it is straightforward to determine (once and for all) the θ_m corresponding to any other value of tol.

Algorithm 3.3.2 **($F = \mathbf{F}(t, A, B, \mathbf{balance})$)** *Given $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times n_0}$, $t \in \mathbb{C}$, and a tolerance* tol*, this algorithm produces an approximation $F \approx e^{tA}B$. The logical variable balance indicates whether or not to apply balancing.*

```
 1  if balance
 2     $\widetilde{A} = D^{-1}AD$
 3     if $\|\widetilde{A}\|_1 < \|A\|_1$, $A = \widetilde{A}$, $B = D^{-1}B$, else balance = false, end
 4  end
 5  $\mu = \mathrm{trace}(A)/n$
 6  $A = A - \mu I$
 7  if $t\|A\|_1 = 0$, $m_* = 0$, $s = 1$, goto 9, end   % The case $tA = 0$.
 8  $[m_*, s] = \mathrm{parameters}(tA)$ % Code Fragment 3.3.1
 9  $F = B$, $\eta = e^{t\mu/s}$
10  for $i = 1\colon s$
11      $c_1 = \|B\|_\infty$
12      for $j = 1\colon m_*$
13          $B = tAB/(sj)$, $c_2 = \|B\|_\infty$
14          $F = F + B$
15          if $c_1 + c_2 \le \mathrm{tol}\|F\|_\infty$, quit, end
16          $c_1 = c_2$
17      end
18      $F = \eta F$, $B = F$
19  end
20  if balance, $F = DF$, end
```

The cost of the algorithm is determined by the number of matrix products; these products occur at line 13 and in the parameters function.

Note that when $n_0 > 1$ we could simply invoke Algorithm 3.3.2 n_0 times to compute $e^A b_j$, $j = 1\colon n_0$, which may require fewer flops than a single invocation of $e^A B$, since the termination test at line 15 may be satisfied earlier for some b_j than for B as whole. The advantage of working with B is the ability to invoke level 3 BLAS [19] [20], which should lead to faster execution.

3.4 Rounding error analysis and conditioning

To assess the numerical stability of Algorithm 3.3.2 in floating point arithmetic we analyze the rounding errors in forming the product $T_m(A)B$, where $T_m(A)$ is the truncated Taylor series (3.14). For simplicity we assume that A does not need scaling (that is, $s = 1$). We then determine the conditioning of the problem and see if our forward error bound reflects the conditioning. We know that $T_m(A) = e^{A+E}$ with $\|E\| \le \mathrm{tol}\|A\|$. The analysis includes two parameters: the backward error in the approximation of the exponential, tol, and the precision of the underlying floating point arithmetic, u. The norm is the 1-norm or the ∞-norm.

Lemma 3.4.1 *Let $X = T_m(A)B$ be formed as in Algorithm* 3.3.2*, but for simplicity ignoring lines* 5*,* 6*, and* 15*, and assume that $s = 1$ in that algorithm with tolerance* tol*. Then the computed $\widehat{X}$ in floating point arithmetic with unit roundoff $u \le \mathrm{tol}$ satisfies $\widehat{X} = e^{A+E}B + R$, where $\|E\| \le \mathrm{tol}\|A\|$ and $\|R\| \le \widetilde{\gamma}_{mn} T_m(\|A\|)\|B\| \le \widetilde{\gamma}_{mn} e^{\|A\|}\|B\|$.*

Proof. Standard error analysis for the evaluation of matrix products [29, Sec. 3.5] shows that $\|X - \widehat{X}\| \le \widetilde{\gamma}_{mn} T_m(\|A\|)\|B\|$. The analysis in Section 3.3 shows that $X = e^{A+E}B$, with $\|E\| \le \mathrm{tol}\|A\|$. The result follows on using $T_m(\|A\|) \le e^{\|A\|}$. □

Lemma 3.4.1 shows that $\widehat{X}$ satisfies a mixed forward–backward error result where the normwise relative backward error bound is tol and the forward error bound is a multiple of $ue^{\|A\|}\|B\|$. Since $\|A\|$ can exceed 1 the forward error bound is potentially large. To judge whether the forward error bound is acceptable we compare it with a perturbation analysis for the problem.

We derive a perturbation result for the product $X = f(A)B$, where f is an arbitrary matrix function and then specialize it to the exponential. As in Section 1.6, we denote by $L_f(A, \varDelta A)$ the Fréchet derivative of f at A in the direction $\varDelta A$ and by vec the operator that stacks the columns of its matrix argument into a long vector. We will use the fact that $\mathrm{vec}(L_f(A, \varDelta A)) = K_f(A)\,\mathrm{vec}(\varDelta A)$, with $K_f(A)$ the $n^2 \times n^2$ Kronecker matrix representation of the Fréchet derivative. The following lemma makes no assumption about $\|A\|$.

Lemma 3.4.2 *Let $X = f(A)B$ and $X + \varDelta X = f(A + \varDelta A)(B + \varDelta B)$ both be defined, where $\|\varDelta A\|_F \le \epsilon\|A\|_F$ and $\|\varDelta B\|_F \le \epsilon\|B\|_F$. Then, assuming that f is Fréchet differentiable at A,*

$$\frac{\|\varDelta X\|_F}{\|X\|_F} \le \epsilon\left(\frac{\|f(A)\|_2\|B\|_F}{\|X\|_F} + \frac{\|(B^T \otimes I)K_f(A)\|_2\|A\|_F}{\|X\|_F}\right) + o(\epsilon), \qquad (3.25)$$

and this bound is attainable to within a factor 2 *to first order in ϵ.*

Proof. We have

$$X + \varDelta X = \big(f(A) + L_f(A, \varDelta A) + o(\|\varDelta A\|_F)\big)(B + \varDelta B).$$

Applying the vec operator, and using the fact that $\mathrm{vec}(UV) = (V^T \otimes I)\,\mathrm{vec}(U)$, gives

$$\mathrm{vec}(\varDelta X) = \mathrm{vec}(f(A)\varDelta B) + (B^T \otimes I)K_f(A)\,\mathrm{vec}(\varDelta A) + o(\epsilon).$$

Taking the 2-norm and exploiting the relation $\|\mathrm{vec}(X)\|_2 = \|X\|_F$ we have

$$\|\varDelta X\|_F \le \epsilon\|f(A)\|_2\|B\|_F + \epsilon\|(B^T \otimes I)K_f(A)\|_2\|A\|_F + o(\epsilon),$$

which is equivalent to (3.25). Since $\varDelta B$ and $\varDelta A$ are arbitrary it is clear that $\|\varDelta X\|_F$ can attain each of the first two terms in the latter bound with only one of $\varDelta B$ and $\varDelta A$ nonzero, and hence the bound is attainable to within a factor 2 to first order. □

In view of the lemma we can regard

$$\kappa_f(A, B) := \frac{\|f(A)\|_2\|B\|_F}{\|X\|_F} + \frac{\|(B^T \otimes I)K_f(A)\|_2\|A\|_F}{\|X\|_F} \qquad (3.26)$$

as a condition number for the $f(A)B$ problem. We can weaken this expression to

$$\kappa_f(A,B) \le \frac{\|f(A)\|_F\|B\|_F}{\|X\|_F}(1+\kappa_f(A)), \tag{3.27}$$

where (see Section 1.6)

$$\kappa_f(A) := \frac{\|L_f(A)\|_F\|A\|_F}{\|f(A)\|_F}, \qquad \|L_f(A)\|_F := \max_{Z\ne 0}\frac{\|L_f(A,Z)\|_F}{\|Z\|_F} = \|K_f(A)\|_2. \tag{3.28}$$

Applying (3.27) with f the exponential gives

$$\kappa_{\exp}(A,B) \le \frac{\|e^A\|_F\|B\|_F}{\|X\|_F}(1+\kappa_{\exp}(A)). \tag{3.29}$$

For comparison with Lemma 3.4.1 we will, for simplicity, replace all norms by the 2-norm, since constant factors are not important. For the condition number in the 2-norm we have [31, Lem. 10.15]

$$\|A\|_2 \le \kappa_{\exp}(A) \le \frac{e^{\|A\|_2}\|A\|_2}{\|e^A\|_2}. \tag{3.30}$$

If $\kappa_{\exp}(A)$ is close to its upper bound in (3.30) then the forward error bound from Lemma 3.4.1 is smaller than the bound for $\kappa_{\exp}(A,B)$ in (3.29) by a factor $\|A\|_2$. However, there is equality in the lower bound (3.30) for normal A [31, Lem. 10.16], [69]. So for normal A, the normwise relative forward error bound for $\|R\|_2/\|X\|_2$ from Lemma 3.4.1 exceeds $\kappa_{\exp}(A,B)u$ by about

$$e^{\|A\|_2}/(\|e^A\|_2(1+\|A\|_2)), \tag{3.31}$$

which ranges between $1/(1+\|A\|_2)$ and $e^{2\|A\|_2}/(1+\|A\|_2)$. For Hermitian A, the upper bound is attained when A is negative semidefinite. However, when we apply this analysis to Algorithm 3.3.2 A refers to the *shifted* matrix $A-(\operatorname{trace}(A)/n)I$, which has extremal eigenvalues of equal magnitude and opposite signs, and so (3.31) always attains its lower bound. Thus for Hermitian matrices our shifting strategy ensures stability.

An example is instructive. Let $A = \operatorname{diag}(-20.5,-1)$ and $B = [1\ 1]^T$. With $x = e^A b$ and $\widehat{x}$ the computed product from Algorithm 3.3.2 with tol $= u_d$, we find that

$$\frac{\|x-\widehat{x}\|_2}{\|x\|_2} = 6.0\times 10^{-16}, \qquad \frac{|x_1-\widehat{x}_1|}{|x_1|} = 2.6\times 10^{-9}, \qquad \frac{|x_2-\widehat{x}_2|}{|x_2|} = 6.0\times 10^{-16}.$$

Since $\widetilde{A} := A - \operatorname{trace}(A)/2 = \operatorname{diag}(-9.75, 9.75)$, Algorithm 3.3.2 takes $s=1$ and therefore evaluates a truncated Taylor series for the unscaled matrix $\widetilde{A}$. This leads to substantial cancellation in the first component, since the terms in Taylor series for $e^{-9.75}$ grow substantially before they decay, but the second component is computed accurately. While there is loss of accuracy in the smaller component, in the normwise sense the computation is entirely satisfactory, as the analysis above predicts, and normwise stability is all we can expect of an algorithm designed for general A.

The conclusion from this analysis is that it is desirable to keep $\|A\|$ small in order for Algorithm 3.3.2 to reflect the conditioning of the problem, but that a large $\|A\|$ does not necessarily imply numerical instability for nonnormal A.

3.5 Computing $e^{tA}B$ over a time interval

In practice, it may be required to evaluate $e^{tA}B$ for several values of t belonging to a time interval $[t_0, t_q]$. Suppose that q equally spaced steps are to be taken. Denote the grid points by $t_k = t_0 + kh$, $k = 0\colon q$, where $h = (t_q - t_0)/q$. If $e^{t_0 A}B$ is available and Algorithm 3.3.2, applied to $e^{(t_q - t_0)A}$, selects a scaling parameter s equal to q, then the algorithm automatically generates the required matrices as intermediate quantities, as is clear from (3.12). In general, though, we need an efficient strategy for computing these matrices. The most obvious way to evaluate $B_k = e^{t_k A}B$, $k = 0\colon q$, is directly from the formula, using Algorithm 3.3.2. However, since the cost of the algorithm is proportional to $\alpha_p(tA) = |t|\alpha_p(A)$, it is more efficient if we reduce each t_k by t_0 by forming $B_0 = e^{t_0 A}B$ and then (recall that $\mathbf{F}$ denotes an invocation of Algorithm 3.3.2)

$$B_k = \mathbf{F}(kh, A, B_0), \qquad k = 1\colon q. \tag{3.32}$$

A further reduction in cost accrues if we obtain each B_k from the preceding one:

$$B_k = \mathbf{F}(h, A, B_{k-1}), \qquad k = 1\colon q. \tag{3.33}$$

In deciding on the best approach we need to consider the effects of rounding errors. We have seen in Chapter 2 that the scaling and squaring method for e^A can suffer from overscaling, which occurs when the initial scaling $A \leftarrow 2^{-i}A$ reduces $\|A\|$ by more than is necessary to achieve the required accuracy and the resulting extra squarings degrade the accuracy due to propagation of rounding errors in the squaring phase. The same danger applies here, but now overscaling can be caused by a too-small stepsize h. The danger is illustrated by the example of computing $(1 + x/100)^{100}$ when x is so small that $e^x \approx 1 + x$ is a good enough approximation; the former expression is clearly much more seriously affected by rounding errors. The gist of the matter can be seen by considering how B_1 and B_2 are computed; both (3.32) and (3.33) compute $B_1 = e^{hA}B_0$, but (3.32) computes $B_2 = e^{2hA}B_0$ while (3.33) uses $B_2 = e^{hA}B_1 = e^{hA}(e^{hA}B_0)$. It may be that $2hA$ needs no scaling (i.e., Algorithm 3.3.2 chooses $s = 1$ when applied to $2hA$), so that B_1 and B_2 are obtained at exactly the same cost from (3.32) as from (3.33). Although these two ways of obtaining B_2 are equivalent in exact arithmetic, in floating point arithmetic the formula $B_2 = e^{hA}(e^{hA}B_0)$ is more likely to suffer from overscaling.

The following algorithm reduces the chance of overscaling with no cost penalty. It uses the direct formula (3.32) whenever it can do so without increasing the cost (that is, without scaling), but uses (3.33) when (3.32) requires scaling and (3.33) does not.

Code Fragment 3.5.1 *This algorithm computes $B_k = e^{t_k A}B$ for $k = 0\colon q$, where $t_k = t_0 + kh$ and $h = (t_q - t_0)/q$, for the case where $q > s_*$, where s_* is the value determined by Algorithm* 3.3.2 *applied to* $(t_q - t_0)A$.

```
$s = s_*$
$d = \lfloor q/s \rfloor$, $j = \lfloor q/d \rfloor$, $r = q - dj$
$h = (t_q - t_0)/q$
$Z = \mathbf{F}(t_0, A, B)$    % $B_0$
$\widetilde{d} = d$
for $i = 1\colon j + 1$
```

```
7       if $i > j$, $\widetilde{d} = r$, end
8       for $k = 1\colon \widetilde{d}$
9           $B_{k+(i-1)d} = \mathbf{F}(kh, A, Z)$
10      end
11      if $i \le j$, $Z = B_{id}$, end
12  end
```

Note that the same parameter s, namely $s = 1$, is used on each invocation of Algorithm 3.3.2 on line 9 of Code Fragment 3.5.1. Some useful computational savings are possible in line 9 by saving and re-using matrix products. We have

$$T_m(kh, A, Z) = \sum_{\ell=0}^{m} \frac{(khA)^\ell Z}{\ell!} = \underbrace{\begin{bmatrix} Z & (hA)Z & \dots & \frac{1}{m!}(hA)^m Z \end{bmatrix}}_{K_m} \begin{bmatrix} 1 \\ k \\ \vdots \\ k^m \end{bmatrix}. \tag{3.34}$$

When invoked at line 9 of Code Fragment 3.5.1, Algorithm 3.3.2 generally increases the value of m_* as k increases until m_* reaches its maximal value (not necessarily $m_{\max}$) at $k = \widetilde{d}$. It would be enough to form the matrix K_m for the maximal value of m and reuse it for the smaller values of k. However, this would destroy the computational saving obtained from the stopping test that Algorithm 3.3.2 uses. Instead, we will build up the required block columns of K_m gradually during the computation by using the stopping test.

With the aid of Code Fragment 3.5.1 and Algorithm 3.3.2, we can write the final algorithm. We will use the notation $X_{:,j}$ to denote the jth block column of the $n \times n_0(q+1)$ matrix X partitioned into $n \times n_0$ blocks; if $n_0 = 1$ (so that B is a vector) then $X_{:,j} = X(:, j)$ in the usual notation.

Algorithm 3.5.2 *Given $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times n_0}$, and a tolerance* tol*, this algorithm computes a matrix $X \in \mathbb{C}^{n\times n_0(q+1)}$ such that $X_{:,k+1} \approx e^{t_k A}B$, $k = 0\colon q$, where $t_k = t_0 + kh$ and $h = (t_q - t_0)/q$. The logical variable balance indicates whether or not to apply balancing.*

```
if balance
    $\widetilde{A} = D^{-1}AD$
    if $\|\widetilde{A}\|_1 < \|A\|_1$, $A = \widetilde{A}$, $B = D^{-1}B$, else balance = false, end
end
$\mu = \mathrm{trace}(A)/n$
$[m_*, s] = \mathrm{parameters}((t_q - t_0)(A - \mu I))$
$X_{:,1} = \mathbf{F}(t_0, A, B, \mathrm{false})$
if $q \le s$
    for $k = 1\colon q$
        $X_{:,k+1} = \mathbf{F}(h, A, X_{:,k}, \mathrm{false})$
    end
    if balancing was used, $X = DX$, end
    quit
end
$A = A - \mu I$
```

```
$d = \lfloor q/s \rfloor$, $j = \lfloor q/d \rfloor$, $r = q - dj$, $\widetilde{d} = d$
Compute $C_{m_*}(dA)$ from (3.20).
$Z = X(:,1)$
for $i = 1\colon j+1$
    if $i > j$, $\widetilde{d} = r$, end
    $K_{:,1} = Z$, $\widehat{m} = 0$
    for $k = 1\colon \widetilde{d}$
        $F = Z$, $c_1 = \|Z\|_\infty$
        for $p = 1\colon m_*$
            if $p > \widehat{m}$
                $K_{:,p+1} = hAK_{:,p}/p$  % Form $K_{:,p+1}$ if not already formed.
            end
            $F = F + k^p K_{:,p+1}$
            $c_2 = k^p \|K_{:,p+1}\|_\infty$
            if $c_1 + c_2 \le \mathrm{tol}\|F\|_\infty$, quit, end
            $c_1 = c_2$
        end
        $\widehat{m} = \max(\widehat{m}, p)$
        $X_{:,k+(i-1)d+1} = e^{kh\mu} F$
    end
    if $i \le j$, $Z = X_{:,id+1}$, end
end
if balance, $X = DX$, end
```

3.6 Numerical experiments

We give a variety of numerical experiments to illustrate the efficiency and accuracy of our algorithms. All were carried out in MATLAB 7.10 (R2010a) on machines with Core i7 or Core 2 Duo E6850 processors, and errors are computed in the 1-norm, unless stated otherwise.

In some of our experiments we will employ two existing codes from the literature, both of which use Krylov techniques along with time-stepping to traverse the interval $[0, t]$. The MATLAB function `expv` of Sidje [63] evaluates $e^{tA}b$ using a Krylov method with a fixed dimension (default 30) for the Krylov subspace. The MATLAB function `phipm` of Niesen [57] uses Krylov techniques to compute a sum of the form $\sum_{k=0}^{p} \varphi_k(tA)u_k$. The size of the Krylov subspace is changed dynamically during the integration, and the code automatically recognizes when A is Hermitian and uses the Lanczos process instead of the Arnoldi process. We use both functions with their default parameters, except for the convergence tolerance, which varies in our experiments.

We will not attempt to show the benefits of using the $\alpha_p(A)$ in place of $\|A\|_1$, as these have already been demonstrated in Chapter 2 for the scaling and squaring algorithm. In all our experiments B is a vector, b.

Experiment 1. The first experiment tests the behavior of Algorithm 3.3.2 in floating point arithmetic. We use a combination of all four sets of test matrices described

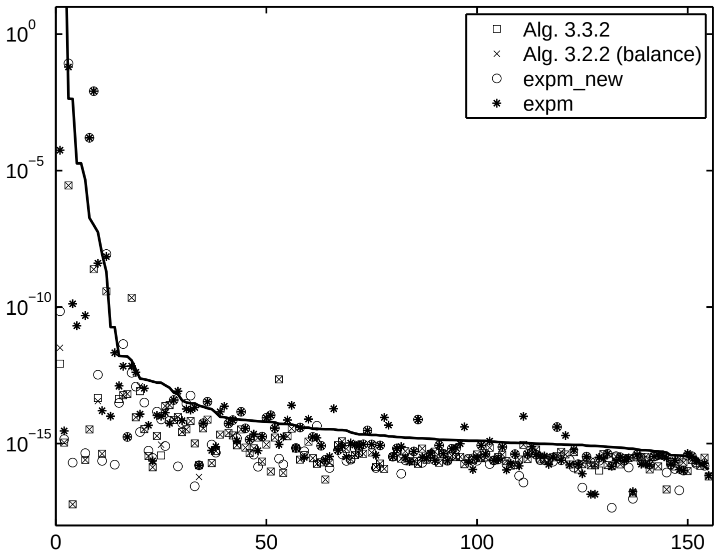

Figure 3.1: Experiment 1: normwise relative errors in $e^{A}b$ computed by Algorithm 3.3.2 with and without balancing and by first computing e^{A} by `expm` or `expm_new`. The solid line is $\kappa_{\exp}(A,b)u_d$.

in Section 2.6, giving 155 matrices in total, with dimensions n up to 50. For each matrix A, and a different randomly generated vector b for each A, we compute $x = e^{A}b$ in three ways:

- using Algorithm 3.3.2 with and without balancing, with $\mathrm{tol} = u_d$,
- as `expm(A)*b`, where `expm` is the MATLAB function for the matrix exponential, which implements the scaling and squaring algorithm of [30], [32],
- as `expm_new(A)*b`, where `expm_new` implements Algorithm 2.6.1.

Figure 3.1 displays the relative errors $\|e^{A}b - \widehat{x}\|_2/\|e^{A}b\|_2$, where the "exact" answer is obtained by computing at 100 digit precision with the Symbolic Math Toolbox. The matrices are sorted by decreasing value of the condition number $\kappa_{\exp}(A,b)$ in (3.26), and $\kappa_{\exp}(A,b)u_d$ is shown as a solid line. We compute $\kappa_{\exp}(A,b)$ exactly, using the function `expm_cond` in the Matrix Function Toolbox [26] to obtain the Kronecker matrix representation $K_{\exp}(A)$ of the Fréchet derivative. Figure 3.2 displays the same data as a performance profile, where for a given α the corresponding point on each curve indicates the fraction p of problems on which the method had error at most a factor α times that of the smallest error over all methods in the experiment.

Figure 3.1 reveals that all the algorithms behave in a generally forward stable manner. Figure 3.2 shows that Algorithm 3.3.2 has the best accuracy, beating both `expm` and the more accurate `expm_new`. Balancing has little effect on the errors, but it can greatly reduce the cost: the quantity "s without balancing divided by s with balancing" had maximum value 1.6×10^{4} and minimum value 0.75, with the two values of s differing in 11 cases. The test (3.22) was satisfied in about 77% of the cases.

Experiment 2. In this experiment we take for A the matrix `gallery('lesp',10)`, which is a nonsymmetric tridiagonal matrix with real, negative eigenvalues, and $b_i =$

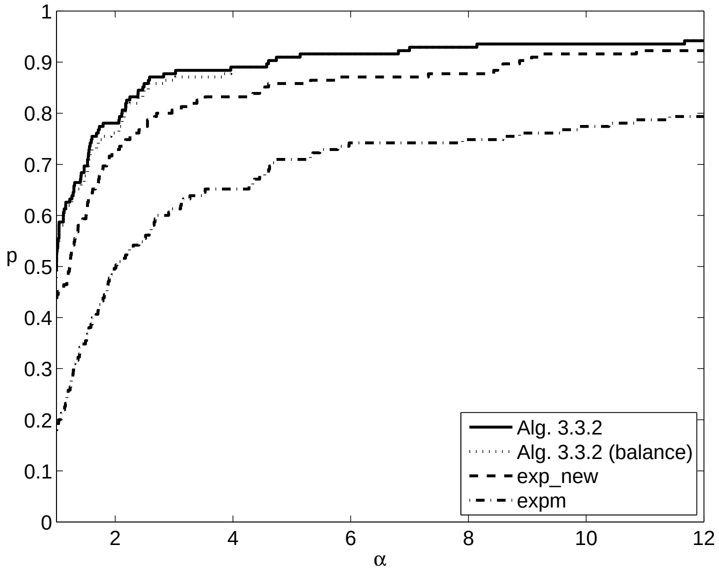

Figure 3.2: Same data as in Figure 3.1 presented as a performance profile.

i. We compute $e^{tA}b$ by Algorithm 3.3.2 for 50 equally spaced $t \in [0, 100]$, with and without balancing, for tol $= u_s$ and tol $= u_d$. The results are shown in Figure 3.3. We see a linear growth of the cost of the algorithm with t, as expected. The relative errors are all below the corresponding solid line representing $\kappa_{\mathrm{exp}}(tA, b)u$, which shows that the algorithm is behaving in a forward stable manner. Balancing has no significant effect on the error but leads to a useful reduction in cost. In this test the inequality (3.22) was satisfied in 5% of the cases.

Experiment 3. Now we investigate the effectiveness of Algorithm 3.5.2 at avoiding overscaling when dense output is requested over an interval. We take for A the matrix `gallery('frank',3)`, b has equally spaced elements on $[-1, 1]$, and tol $=$ u_d; balancing leaves this matrix unchanged. We apply Algorithm 3.5.2 twice with $t \in [0, 10]$ and $q = 200$, once in its given form and again with the "if" test at line 8 forced to be satisfied, thus turning off the logic for avoiding overscaling. The relative errors are shown in Figure 3.4. The improved accuracy provided by our strategy for avoiding overscaling is clear.

Experiment 4. Our next experiment is a variation of one from Trefethen, Weideman, and Schmelzer [68, Sec. 3], in which $A \in \mathbb{R}^{9801\times 9801}$ is a multiple of the standard finite difference discretization of the 2D Laplacian, namely the block tridiagonal matrix constructed by `-2500*gallery('poisson',99)` in MATLAB. We compute $e^{\alpha t A}b$ for the b specified in [68] for $q = 100$ equally spaced values of t on $[0, 1]$, with $\alpha = 0.02$ (the problem as in [68]) and $\alpha = 1$. We use four different methods.

1. Algorithm 3.5.2, with tol $= u_d$. Since A is symmetric, balancing is not needed.
2. The MATLAB functions `expv` and `phipm`, both called q times with error tolerance u_d. Since both functions use a time-stepping strategy to advance $e^{\tau A}b$ from $\tau = 0$ to $\tau = t$, our repeated calls to `expv` and `phipm` result in wasted computation, but they are unavoidable because the functions do not offer the option of "dense output" to return intermediate vectors.

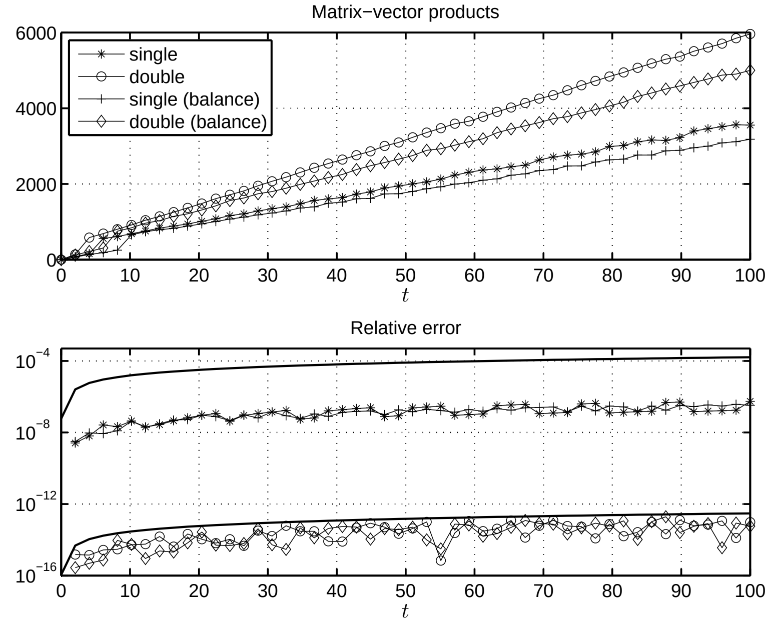

Figure 3.3: Experiment 2: t versus cost (top) and accuracy (bottom) of Algorithm 3.3.2 with and without balancing for $e^{tA}b$. In the bottom plot the solid lines are $\kappa_{\exp}(tA,b)u_s$ and $\kappa_{\exp}(tA,b)u_d$.

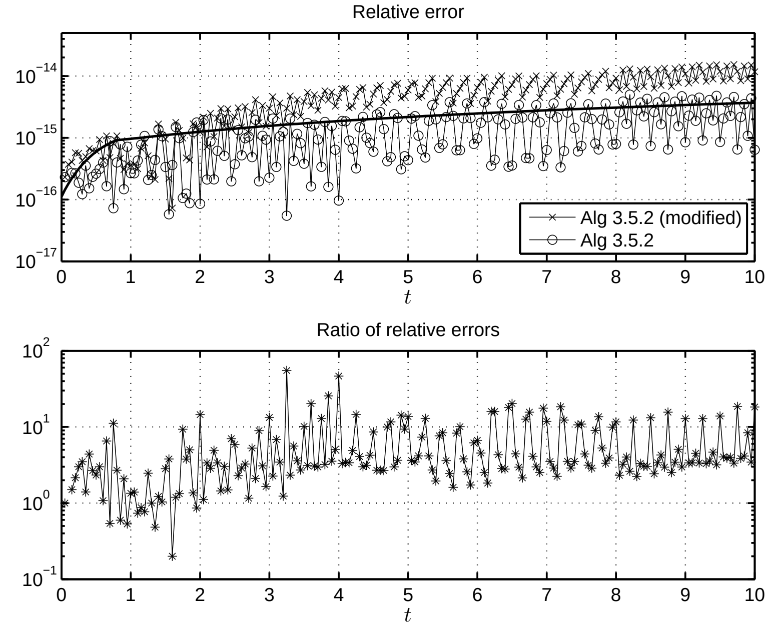

Figure 3.4: Experiment 3: relative errors (top) for Algorithm 3.5.2 and for modified version of the algorithm without the logic to avoid overscaling, and ratio of relative errors “modified/original” (bottom). The solid line is $\kappa_{\exp}(tA,b)u_d$.

Table 3.2: Experiment 4: speed is time for method divided by time for Algorithm 3.5.2.

	$\alpha = 0.02$			$\alpha = 1$		
	speed	cost	diff	speed	cost	diff
Algorithm 3.5.2	1	1119		1	49544	
`expv`	46.6	25575	4.5e-15	66.0	516429	6.2e-14
`phipm`	10.5	10744	5.5e-15	9.3	150081	6.3e-14
rational	107.8	700	9.1e-14	7.9	700	1.0e-12

3. The MATLAB code in [68, Fig. 4.4], which uses a best rational L_∞ approximation to the exponential of type $(N-1, N)$ with $N = 14$. It is called repeatedly for each t-value. Unlike the previous two methods, the dominant computational cost is the solution of linear systems with matrices of the form $\alpha I - \beta A$, which is done via the backslash operator.

The results are given in Table 3.2, where “cost” denotes the number of matrix–vector products for the first three methods and the number of linear system solves for the rational approximation method, and “diff” is the normwise relative difference between the matrix of result vectors computed by Algorithm 3.5.2 and the other methods.

Here, $\|A\|_1 = 2\times 10^4$, and Algorithm 3.5.2 takes $s = 21$ for $\alpha = 0.02$ and $s = 1014$ for $\alpha = 1$. In spite of the fairly heavy scaling, this algorithm is still substantially faster than `expv`, `phipm`, and the rational approximation method. Algorithm 3.5.2 needed to use its logic to avoid overscaling for $\alpha = 0.02$ but not for $\alpha = 1$.

We used the MATLAB `profile` function to profile the M-files in this experiment. We found that Algorithm 3.5.2 spent 88% of its time doing matrix–vector products. By contrast, `expv` and `phipm` spent 12% and 28% of their time, respectively, on matrix–vector products and most of the rest within the Arnoldi recurrence or in evaluating matrix exponentials via `expm` (1% and 6%, respectively). Note that the vectors in the Arnoldi and Lanczos recurrences are generally full if b is, so the inner products and additions in these recurrences can be of similar cost to a matrix–vector product when A is very sparse, as it is here.

Experiment 5. This experiment uses essentially the same tests as Niesen and Wright [58, Experiment 1], which in turn are based on those of Sidje [63]. The three matrices belong to the Harwell-Boeing collection and are obtained from the University of Florida Sparse Matrix Collection [14], [15]. For the first two matrices we compute $e^{At}b$. The matrices and problem details are:

- `orani678`, $n = 2529$, $t = 100$, $b = [1, 1, \ldots, 1]^T$;
- `bcspwr10`, $n = 5300$, $t = 10$, $b = [1, 0, \ldots, 0, 1]^T$.

The third matrix is `gr_30_30`, with $n = 900$, $t = 2$, $b = [1, 1, \ldots, 1]^T$, and we compute $e^{-tA}e^{tA}b$. The tolerance is u_s and for the first two problems we regard the solution computed with Algorithm 3.3.2 with tol $= u_d$ as the exact solution. Balancing is not applied because MATLAB does not support balancing of matrices stored with the sparse attribute; however, balancing is certainly possible for sparse matrices, and several algorithms are developed by Chen and Demmel [10]. The results are shown

Table 3.3: Experiment 5: speed is time for method divided by time for Algorithm 3.3.2.

		Algorithm 3.3.2			`phipm`			`expv`		
	$\|A\|_1$	speed	cost	error	speed	cost	error	speed	cost	error
`orani678`	1.0e5	1	1278	1.6e-16	5.9	651	1.2e-13	19.4	8990	4.0e-14
`bcspwr10`	1.4e2	1	338	2.0e-15	1.2	103	6.6e-15	5.5	341	2.6e-14
`gr_30_30`	3.2e1	1	80	7.2e-9	1.5	40	1.9e-9	3.0	124	9.5e-8

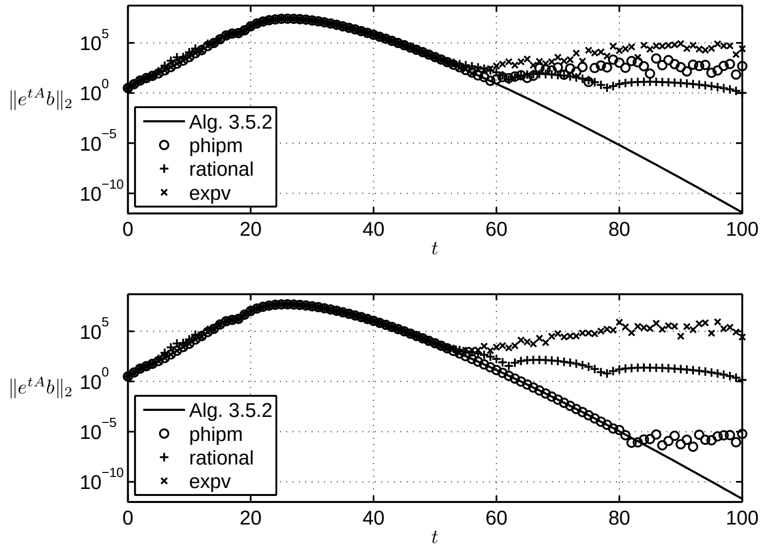

Figure 3.5: Experiment 6: t versus $\|e^{tA}b\|_2$, with $\alpha = 4$ (top) and $\alpha = 4.1$ (bottom).

in Table 3.3. All three methods deliver the required accuracy, but Algorithm 3.3.2 proves to be the fastest.

Experiment 6. This example reveals the smoothness of the solution computed by Algorithm 3.5.2. The matrix A is `-gallery('triw',20,alpha)`, which is upper triangular with constant diagonal -1 and all superdiagonal elements equal to $-\alpha$. We take $b_i = \cos i$ and compute the norms $\|e^{tA}b\|_2$ with tol $= u_d$ for $\alpha = 4$ and $\alpha = 4.1$ by all the methods discussed above for $t = 0\colon 100$. The best rational L_∞ approximation is applicable since A has real, negative eigenvalues. Balancing has no effect on this matrix. Figure 3.5 shows that Algorithm 3.5.2 is the only method to produce a smooth curve that tracks the growth and decay of the exponential across the whole interval, and indeed each computed norm for Algorithm 3.5.2 has relative error less than 5×10^{-14}. The accuracy of the other methods is affected by the nonnormality of A, which is responsible for the hump. The rational approximation method solves linear systems with condition numbers up to order 10^{12}, which causes its ultimate loss of accuracy. The Krylov methods are so sensitive to rounding errors that changing α from 4 to 4.1 produces quite different results, and qualitatively so for `phipm`. The problem is very ill-conditioned: $\kappa_{\exp}(A, b) \ge u_d^{-1}$ for $t \gtrsim 53$.

Experiment 7. Since our approach is in the spirit of the sixth of Moler and Van Loan's "19 dubious ways" [54, Sec. 4], it is appropriate to make a comparison with a direct implementation of their "Method 6: single step ODE methods". For the Laplacian matrix and vector b from Experiment 4, we compute $e^{\alpha A}b$, for $\alpha = 1$ and $\alpha = 4$,

Table 3.4: Experiment 7: cost is the number of matrix–vector products, except for `ode15s` for which it "number of matrix–vector products/number of LU factorizations/number of linear system solves". The subscript on `poisson` denotes the value of α.

	Algorithm 3.3.2			`ode45`			`ode15s`		
	Time	Cost	Error	Time	Cost	Error	Time	Cost	Error
`orani678`	0.13	878	4.0e-8	3.29	14269	2.7e-8	132	7780/606/7778	1.6e-6
`bcspwr10`	0.021	215	7.2e-7	0.54	3325	9.5e-7	2.96	1890/76/1888	4.8e-5
`poisson`$_1$	3.76	29255	2.2e-6	8.6	38401	3.7e-7	2.43	402/33/400	8.3e-6
`poisson`$_4$	15	116849	9.0e-6	35.2	154075	2.2e0	3.07	494/40/492	1.4e-1

using Algorithm 3.3.2 with tol $= u_s$, and the `ode45` and `ode15s` functions from MATLAB, with absolute and relative error tolerances (used in a mixed absolute/relative error criterion) both set to u_s. The `ode45` function uses an explicit Runge–Kutta (4,5) formula while `ode15s` uses implicit multistep methods. We called both solvers with time interval specified as `[0 t/2 t]` instead of `[0 t]` in order to stop them returning output at each internal mesh point, which substantially slows the integration. The results in Table 3.4 show the superior efficiency of Algorithm 3.3.2 over `ode45`. The `ode15s` function performs variably, being extremely slow for the `orani678` problem, but faster than Algorithm 3.3.2 for the `poisson` problem with $\alpha = 1$, in this case being helped by the fact that A is highly sparse and structured, so that the linear system solves it requires are relatively inexpensive. However, both ODE solvers fail to produce the desired accuracy for the `poisson` problem with $\alpha = 4$.

Experiment 8. Our final experiment illustrates the application of Theorem 3.2.1 in order to compute the exponential integrator approximation (3.4) via (3.11). We take for $A \in \mathbb{R}^{400\times 400}$ the symmetric matrix `-gallery('poisson',20)` and random vectors u_k, $k = 0\colon p$, with elements from the normal (0,1) distribution, where $p = 5\colon 5\colon 20$. The matrix $W \in \mathbb{R}^{400\times p}$ has columns $W(:, p-k+1) = u_k$, $k = 1\colon p$. For each p, we compute $\widehat{u}(t)$ at each $t = 1\colon 0.5\colon 10$ by Algorithm 3.5.2 (with $t_0 = 1$, $t_q = 10$, and $q = 18$) and by `phipm`, which has the ability to compute $\widehat{u}(t)$ via the expression (3.4). For computing errors, we regard as the "exact" solution the vector obtained by using `expm` to compute the exponential on the right-hand side of (3.11). We set the tolerance to u_d for both algorithms. Figure 3.6 plots the results. The total number of matrix–vector products is 1097 for Algorithm 3.5.2 and 3749 for `phipm`.

The relative error produced by Algorithm 3.5.2 is of order u_d for all t and p, whereas the relative error for `phipm` deteriorates with increasing t, the more rapidly so for the larger p, suggesting instability in the recurrences that the code uses. The practical implication is that p, which is the degree of the polynomial approximation in an exponential integrator, may be need to be limited for use with `phipm` but is unrestricted for our algorithm. The run time for this experiment is 0.093 seconds for Algorithm 3.5.2 and 0.62 seconds for `phipm`.

The importance of the normalization by η in (3.11) is seen if we multiply W by 10^6 and repeat the experiment with the default η and then $\eta = 1$. The maximum relative errors for Algorithm 3.5.2 in the two cases are 2.3×10^{-15} and 3.3×10^{-12}.

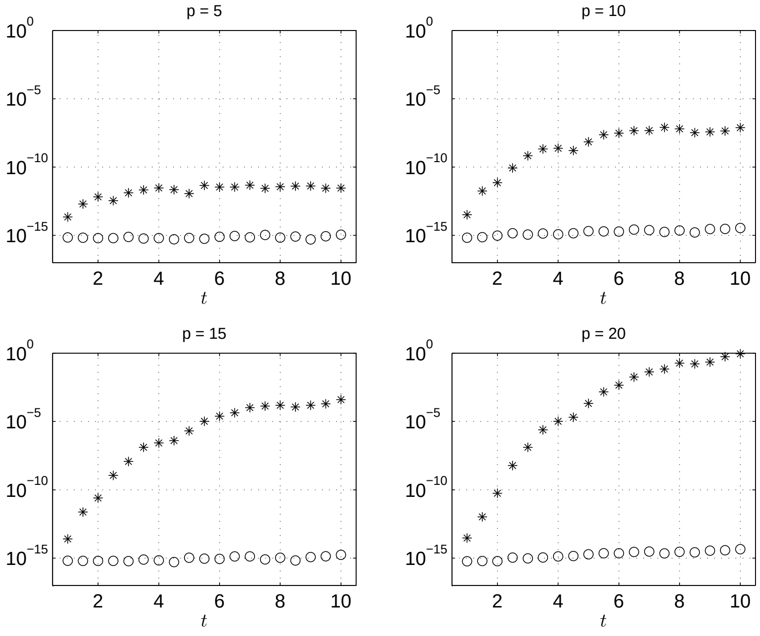

Figure 3.6: Experiment 8: relative errors of computed $\widehat{u}(t)$ in (3.4) from Algorithm 3.5.2 ($\circ$) and `phipm` ($*$) over the interval $[1, 10]$ for $p = 5\colon 5\colon 20$.

Chapter 4

Computing the Fréchet Derivative of the Matrix Exponential, with an Application to Condition Number Estimation

4.1 Introduction

The sensitivity of a Fréchet differentiable matrix function $f : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ (see Section 1.6) to small perturbations is governed by the Fréchet derivative.

It is desirable to be able to evaluate efficiently both $f(A)$ and the Fréchet derivative in order to obtain sensitivity information or to apply an optimization algorithm requiring derivatives. However, while the numerical computation of matrix functions is quite well developed, fewer methods are available for the Fréchet derivative, and the existing methods for $L_f(A, E)$ usually do not fully exploit the fact that $f(A)$ is being computed [31].

The norm of the Fréchet derivative yields a condition number as Theorem 1.6.4 states, so when evaluating $f(A)$ we would like to be able to efficiently estimate $\operatorname{cond}(f, A)$. The computation of $L_f(A, Z)$ at several Z is the key component of this process as established in Section 1.6.

The main aim of the work in this chapter is to develop an efficient algorithm for simultaneously computing e^A and $L_{\exp}(A, E)$ and to use it to construct an algorithm for computing e^A along with an estimate of $\operatorname{cond}(\exp, A)$. The need for such algorithms is demonstrated by a recent paper in econometrics [38] in which the authors state that "One problem we did discover, that has not been accentuated in the literature, is that altering the stability properties of the coefficient matrix through a change in just one parameter can dramatically alter the theoretical and computed matrix exponential." If $A = A(t)$ depends smoothly on a vector $t \in \mathbb{C}^p$ of parameters then the change in e^A induced by small changes θh in t ($\theta \in \mathbb{C}$, $h \in \mathbb{C}^p$) is approximated by $\theta L_{\exp}(A, \sum_{i=1}^p h_i \partial A(t)/\partial t_i)$, since

$$\begin{aligned} f(A(t+\theta h)) &= f\left(A + \theta \sum_{i=1}^p \frac{\partial A(t)}{\partial t_i} h_i + O(\theta^2)\right) \\ &= f(A) + L_f\left(A, \theta \sum_{i=1}^p \frac{\partial A(t)}{\partial t_i} h_i + O(\theta^2)\right) + o(\theta) \\ &= f(A) + \theta L_f\left(A, \sum_{i=1}^p \frac{\partial A(t)}{\partial t_i} h_i\right) + o(\theta). \end{aligned}$$

Thus a single Fréchet derivative evaluation with $h = e_j$ (the jth unit vector) provides the information that the authors of [38] needed about the effect of changing a single parameter t_j.

We begin in Section 4.2 by recalling a useful connection between the Fréchet derivative of a function and the same function evaluated at a certain block triangular matrix. We illustrate how this relation can be used to derive new iterations for computing $L_f(A, E)$ given an iteration for $f(A)$. Then in Section 4.3 we show how to efficiently evaluate the Fréchet derivative when f has a power series expansion, by exploiting a convenient recurrence for the Fréchet derivative of a monomial. In Section 4.4 we show that under reasonable assumptions a matrix polynomial and its Fréchet derivative can both be evaluated at a cost at most three times that of evaluating the polynomial itself. Then in Section 4.5 we show how to evaluate the Fréchet derivative of a rational function and give a framework for evaluating f and its Fréchet derivative via Padé approximants. In Section 4.6 we apply this framework to the scaling and squaring algorithm for e^A, Algorithm 2.3.1. We extend Higham's analysis to show that, modulo rounding errors, the approximations obtained from the new algorithm are $e^{A+\Delta A}$ and $L_{\exp}(A + \Delta A, E + \Delta E)$, with the *same* ΔA in both cases—a genuine backward error result. The computable bounds on $\|\Delta A\|$ and $\|\Delta E\|$ enable us to choose the algorithmic parameters in an optimal fashion. The new algorithm is shown to have significant advantages over existing ones. In Section 4.7 we combine the new algorithm for $L_{\exp}(A, E)$ with an existing matrix 1-norm estimator to develop an algorithm for computing both e^A and an estimate of its condition number, and we show experimentally that the condition estimate can provide a useful guide to the accuracy of the scaling and squaring algorithm. Similarly in Section 4.8, we extend the new scaling and squaring algorithm (Algorithm 2.6.1) that we developed in Chapter 2 to an algorithm for computing both e^A and $L_{\exp}(A, E)$, and then adapt it to an algorithm for simultaneously evaluating e^A and estimating its condition number.

4.2 Fréchet derivative via function of block triangular matrix

The following result shows that the Fréchet derivative appears as the $(1, 2)$ block when f is evaluated at a certain block triangular matrix. Let $\mathcal{D}$ denote an open subset of $\mathbb{R}$ or $\mathbb{C}$.

Theorem 4.2.1 *Let f be $2n-1$ times continuously differentiable on $\mathcal{D}$ and let the spectrum of X lie in $\mathcal{D}$. Then*

$$f\left(\begin{bmatrix} X & E \\ 0 & X \end{bmatrix}\right) = \begin{bmatrix} f(X) & L(X,E) \\ 0 & f(X) \end{bmatrix}. \tag{4.1}$$

Proof. See Mathias [52, Thm. 2.1] or Higham [31, Sec. 3.1]. The result is also proved by Najfeld and Havel [56, Thm. 4.11] under the assumption that f is analytic. □

The significance of Theorem 4.2.1 is that given a smooth enough f and any method for computing $f(A)$, we can compute the Fréchet derivative by applying the method to the $2n \times 2n$ matrix in (4.1). The doubling in size of the problem is unwelcome, but if we exploit the block structure the computational cost can be reduced. Moreover, the theorem can provide a simple means to derive, and prove the convergence of, iterations for computing the Fréchet derivative.

To illustrate the use of the theorem we consider the principal square root function, $f(A) = A^{1/2}$, which for $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on $\mathbb{R}^-$ (the closed negative real axis) is the unique square root X of A whose spectrum lies in the open right half-plane. The Denman–Beavers iteration

$$\begin{aligned} X_{k+1} &= \frac{1}{2}\left(X_k + Y_k^{-1}\right), \qquad X_0 = A, \\ Y_{k+1} &= \frac{1}{2}\left(Y_k + X_k^{-1}\right), \qquad Y_0 = I \end{aligned} \tag{4.2}$$

is a Newton variant that converges quadratically with [31, Sec. 6.3]

$$\lim_{k\to\infty} X_k = A^{1/2}, \qquad \lim_{k\to\infty} Y_k = A^{-1/2}. \tag{4.3}$$

It is easy to show that if we apply the iteration to $\widetilde{A} = \left[\begin{smallmatrix} A & E \\ 0 & A \end{smallmatrix}\right]$ then iterates $\widetilde{X}_k$ and $\widetilde{Y}_k$ are produced for which

$$\widetilde{X}_k = \begin{bmatrix} X_k & F_k \\ 0 & X_k \end{bmatrix}, \qquad \widetilde{Y}_k = \begin{bmatrix} Y_k & G_k \\ 0 & Y_k \end{bmatrix},$$

where

$$\begin{aligned} F_{k+1} &= \frac{1}{2}\left(F_k - Y_k^{-1} G_k Y_k^{-1}\right), \qquad F_0 = E, \\ G_{k+1} &= \frac{1}{2}\left(G_k - X_k^{-1} F_k X_k^{-1}\right), \qquad G_0 = 0. \end{aligned} \tag{4.4}$$

By applying (4.3) and Theorem 4.2.1 to $\widetilde{A}$ we conclude that

$$\lim_{k\to\infty} F_k = L_{x^{1/2}}(A,E), \qquad \lim_{k\to\infty} G_k = L_{x^{-1/2}}(A,E). \tag{4.5}$$

Moreover, scaling strategies for accelerating the convergence of (4.2) [31, Sec. 6.5] yield corresponding strategies for (4.4).

The next result shows quite generally that differentiating a fixed point iteration for a matrix function yields a fixed point iteration for the Fréchet derivative.

Theorem 4.2.2 *Let f and g be $2n-1$ times continuously differentiable on $\mathcal{D}$. Suppose that for any matrix $X \in \mathbb{C}^{n\times n}$ whose spectrum lies in $\mathcal{D}$, g has the fixed point $f(X)$, that is, $f(X) = g(f(X))$. Then for any such X, L_g at $f(X)$ has the fixed point $L_f(X, E)$ for all E.*

Proof. Applying the chain rule to $f(X) \equiv g(f(X))$ gives the relation $L_f(X, E) = L_g(f(X), L_f(X, E))$, which is the result. $\square$

The iteration (4.4) for computing the Fréchet derivative of the square root function is new, and other new iterations for the Fréchet derivative of the matrix square root and related functions can be derived, and their convergence proved, in the same way, or directly by using Theorem 4.2.2. In the case of the Newton iteration for the matrix sign function this approach yields an iteration for the Fréchet derivative proposed by Kenney and Laub [43, Thm. 3.3] (see also [31, Thm. 5.7]) and derived using Theorem 4.2.1 by Mathias [52].

In the rest of this chapter we consider the situation in which the underlying method for computing $f(A)$ is based on direct approximation rather than iteration, and we develop techniques that are more sophisticated than a direct application of Theorem 4.2.1.

4.3 Fréchet derivative via power series

When f has a power series expansion the Fréchet derivative can be expressed as a related series expansion.

Theorem 4.3.1 *Suppose f has the power series expansion $f(x) = \sum_{k=0}^{\infty} a_k x^k$ with radius of convergence r. Then for $A, E \in \mathbb{C}^{n\times n}$ with $\|A\| < r$, the Fréchet derivative*

$$L_f(A, E) = \sum_{k=1}^{\infty} a_k \sum_{j=1}^{k} A^{j-1} E A^{k-j}. \tag{4.6}$$

Proof. See [31, Prob. 3.6]. $\square$

The next theorem gives a recurrence that can be used to evaluate (4.6).

Theorem 4.3.2 *Under the assumptions of Theorem* 4.3.1*,*

$$L_f(A, E) = \sum_{k=1}^{\infty} a_k M_k, \tag{4.7}$$

where $M_k = L_{x^k}(A, E)$ satisfies the recurrence

$$M_k = M_{\ell_1} A^{\ell_2} + A^{\ell_1} M_{\ell_2}, \qquad M_1 = E, \tag{4.8}$$

with $k = \ell_1 + \ell_2$ and ℓ_1 and ℓ_2 positive integers. In particular,

$$M_k = M_{k-1} A + A^{k-1} M_1, \qquad M_1 = E. \tag{4.9}$$

In addition,

$$\|f(A)\| \le \widetilde{f}(\|A\|), \qquad \|L_f(A)\| \le \widetilde{f}'(\|A\|), \tag{4.10}$$

where $\widetilde{f}(x) = \sum_{k=0}^{\infty} |a_k| x^k$.

Proof. Since the power series can be differentiated term-by-term within its radius of convergence, we have

$$L_f(A,E) = \sum_{k=1}^{\infty} a_k M_k, \qquad M_k = L_{x^k}(A,E).$$

One way to develop the recurrence (4.8) is by applying Theorem 4.2.1 to the monomial $x^k = x^{\ell_1+\ell_2}$. A more direct approach is to use the product rule for Fréchet derivatives from Theorem 1.6.5 to obtain

$$M_k = L_{x^k}(A,E) = L_{x^{\ell_1}}(A,E)A^{\ell_2} + A^{\ell_1}L_{x^{\ell_2}}(A,E) = M_{\ell_1}A^{\ell_2} + A^{\ell_1}M_{\ell_2}.$$

Taking $\ell_1 = k-1$ and $\ell_2 = 1$ gives (4.9). It is straightforward to see that $\|f(A)\| \leq \widetilde{f}(\|A\|)$. Taking norms in (4.6) gives

$$\|L_f(A,E)\| \leq \|E\| \sum_{k=1}^{\infty} k|a_k| \|A\|^{k-1} = \|E\|\widetilde{f}'(\|A\|),$$

and maximizing over all nonzero E gives $\|L_f(A)\| \leq \widetilde{f}'(\|A\|)$. □

The recurrence (4.8) will prove very useful in the rest of the chapter.

4.4 Cost analysis for polynomials

Practical methods for approximating $f(A)$ may truncate a Taylor series to a polynomial or use a rational approximation. Both cases lead to the need to evaluate both a polynomial and its Fréchet derivative at the same argument. The question arises "what is the extra cost of computing the Fréchet derivative?" Theorem 4.3.2 does not necessarily answer this question because it only describes one family of recurrences for evaluating the Fréchet derivative. Moreover, the most efficient polynomial evaluation schemes are based on algebraic rearrangements that avoid explicitly forming all the matrix powers. Does an efficient evaluation scheme for a polynomial p also yield an efficient evaluation scheme for L_p?

Consider schemes for evaluating $p_m(X)$, where p_m is a polynomial of degree m and $X \in \mathbb{C}^{n\times n}$, that consist of s steps of the form

$$q_1^{(k)}(X) = q_2^{(k-1)}(X)\,q_3^{(k-1)}(X) + q_4^{(k-1)}(X), \qquad k = 1\colon s, \tag{4.11a}$$

$$\deg q_i^{(k)} < m, \quad i = 1\colon 4,\ k < s, \qquad \deg q_i^{(k)} \geq 1, \quad i = 2\colon 3, \tag{4.11b}$$

where the q_i are polynomials, $q_i^{(k)}$, $i = 2\colon 4$, is a linear combination of $q_1^{(1)}, \ldots, q_1^{(k-1)}$, and $p_m(X) = q_1^{(s)}(X)$. This class contains all schemes of practical interest, which include Horner's method, evaluation by explicit powers, and the Paterson and Stockmeyer method [60] (all of which are described in [31, Sec. 4.2]), as well as more ad hoc schemes such as those described below. We measure the cost of the scheme by the number of matrix multiplications it requires. The next result shows that the overhead of evaluating the Fréchet derivative is at most twice the original cost.

Theorem 4.4.1 *Let p be a polynomial and let π_p denote the cost of evaluating $p(X)$ by any scheme of the form (4.11). Let σ_p denote the extra cost required to compute $L_p(X, E)$ by using the scheme obtained by differentiating the scheme for $p(X)$. Then $\sigma_p \le 2\pi_p$.*

Proof. The proof is by induction on the degree m of the polynomial. For $m = 1$, $p_1(x) = b_0 + b_1 x$ and the only possible scheme is the obvious evaluation $p_1(X) = b_0 I + b_1 X$ with $\pi_1 = 0$. The corresponding Fréchet derivative scheme is $L_{p_1}(X, E) = b_1 E$ and $\sigma_1 = 0$, so the result is trivially true for $m = 1$. Suppose the result is true for all polynomials of degree at most $m - 1$ and consider a polynomial p_m of degree m. By (4.11) the last stage of the scheme can be written $p_m(X) = q_2^{(s-1)}(X)\, q_3^{(s-1)}(X) + q_4^{(s-1)}(X)$, where the polynomials $q_i \equiv q_i^{(s-1)}$, $i = 2\colon 4$ are all of degree less than m. Note that $\pi_{p_m} = \pi_{q_2} + \pi_{q_3} + \pi_{q_4} + 1$ and by the inductive hypothesis, $\sigma_{q_i} \le 2\pi_{q_i}$, $i = 2\colon 4$. Now $L_{p_m}(X, E) = L_{q_2}(X, E) q_3(X) + q_2(X) L_{q_3}(X, E) + L_{q_4}(X, E)$ by the product rule and so

$$\sigma_{p_m} \le \sigma_{q_2} + \sigma_{q_3} + \sigma_{q_4} + 2 \le 2(\pi_{q_2} + \pi_{q_3} + \pi_{q_4} + 1) = 2\pi_{p_m},$$

as required. This proof tacitly assumes that there are no dependencies between the $q_i^{(k)}$ that reduce the cost of evaluating p, for example, $q_2^{(s-1)} = q_3^{(s-1)}$. However, any dependencies equally benefit the L_p evaluation and the result remains valid. □

To illustrate the theorem, consider the polynomial $p(X) = I + X + X^2 + X^3 + X^4 + X^5$. Rewriting it as

$$p(X) = I + X\left(I + X^2 + X^4\right) + X^2 + X^4,$$

we see that $p(X)$ can be evaluated in just three multiplications via $X_2 = X^2$, $X_4 = X_2^2$, and $p(X) = I + X(I + X_2 + X_4) + X_2 + X_4$. Differentiating gives

$$\begin{aligned} L_p(X, E) &= L_{x(1+x^2+x^4)}(X, E) + M_2 + M_4 \\ &= E(I + X_2 + X_4) + X(M_2 + M_4) + M_2 + M_4, \end{aligned}$$

where $M_2 = XE + EX$ and $M_4 = M_2 X_2 + X_2 M_2$ by (4.8). Hence the Fréchet derivative can be evaluated with six additional multiplications, and the total cost is nine multiplications.

4.5 Computational framework

For a number of important functions f, such as the exponential, the logarithm, and the sine and cosine, successful algorithms for $f(A)$ have been built on the use of Padé approximants: a Padé approximant r_m of f of suitable degree m is evaluated at a transformed version of A and the transformation is then undone. Here, $r_m(x) = p_m(x)/q_m(x)$ with p_m and q_m polynomials of degree m such that $f(x) - r_m(x) = O(x^{2m+1})$ [40]. It is natural to make use of this Padé approximant by approximating L_f by the Fréchet derivative L_{r_m} of r_m. The next result shows how to evaluate L_{r_m}.

Lemma 4.5.1 *The Fréchet derivative L_{r_m} of the rational function $r_m(x) = p_m(x)/q_m(x)$ satisfies*

$$q_m(A) L_{r_m}(A, E) = L_{p_m}(A, E) - L_{q_m}(A, E) r_m(A). \tag{4.12}$$

Proof. Applying the Fréchet derivative product rule to $q_m r_m = p_m$ gives

$$L_{p_m}(A,E) = L_{q_m r_m}(A,E) = L_{q_m}(A,E) r_m(A) + q_m(A) L_{r_m}(A,E),$$

which rearranges to the result. □

We can now state a general framework for simultaneously approximating $f(A)$ and $L_f(A,E)$ in a way that reuses matrix multiplications from the approximation of f in the approximation of L_f.

1. Choose a suitable Padé degree m and transformation function g and set $A \leftarrow g(A)$.
2. Devise efficient schemes for evaluating $p_m(A)$ and $q_m(A)$.
3. Fréchet differentiate the schemes in the previous step to obtain schemes for evaluating $L_{p_m}(A,E)$ and $L_{q_m}(A,E)$. Use the recurrences (4.8) and (4.9) as necessary.
4. Solve $q_m(A) r_m(A) = p_m(A)$ for $r_m(A)$.
5. Solve $q_m(A) L_{r_m}(A,E) = L_{p_m}(A,E) - L_{q_m}(A,E) r_m(A)$ for $L_{r_m}(A,E)$.
6. Apply the appropriate transformations to $r_m(A)$ and $L_{r_m}(A,E)$ that undo the effect of the initial transformation on A.

In view of Theorem 4.4.1, the cost of this procedure is at most $(3\pi_m + 1)M + 2D$, where $\pi_m M$ is the cost of evaluating both $p_m(A)$ and $q_m(A)$, and M and D denote a matrix multiplication and the solution of a matrix equation, respectively.

If we are adding the capability to approximate the Fréchet derivative to an existing Padé-based method for $f(A)$ then our attention will focus on step 1, where we must reconsider the choice of m and transformation to ensure that both f and L_f are approximated to sufficient accuracy.

In the next sections we apply this framework to the matrix exponential.

4.6 Scaling and squaring algorithm for the exponential and its Fréchet derivative (I)

Our aim here is to adapt Algorithm 2.3.1 that we review in Section 2.3 to compute $L_{\exp}(A,E)$ along with e^A. A recurrence for the Fréchet derivative of the exponential can be obtained by differentiating (2.1). Note first that differentiating the identity $e^A = (e^{A/2})^2$ using the chain rule of Theorem 1.6.5 along with $L_{x^2}(A,E) = AE + EA$ gives the relation

$$\begin{aligned} L_{\exp}(A,E) &= L_{x^2}\big(e^{A/2}, L_{\exp}(A/2,E/2)\big) \\ &= e^{A/2} L_{\exp}(A/2,E/2) + L_{\exp}(A/2,E/2) e^{A/2}. \end{aligned} \tag{4.13}$$

Repeated use of this relation leads to the recurrence

$$\begin{aligned} \widetilde{L}_s &= L_{\exp}(2^{-s}A, 2^{-s}E), \\ \widetilde{L}_{i-1} &= e^{2^{-i}A}\widetilde{L}_i + \widetilde{L}_i e^{2^{-i}A}, \qquad i = s\colon -1\colon 1 \end{aligned} \tag{4.14}$$

Table 4.1: Maximal values ℓ_m of $\|2^{-s}A\|$ such that the backward error bound (4.19) does not exceed $u = 2^{-53}$, along with maximal values θ_m such that a bound for $\|\Delta A\|/\|A\|$ does not exceed u.

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
θ_m	3.65e-8	5.32e-4	1.50e-2	8.54e-2	2.54e-1	5.41e-1	9.50e-1	1.47e0	2.10e0	2.81e0	3.60e0	4.46e0	5.37e0	6.33e0	7.34e0	8.37e0	9.44e0	1.05e1	1.17e1	1.28e1
ℓ_m	2.11e-8	3.56e-4	1.08e-2	6.49e-2	2.00e-1	4.37e-1	7.83e-1	1.23e0	1.78e0	2.42e0	3.13e0	3.90e0	4.74e0	5.63e0	6.56e0	7.52e0	8.53e0	9.56e0	1.06e1	1.17e1

for $\widetilde{L}_0 = L_{\exp}(A, E)$. Our numerical method replaces $L_{\exp}$ by L_{r_m} and $e^{2^{-i}A}$ by $r_m(2^{-s}A)^{2^{s-i}}$, producing approximations L_i to $\widetilde{L}_i$:

$$\begin{aligned} X_s &= r_m(2^{-s}A),\\ L_s &= L_{r_m}(2^{-s}A, 2^{-s}E),\\ \left.\begin{aligned} L_{i-1} &= X_i L_i + L_i X_i\\ X_{i-1} &= X_i^2 \end{aligned}\right\} &\quad i = s\colon -1\colon 1. \end{aligned} \tag{4.15}$$

The key question is what can be said about the accuracy or stability of L_0 relative to that of the approximation $X_0 = (r_m(2^{-s}A))^{2^s}$ to e^A. To answer this question we recall the key part of the error analysis from Section 2.3. Select $s \geq 0$ such that $2^{-s}A \in \Omega$ and hence

$$r_m(2^{-s}A) = e^{2^{-s}A + h_{2m+1}(2^{-s}A)} \tag{4.16}$$

Differentiating (4.16) gives, using the chain rule,

$$\begin{aligned} L_s &= L_{r_m}(2^{-s}A, 2^{-s}E)\\ &= L_{\exp}\big(2^{-s}A + h_{2m+1}(2^{-s}A), 2^{-s}E + L_{h_{2m+1}}(2^{-s}A, 2^{-s}E)\big). \end{aligned} \tag{4.17}$$

From (4.15), (4.16), and (4.17),

$$\begin{aligned} L_{s-1} &= r_m(2^{-s}A)L_s + L_s r_m(2^{-s}A)\\ &= e^{2^{-s}A+h_{2m+1}(2^{-s}A)}\, L_{\exp}\big(2^{-s}A + h_{2m+1}(2^{-s}A), 2^{-s}E + L_{h_{2m+1}}(2^{-s}A, 2^{-s}E)\big)\\ &\quad + L_{\exp}\big(2^{-s}A + h_{2m+1}(2^{-s}A), 2^{-s}E + L_{g_m}(2^{-s}A, 2^{-s}E)\big)\, e^{2^{-s}A+h_{2m+1}(2^{-s}A)}\\ &= L_{\exp}\big(2^{-(s-1)}A + 2h_{2m+1}(2^{-s}A), 2^{-(s-1)}E + L_{h_{2m+1}}(2^{-s}A, 2^{-(s-1)}E)\big), \end{aligned}$$

where we have used (4.13) and the fact that L is linear in its second argument. Continuing this argument inductively, and using

$$X_i = X_s^{2^{s-i}} = \big(e^{2^{-s}A+h_{2m+1}(2^{-s}A)}\big)^{2^{s-i}} = e^{2^{-i}A+2^{s-i}h_{2m+1}(2^{-s}A)},$$

we obtain the following result.

Theorem 4.6.1 *If* $2^{-s}A \in \Omega_m$ *in* (2.13) *then* L_0 *from* (4.15) *satisfies*

$$L_0 = L_{\exp}\big(A + 2^s h_{2m+1}(2^{-s}A), E + L_{h_{2m+1}}(2^{-s}A, E)\big). \qquad \square \tag{4.18}$$

Theorem 4.6.1 is a backward error result: it says that L_0 is the exact Fréchet derivative for the exponential of a perturbed matrix in a perturbed direction. We emphasize that the backward error is with respect to the effect of truncation errors in the Padé approximation, not to rounding errors, which for the moment are ignored.

We have $X_0 = e^{A+\Delta A}$ and $L_0 = L_{\exp}(A + \Delta A, E + \Delta E)$ with the *same* $\Delta A = 2^s h_{2m+1}(2^{-s}A)$. We already know from the analysis in Section 2.3 how to choose s and m to keep ΔA acceptably small. It remains to investigate the norm of $\Delta E = L_{h_{2m+1}}(2^{-s}A, E)$.

Recall that $\widetilde{h}_{2m+1}(x)$ is the function used in the bound (2.16), which is the power series resulting from replacing the coefficients of the power series expansion of the function $h_{2m+1}(x) = \log(e^{-x}r_m(x))$ by their absolute values. Using the second bound in (4.10) we have

$$\frac{\|\Delta E\|}{\|E\|} = \frac{\|L_{h_{2m+1}}(2^{-s}A, E)\|}{\|E\|} \le \|L_{h_{2m+1}}(2^{-s}A)\| \le \widetilde{h}'_{2m+1}(\theta), \tag{4.19}$$

where $\theta = \|2^{-s}A\|$. Define $\ell_m = \max\{\,\theta : \widetilde{h}'_{2m+1}(\theta) \le u\,\}$, where $u = 2^{-53} \approx 1.1\times 10^{-16}$ is the unit roundoff for IEEE double precision arithmetic. Using Symbolic Math Toolbox we evaluated ℓ_m, $m = 1\colon 20$, by summing the first 150 terms of the series symbolically in 250 decimal digit arithmetic. Table 4.1 shows these values along with analogous values θ_m calculated in [30], which are the maximal values of θ obtained from (2.17). In every case $\ell_m < \theta_m$, which is not surprising given that we are approximating L_{r_m} by an approximation chosen for computational convenience rather than its approximation properties, but the ratio θ_m/ℓ_m is close to 1. For each m, if $\theta \le \ell_m$ then we are assured that

$$X_0 = e^{A+\Delta A}, \quad L_0 = L_{\exp}(A + \Delta A, E + \Delta E), \quad \|\Delta A\| \le u\|A\|, \quad \|\Delta E\| \le u\|E\|;$$

in other words, perfect backward stability is guaranteed for such θ.

In order to develop an algorithm we now need to look at the cost of evaluating $r_m = p_m/q_m$ and L_{r_m}, where r_m is the $[m/m]$ Padé approximant to e^x. Higham [30] shows how to efficiently evaluate $p_m(A)$ and $q_m(A)$ by using the schemes (2.18) and (2.19); the number of matrix multiplications, π_m, required to compute $p_m(A)$ and $q_m(A)$ is given in [30, Table 2.2]. As Theorem 4.4.1 suggests, the Fréchet derivatives L_{p_m} and L_{q_m} can be calculated at an extra cost of $2\pi_m$ multiplications by differentiating the schemes for p_m and q_m. We now give the details.

Recall the scheme (2.18) for $m = 3, 5, 7, 9$

$$p_m(A) = A \sum_{k=0}^{(m-1)/2} b_{2k+1}A^{2k} + \sum_{k=0}^{(m-1)/2} b_{2k}A^{2k} =: u_m(A) + v_m(A).$$

It follows that $q_m(A) = -u_m(A) + v_m(A)$ since $q_m(A) = p_m(-A)$, and hence

$$L_{p_m} = L_{u_m} + L_{v_m}, \qquad L_{q_m} = -L_{u_m} + L_{v_m}.$$

We obtain $L_{u_m}(A, E)$ and $L_{v_m}(A, E)$ by differentiating u_m and v_m, respectively:

$$L_{u_m}(A, E) = A \sum_{k=1}^{(m-1)/2} b_{2k+1}M_{2k} + E \sum_{k=0}^{(m-1)/2} b_{2k+1}A^{2k} \tag{4.20}$$

$$L_{v_m}(A, E) = \sum_{k=1}^{(m-1)/2} b_{2k}M_{2k}. \tag{4.21}$$

The $M_k = L_{x^k}(A, E)$ are computed using (4.8).

For $m = 13$, we write the scheme (2.19) for $p_{13}(A) = u_{13}(A) + v_{13}(A)$ in the form

$$\begin{gathered} u_{13}(A) = Aw(A), \quad w(A) = A^6 w_1(A) + w_2(A), \quad v_{13}(A) = A^6 z_1(A) + z_2(A), \\ w_1(A) = b_{13}A^6 + b_{11}A^4 + b_9A^2, \quad w_2(A) = b_7A^6 + b_5A^4 + b_3A^2 + b_1I, \\ z_1(A) = b_{12}A^6 + b_{10}A^4 + b_8A^2, \quad z_2(A) = b_6A^6 + b_4x^4 + b_2A^2 + b_0I. \end{gathered}$$

Differentiating these polynomials yields

$$\begin{aligned} L_{u_{13}}(A, E) &= AL_w(A, E) + Ew(A), \\ L_{v_{13}}(A, E) &= A^6 L_{z_1}(A, E) + M_6 z_1(A) + L_{z_2}(A, E), \end{aligned}$$

where

$$\begin{aligned} L_w(A, E) &= A^6 L_{w_1}(A, E) + M_6 w_1(A) + L_{w_2}(A, E), \\ L_{w_1}(A, E) &= b_{13}M_6 + b_{11}M_4 + b_9M_2, \\ L_{w_2}(A, E) &= b_7M_6 + b_5M_4 + b_3M_2, \\ L_{z_1}(A, E) &= b_{12}M_6 + b_{10}M_4 + b_8M_2, \\ L_{z_2}(A, E) &= b_6M_6 + b_4M_4 + b_2M_2. \end{aligned}$$

Then $L_{p_{13}} = L_{u_{13}} + L_{v_{13}}$ and $L_{q_{13}} = -L_{u_{13}} + L_{v_{13}}$. We finally solve for $r_m(A)$ and $L_{r_m}(A, E)$ the equations

$$\begin{aligned} (-u_m + v_m)(A) r_m(A) &= (u_m + v_m)(A), \\ (-u_m + v_m)(A) L_{r_m}(A, E) &= (L_{u_m} + L_{v_m})(A, E) \\ &\quad + (L_{u_m} - L_{v_m})(A, E) r_m(A). \end{aligned} \tag{4.22}$$

We are now in a position to choose the degree m and the scaling parameter s. Table 4.2 reports the total number of matrix multiplications, $\omega_m = 3\pi_m + 1$, necessary to evaluate r_m and L_{r_m} for a range of m, based on [30, Table 2.2] and the observations above. In evaluating the overall cost we need to take into account the squaring phase. If $\|A\| > \ell_m$ then in order to use the $[m/m]$ Padé approximant we must scale A by 2^{-s} so that $\|2^{-s}A\| \le \ell_m$, that is, we need $s = \lceil \log_2(\|A\|/\ell_m) \rceil$. From the recurrence (4.15), we see that $3s$ matrix multiplications are added to the cost of evaluating r_m and L_{r_m}. Thus the overall cost in matrix multiplications is

$$\omega_m + 3s = 3\pi_m + 1 + 3\max(\lceil \log_2 \|A\| - \log_2 \ell_m \rceil, 0). \tag{4.23}$$

To minimize the cost we therefore choose m to minimize the quantity

$$C_m = \pi_m - \log_2 \ell_m, \tag{4.24}$$

where we have dropped the constant terms and factors in (4.23). Table 4.2 reports the C_m values. The table shows that $m = 13$ is the optimal choice, just as it is for the scaling and squaring method for the exponential itself [30]. The ω_m values also show that only $m = 1, 2, 3, 5, 7, 9$ need be considered if $\|A\| < \ell_{13}$. As in [30] we rule out $m = 1$ and $m = 2$ on the grounds of possible loss of significant figures in floating point arithmetic.

Table 4.2: Number of matrix multiplications, ω_m, required to evaluate $r_m(A)$ and $L_{r_m}(A, E)$, and measure of overall cost C_m in (4.24).

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ω_m	1	4	7	10	10	13	13	16	16	19	19	19	19	22	22	22	22	25	25	25
C_m	25.5	12.5	8.5	6.9	5.3	5.2	4.4	4.7	4.2	4.7	4.4	4.0	3.8	4.5	4.3	4.1	3.9	4.7	4.6	4.5

It remains to check that the evaluation of L_{p_m}, L_{q_m}, and L_{r_m} is done accurately in floating point arithmetic. The latter matrix is evaluated from (4.22), which involves solving a matrix equation with coefficient matrix $q_m(A)$, just as in the evaluation of r_m, and the analysis from [30] guarantees that $q_m(A)$ is well conditioned for the scaled A. It can be shown that for our schemes for evaluating L_{p_m} we have

$$\|L_{p_m}(A, E) - fl(L_{p_m}(A, E))\|_1 \leq \widetilde{\gamma}_{n^2} p'_m(\|A\|_1)\|E\|_1 \approx \widetilde{\gamma}_{n^2} e^{\|A\|_1/2}\|E\|_1,$$

where we have used the facts that p_m has positive coefficients and $p_m(x) \approx e^{x/2}$. Here, $\widetilde{\gamma}_k = cku/(1 - cku)$, where c denotes a small integer constant. At least in an absolute sense, this bound is acceptable for $\|A\| \leq \ell_{13}$. An entirely analogous bound can be obtained for L_{q_m}, since $q_m(x) = p_m(-x)$.

The following code fragment exploits the evaluation schemes for Padé approximants and their Fréchet derivatives as described above. We will use it several of our later algorithms.

Code Fragment 4.6.2 (outputs = XL($m, A,$ inputs)) *This code evaluates the Padé approximant for the matrix exponential, $X := r_m(A)$, or its Fréchet derivative, $L := L_{r_m}(A, E)$ or both depending on the given inputs. The potential inputs are $\{A_{2k}\}_{k=1}^{d}$, where $d = (m-1)/2$ if $m \in \{3, 5, 7, 9\}$ or $d = 3$ if $m = 13$, and the matrices E, W_m, W (for $m = 13$), X, and the LU factorization of $-U + V$, denoted "LU fact.".*

```
if $m \in \{3, 5, 7, 9\}$
   for $k = 0 : (m-1)/2$
       if $A_{2k} \notin$ inputs, compute $A_{2k} := A^{2k}$, end
   end
   if $W_m \notin$ inputs, $W_m = \sum_{k=0}^{(m-1)/2} b_{2k+1} A_{2k}$, end
   $U = AW_m$
   $V = \sum_{k=0}^{(m-1)/2} b_{2k} A_{2k}$
   if $E \in$ inputs
      Evaluate $M_{2k} := L_{x^{2k}}(A, E)$, $k = 1 : (m-1)/2$, using (4.8).
      $L_u = A \sum_{k=1}^{(m-1)/2} b_{2k+1} M_{2k} + EW_m$
      $L_v = \sum_{k=1}^{(m-1)/2} b_{2k} M_{2k}$
   end
else if $m = 13$
```

```
14      for $k = 1{:}3$
15          if $A_{2k} \notin$ **inputs**, compute $A_{2k}{:} = A^{2k}$, end
16      end
17      $W_1 = b_{13}A_6 + b_{11}A_4 + b_9A_2$
18      $W_2 = b_7A_6 + b_5A_4 + b_3A_2 + b_1I$
19      $Z_1 = b_{12}A_6 + b_{10}A_4 + b_8A_2$
20      $Z_2 = b_6A_6 + b_4A_4 + b_2A_2 + b_0I$
21      if $W \notin$ **inputs**, $W = A_6W_1 + W_2$, end
22      $U = AW$
23      $V = A_6Z_1 + Z_2$
24      if $E \in$ **inputs**
25          $M_2 = AE + EA$, $M_4 = A_2M_2 + M_2A_2$, $M_6 = A_4M_2 + M_4A_2$
26          $L_{w_1} = b_{13}M_6 + b_{11}M_4 + b_9M_2$
27          $L_{w_2} = b_7M_6 + b_5M_4 + b_3M_2$
28          $L_{z_1} = b_{12}M_6 + b_{10}M_4 + b_8M_2$
29          $L_{z_2} = b_6M_6 + b_4M_4 + b_2M_2$
30          $L_w = A_6L_{w_1} + M_6W_1 + L_{w_2}$
31          $L_u = AL_w + EW$
32          $L_v = A_6L_{z_1} + M_6Z_1 + L_{z_2}$
33      end
34  end
35  if “LU fact.” $\notin$ **inputs**
36      Compute LU factorization of $-U + V$.
37  end
38  if $X \notin$ **inputs**
39      Solve $(-U + V)X = U + V$ for $X$.
40  if $E \in$ **inputs**
41      Solve $(-U + V)L = L_u + L_v + (L_u - L_v)X$ for $L$.
42  end
```

This code presents the central part of the forthcoming algorithms, so it is very crucial to understand how to exploit it. We illustrate this by the following example. For evaluating $X = \mathbf{XL}(7, A, A_2)$, the code only executes lines 1–7 and lines 35–39, but avoids recompting A_2 in line 3 since it is already given. The lines 9–11 and line 41 will not be executed since they require E , which is not given.

The linear systems at lines 39 and 41 have the same coefficient matrix, so an LU factorization can be computed once and reused.

We now state the complete algorithm that intertwines the computation of the matrix exponential with its Fréchet derivative.

Algorithm 4.6.3 *Given $A, E \in \mathbb{C}^{n\times n}$ this algorithm computes $X = e^A$ and $L = L_{\exp}(A, E)$ by a scaling and squaring algorithm. It uses the parameters ℓ_m listed in Table 4.1. The algorithm is intended for IEEE double precision arithmetic.*

```
for $m = [3\ 5\ 7\ 9]$
    if $\|A\|_1 \le \ell_m$
        $[X, L] = \mathbf{XL}(m, A, E)$ % Code Fragment 4.6.2.
        quit
        end
    end
    $s = \lceil \log_2(\|A\|_1/\ell_{13}) \rceil$, the minimal integer such that $\|2^{-s}A\|_1 \le \ell_{13}$.
    $A \leftarrow 2^{-s}A$ and $E \leftarrow 2^{-s}E$
    $[X, L] = \mathbf{XL}(13, A, E)$ % Code Fragment 4.6.2.
    for $k = 1\colon s$
        $L \leftarrow XL + LX$
        $X \leftarrow X^2$
    end
```

Cost: $(\omega_m + 3s)M + 2D$, where m is the degree of Padé approximant used and ω_m is given in Table 4.2.

Since $L_{\exp}(A, \alpha E) = \alpha L_{\exp}(A, E)$, an algorithm for computing $L_{\exp}(A, E)$ should not be influenced in any significant way by $\|E\|$, and this is the case for Algorithm 4.6.3. Najfeld and Havel [56] propose computing $L_{\exp}(A, E)$ using their version of the scaling and squaring method for the exponential in conjunction with (4.1). With this approach E affects the amount of scaling, and overscaling results when $\|E\| \gg \|A\|$, while how to scale E to produce the most accurate result is unclear.

To assess the cost of Algorithm 4.6.3 we compare it with Algorithm 2.3.1 and with a "Kronecker–Sylvester scaling and squaring algorithm" of Kenney and Laub [44], which is based on a Kronecker sum representation of the Fréchet derivative. In the form detailed in [31, Sec. 10.6.2], this latter algorithm scales to obtain $\|2^{-t}A\| \le 1$, evaluates the [8/8] Padé approximant to $\tanh(x)/x$ at the scaled Kronecker sum, and then uses the recurrence (4.15) or the variant (4.14) that explicitly computes $X_i = e^{2^{-i}A}$ in each step. It requires one matrix exponential, $(17 + 3t)M$, and the solution of 8 Sylvester equations if (4.15) is used, or s matrix exponentials, $(18+2t)M$, and the same number of Sylvester equation solutions if (4.14) is used.

To compare the algorithms, assume that the Padé degree $m = 13$ is used in Algorithms 2.3.1 and 4.6.3. Then Algorithm 4.6.3 costs $(19 + 3s)M + 2D$ and Algorithm 2.3.1 costs $(6 + s)M + D$. Two conclusions can be drawn. First, Algorithm 4.6.3 costs about three times as much as just computing e^A. Second, since the cost of solving a Sylvester equation is about $60n^3$ flops, which is the cost of 30 matrix multiplications, the Kronecker–Sylvester algorithm is an order of magnitude more expensive than Algorithm 4.6.3. To be more specific, consider the case where $\|A\| = 9$, so that $s = 1$ in Algorithms 2.3.1 and 4.6.3 and $t = 4$, and ignore the cost of computing the matrix exponential in the less expensive "squaring" variant of the Kronecker–Sylvester algorithm. Then the operation counts in flops are approximately $48n^3$ for Algorithm 4.6.3 (e^A and $L_{\exp}(A, E)$), $16n^3$ for Algorithm 2.3.1 (e^A only), and $538n^3$ for the Kronecker–Sylvester algorithm ($L_{\exp}(A, E)$ only). A further drawback of the Kronecker–Sylvester algorithm is that it requires complex arithmetic, so the effective flop count is even higher.

Other algorithms for $L_{\exp}(A, E)$ are those of Kenney and Laub [42] and Mathias [51] (see also [31, Sec. 10.6.1]), which apply quadrature to an integral representation of the Fréchet derivative. These algorithms are intended only for low accuracy approximations and do not lend themselves to combination with Algorithm 2.3.1.

We describe a numerical experiment that tests the accuracy of Algorithm 4.6.3. We took the 155 test matrices we used in Experiment 1 of Section 3.6, which are a combination of all four sets of tests matrices described in Experiment 2.6. We evaluated the normwise relative errors of the computed Fréchet derivatives $L_{\exp}(A, E)$, using a different E, generated as `randn(n)`, for each A. The "exact" Fréchet derivative is obtained using (4.1) with the exponential evaluated at 100 digit precision via Symbolic Math Toolbox. Figure 4.1 displays the Frobenius norm relative errors for Algorithm 4.6.3 and for the Kronecker–Sylvester algorithm in both "squaring" and "exponential" variants. Also shown is a solid line representing a finite difference approximation to $\operatorname{cond}(L_{\exp}, A)u$, where $\operatorname{cond}(L_{\exp}, A)$ is a condition number defined in terms of the Jacobian of the map L regarded as a function of A and E (we use (1.6) with a small, random E); this line indicates the accuracy we would expect from a forward stable algorithm for computing the Fréchet derivative. Figure 4.1 shows that all the methods are performing in a reasonably forward stable manner but does not clearly reveal differences between the methods.

Figure 4.2 plots the same data as a performance profile: for a given α the corresponding point on each curve indicates the fraction p of problems on which the method had error at most a factor α times that of the smallest error over all three methods. The results show clear superiority of Algorithm 4.6.3 over the Kronecker–Sylvester algorithm in terms of accuracy, for both variants of the latter algorithm. Since Algorithm 4.6.3 is also by far the more efficient, as explained above, it is clearly the preferred method.

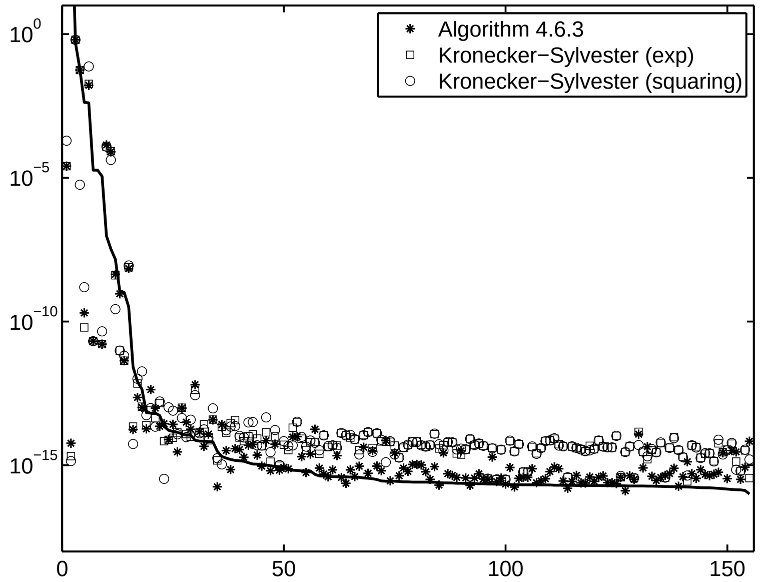

Figure 4.1: Normwise relative errors in Fréchet derivatives $L_{\exp}(A, E)$ computed by Algorithm 4.6.3 and two variants of the Kronecker–Sylvester algorithm for 155 matrices A with a different random E for each A, along with estimate of $\operatorname{cond}(L_{\exp}, A)u$ (solid line).

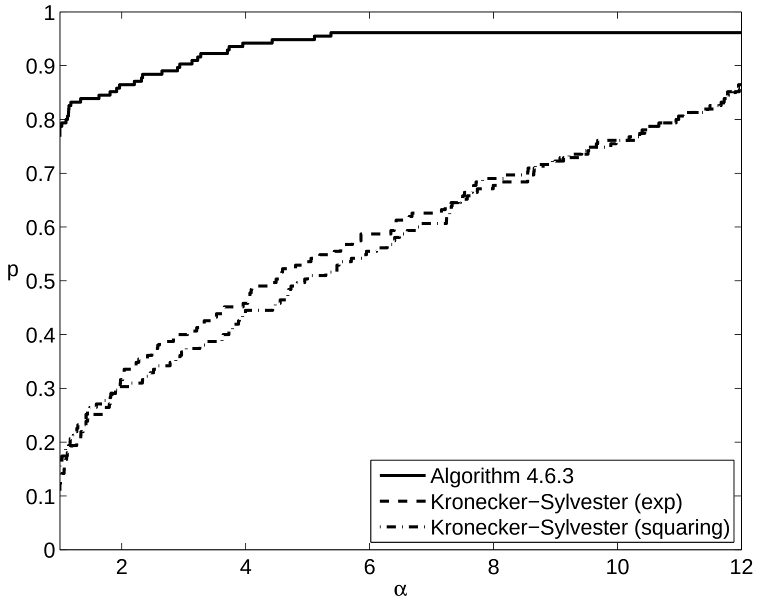

Figure 4.2: Same data as in Figure 4.1 presented as a performance profile.

4.7 Condition number estimation

We now turn our attention to estimating the condition number of the matrix exponential, which from Theorem 1.6.4 is

$$\kappa_{\exp}(A) = \frac{\|L_{\exp}(A)\| \, \|A\|}{\|e^A\|}.$$

Algorithm 4.6.3 can compute $L_{\exp}(A, E)$ for any direction E. This is an essential component to compute or estimate $\|L_{\exp}(A)\|$ by Algorithms 1.6.8, 1.6.9, and 1.6.12, using the fact that $L^{\star}_{\exp}(A, W) \equiv L_{\exp}(A^*, W)$ (see Subsec. 1.6.2).

Our interest is in how to combine Algorithms 4.6.3 and 1.6.12 in the most efficient manner. We need to evaluate $L_{\exp}(A, E)$ and $L_{\exp}(A^*, E)$ for a fixed A and several different E, without knowing all the E at the start of the computation. To do so we will store matrices accrued during the initial computation of e^A and reuse them in the Fréchet derivative evaluations. This of course assumes the availability of extra storage, but in modern computing environments ample storage is usually available.

In view of the evaluation schemes (2.18)–(4.21) and (4.22), for $m \in \{3, 5, 7, 9\}$ we need to store A^{2k}, $k = 1\colon d_{(m-1)/2}$, where $d = [0\ 1\ 2\ 2]$, along with $W_m(A) = \sum_{k=0}^{(m-1)/2} b_{2k+1} A^{2k}$, $r_m(A)$, and the LU factorization of $(-u_m + v_m)(A)$. For $m = 13$, the matrix A needs to be scaled, to $B = A/2^s$. According to the scheme used in Algorithm 4.6.3 we need to store B^{2k}, $k = 1\colon 3$, $W \equiv w(B)$, the LU factorization of $(-u_m + v_m)(B)$, and $r_m(B)^{2^i}$, $i = 0\colon s-1$. Table 4.3 summarizes the matrices that need to be stored for each m.

The following algorithm computes the matrix exponential and estimates its condition number. Since the condition number is not needed to high accuracy we use the parameters θ_m in Table 4.1 (designed for e^A) instead of ℓ_m (designed for $L(A, E)$). The bound in (4.19) for the Fréchet derivative backward error $\|\Delta E\|/\|E\|$ does not

Table 4.3: Matrices that must be computed and stored during the initial e^A evaluation, to be reused during the Fréchet derivative evaluations. "LU fact" stands for LU factorization of $-u_m + v_m$, and $B = A/2^s$.

m						
3	$r_3(A)$				LU fact.	$W_3(A)$
5	$r_5(A)$	A^2			LU fact.	$W_5(A)$
7	$r_7(A)$	A^2	A^4		LU fact.	$W_7(A)$
9	$r_9(A)$	A^2	A^4		LU fact.	$W_9(A)$
13	$r_{13}(B)^{2^i}$, $i = 0\colon s-1$	B^2	B^4	B^6	LU fact.	W

exceed $28u$ for $m \leq 13$ when we use the θ_m, so the loss in backward stability for the Fréchet derivative evaluation is negligible. If the condition estimate is omitted, the algorithm reduces to Algorithm 2.3.1. The algorithm exploits the relation $L_f(A^*, E) = L_f(A, E^*)^*$, which holds for any f with a power series expansion with real coefficients, by (4.6).

Algorithm 4.7.1 *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes $X = e^A$ by the scaling and squaring method* (*Algorithm* 2.3.1) *and an estimate $\gamma \approx \kappa_{\exp}(A)$ using the block* 1*-norm estimator* (*Algorithm* 1.6.12)*. It uses the values θ_m listed in Table* 4.1*. The algorithm is intended for IEEE double precision arithmetic.*

```
 1  $\alpha = \|A\|_1$
 2  for $m = [3\ 5\ 7\ 9]$
 3      if $\alpha \leq \theta_m$
 4          $[X, \{A_{2k}\}_{k=1}^{(m-1)/2}, W_m$, "LU fact."$] = \mathbf{XL}(m, A)$ % Code Fragment 4.6.2.
            % The output matrices need to be stored (see Table 4.3).
 5          Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(A)\|_1$.
            ...... To compute $L = L_{\exp}(A, E)$ for a given $E$:
 6                  $L = \mathbf{XL}(m, A, E, X, \{A_{2k}\}_{k=1}^{(m-1)/2}, W_m$, "LU fact."$)$
 7          ...... To compute $L = L^{\star}_{\exp}(A, E)$ for a given $E$:
 8                  $L = \mathbf{XL}(m, A, E^*, X, \{A_{2k}\}_{k=1}^{(m-1)/2}, W_m$, "LU fact."$)$
 9                  $L = L^*$
10      goto line 27
11  end
12  % Use degree $m = 13$.
13  $s = \lceil \log_2(\alpha/\theta_{13}) \rceil$, the minimal integer such that $2^{-s}\alpha \leq \theta_{13}$.
14  $A \leftarrow 2^{-s}A$
15  $[X_s, \{A_{2k}\}_{k=1}^{3}, W$, "LU fact."$] = \mathbf{XL}(13, A)$
16  for $i = s\colon -1\colon 1$
17      $X_{i-1} = X_i^2$
18  end
    % $\{X_i\}_{i=1}^{s}$ and the output matrices need to be stored (see Table 4.3).
19  $X = X_0$
20  Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(\widetilde{A})\|_1$,
    where $\widetilde{A}$ denotes the original input matrix $A$.
```

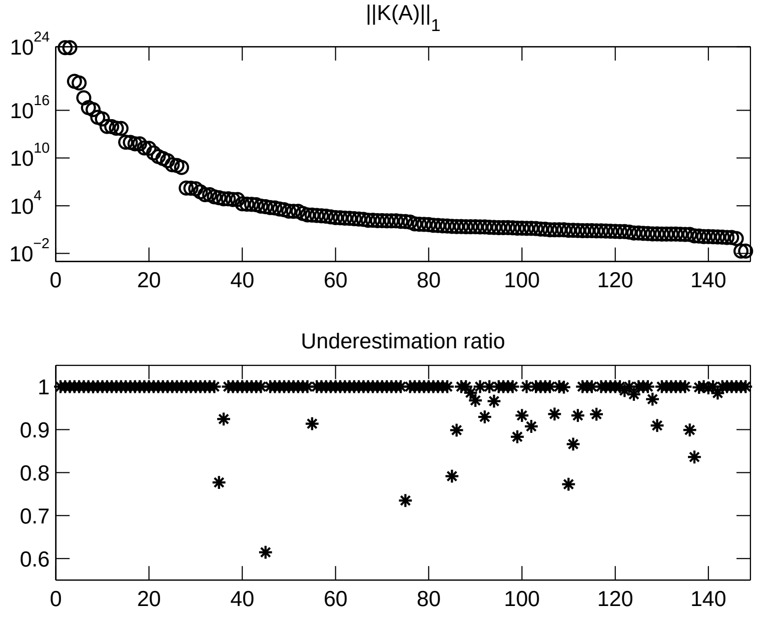

Figure 4.3: $\|K(A)\|_1$ and underestimation ratio $\eta/\|K(A)\|_1$, where η is the estimate of $\|K(A)\|_1$ produced by Algorithm 4.7.1.

```
   ...... To compute $L_{\exp}(\widetilde{A}, E)$ for a given $E$:
21          $E \leftarrow 2^{-s}E$
22          $L = \mathbf{XL}(13, A, E, X_s, \{A_{2k}\}_{k=1}^{3}, W,$ "LU fact.")
23          for $i = s\colon -1\colon 1$
24              $L \leftarrow X_i L + L X_i$
25          end
   ...... To compute $L^{\star}_{\exp}(\widetilde{A}, E)$ for a given $E$:
26                Execute lines 21–25 with $E$ replaced by its conjugate
                  transpose and take the conjugate transpose of the result.
27 $\gamma = \eta\alpha/\|X\|_1$
```

The cost of Algorithm 4.7.1 is the cost of computing e^A plus the cost of about 8 Fréchet derivative evaluations, so obtaining e^A and the condition estimate multiplies the cost of obtaining just e^A by a factor of about 17. This factor can be reduced to 9 if the parameter t in the block 1-norm power method is reduced to 1, at a cost of slightly reduced reliability.

In our MATLAB implementation of Algorithm 4.7.1 we invoke the function `funm_condest1` from the Matrix Function Toolbox [26], which interfaces to the MATLAB function `normest1` that implements the block 1-norm estimation algorithm (Algorithm 1.6.12).

With the same matrices as in the test of the previous section we used Algorithm 4.7.1 to estimate $\|K(A)\|_1$ and also computed $\|K(A)\|_1$ exactly by forming $K(A)$ using Algorithm 1.6.8. Figure 4.3 plots the norms and the estimates. The worst underestimation ratio is 0.61, so the estimates are all within a factor 2 of the true 1-norm.

Finally, we invoked Algorithm 4.7.1 on the same set of test matrices and computed the "exact" exponential in 100 digit precision. Figure 4.4 plots the error in the

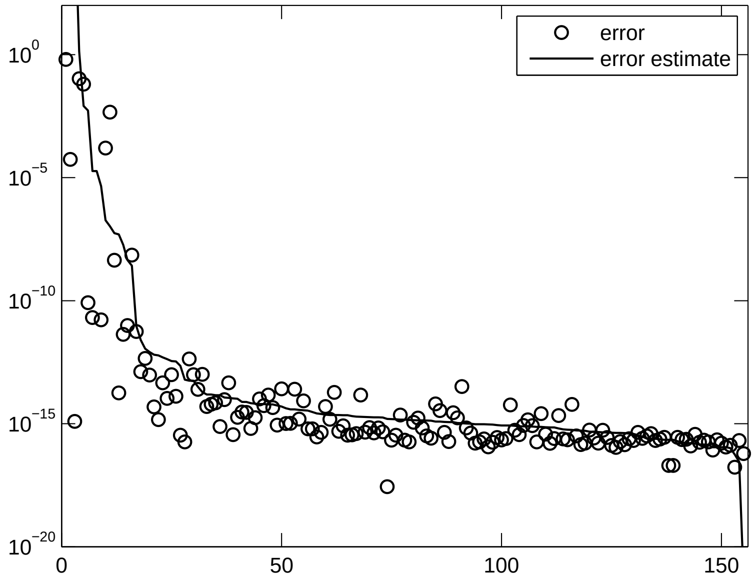

Figure 4.4: Normwise relative error for computed exponential and error estimate comprising condition number estimate times unit roundoff.

computed exponential along with the quantity γu: the condition estimate multiplied by the unit roundoff, regarded as an error estimate. If the scaling and squaring algorithm were forward stable and the condition estimate reliable we would expect the error to be bounded by $\phi(n)\gamma u$ for some low degree polynomial ϕ. The overall numerical stability of the scaling and squaring algorithm is not understood [31], but our experience is that the method usually does behave in a forward stable way. Figure 4.4 indicates that the condition estimate from Algorithm 4.7.1 provides a useful guide to the accuracy of the computed exponential from the algorithm.

4.8 Scaling and squaring algorithm for the exponential and its Fréchet derivative (II)

In this section we extend Algorithm 2.6.1 to an algorithm for simultaneously computing e^A and $L_{\exp}(A, E)$. Then we adapt it to an algorithm for computing e^A and estimating its condition number. The algorithms inherit the improved scaling strategy of Algorithm 2.6.1 and its ability to exploit triangularity.

We have established at the beginning of this chapter the notion that if a numerical algorithm is available for a matrix function, then it can be "Fréchet differentiated" to yield an algorithm for intertwining the computation of the matrix function itself and its Fréchet derivative. This was done to produce Algorithm 4.6.3, where we differentiated the evaluation schemes underling Algorithm 2.3.1.

We will evaluate e^A and $L_{\exp}(A, E)$ using the recurrence (4.15) for A and E with s being the scaling parameter determined by Algorithm 2.6.1. Our numerical experiment below shows that the algorithm behaves in a highly stable manner.

We begin by adapting Code Fragment 2.2.1 as follows

Code Fragment 4.8.1

```
Form $X = r_m(2^{-s}T)$ and $L = L_{r_m}(2^{-s}T, 2^{-s}E)$.
Replace diag($X$) by exp($2^{-s}$diag($T$)).
for $i = s-1\colon -1\colon 0$
    $L \leftarrow XL + LX$
    $X \leftarrow X^2$
    Replace diag($X$) by exp($2^{-i}$diag($T$)).
    Replace (first) superdiagonal of $X$ by explicit formula
    for superdiagonal of exp($2^{-i}T$) from [31, eq. (10.42)].
end
```

We write the algorithm using the functions `normest` and `ell` as defined in Algorithm 2.5.1.

Algorithm 4.8.2 *Given $A, E \in \mathbb{C}^{n\times n}$ this algorithm computes $X = e^A$ and $L = L_{\exp}(A, E)$ by a scaling and squaring algorithm. It uses the functions and the parameters of Algorithm* 2.6.1. *The algorithm is intended for IEEE double precision arithmetic.*

```
$A_2 = A^2$
$d_6 = \texttt{normest}(A_2, 3)^{1/6}$, $\eta_1 = \max\big(\texttt{normest}(A_2, 2)^{1/4}, d_6\big)$
if $\eta_1 \le \theta_3$ and $\texttt{ell}(A, 3) = 0$
  $[X, L] = \mathbf{XL}(3, A, A_2, E)$ % Code Fragment 4.6.2.
  quit
end
$A_4 = A_2^2$, $d_4 = \|A_4\|_1^{1/4}$
$\eta_2 = \max\big(d_4, d_6\big)$
if $\eta_2 \le \theta_5$ and $\texttt{ell}(A, 5) = 0$
  $[X, L] = \mathbf{XL}(5, A, A_2, A_4, E)$
  quit
end
$A_6 = A_2A_4$, $d_6 = \|A_6\|_1^{1/6}$
$d_8 = \texttt{normest}(A_4, 2)^{1/8}$, $\eta_3 = \max(d_6, d_8)$
for $m = [7, 9]$
    if $\eta_3 \le \theta_m$ and $\texttt{ell}(A, m) = 0$
      $[X, L] = \mathbf{XL}(m, A, A_2, A_4, A_6, E)$
    quit
    end
end
$\eta_4 = \max\big(d_8, \texttt{normest}(A_4, A_6)^{1/10}\big)$
$\eta_5 = \min(\eta_3, \eta_4)$
$s = \max\big(\lceil \log_2(\eta_5/\theta_{13})\rceil, 0\big)$
$s = s + \texttt{ell}(2^{-s}A, 13)$
$A \leftarrow 2^{-s}A$, $A_2 \leftarrow 2^{-2s}A_2$, $A_4 \leftarrow 2^{-4s}A_4$, $A_6 \leftarrow 2^{-6s}A_6$
$E \leftarrow 2^{-s}E$
$[X, L] = \mathbf{XL}(13, A, A_2, A_4, A_6, E)$
if $A$ is triangular
```

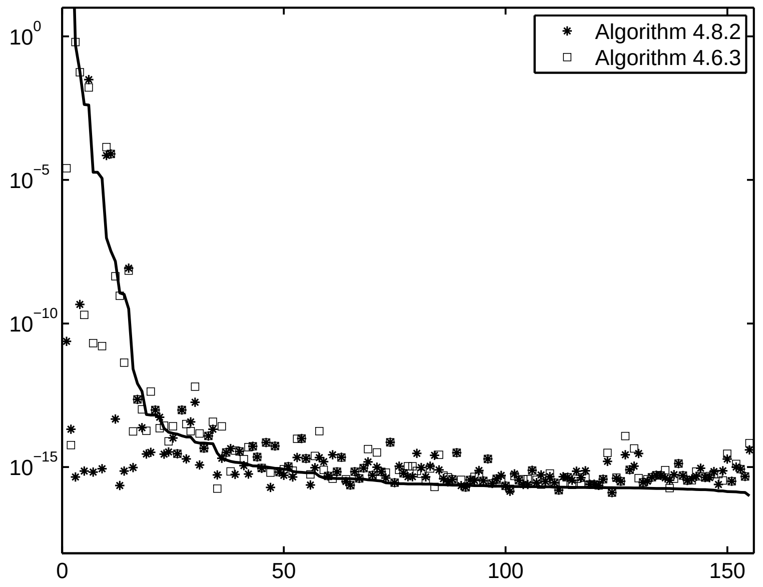

Figure 4.5: Normwise relative errors in Fréchet derivatives $L_{\exp}(A, E)$ computed by Algorithm 4.6.3 and Algorithm 4.8.2 for 155 matrices A with a different random E for each A, along with estimate of $\mathrm{cond}(L_{\exp}, A)u$ (solid line).

```
    Invoke Code Fragment 4.8.1.
else
    for $k = 1\colon s$
        $L \leftarrow XL + LX$
        $X \leftarrow X^2$
    end
end
```

We now assess this algorithm. Since Algorithm 4.6.3 shows superiority over the Kronecker–Sylvester algorithm in both "squaring" and "exponential" variants, it suffices to test Algorithm 4.8.2 versus Algorithm 4.6.3. We *exactly* use the same test matrices used for the experiment shown in Figure 4.1. We plot the test in Figure 4.5, and then present the same data as a performance profile in Figure 4.6. Obviously, Algorithm 4.8.2 demonstrates superiority over Algorithm 4.6.3. It really reflects the advantage of the scaling strategy and exploitation of triangularity that Algorithm 2.6.1 uses.

Similar to Algorithm 4.7.1, the following algorithm computes the matrix exponential and estimates its condition number. It is an adapted version of Algorithm 4.8.2 and stores the matrices shown in Table 4.3.

Algorithm 4.8.3 *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes $X = e^A$ by the scaling and squaring method* (*Algorithm* 2.6.1) *and an estimate $\gamma \approx \kappa_{\exp}(A)$ using the block* 1-*norm estimator* (*Algorithm* 1.6.12). *It uses the functions and the parameters of Algorithm* 2.6.1. *The algorithm is intended for IEEE double precision arithmetic.*

```
$\alpha = \|A\|_1$
$A_2 = A^2$
```

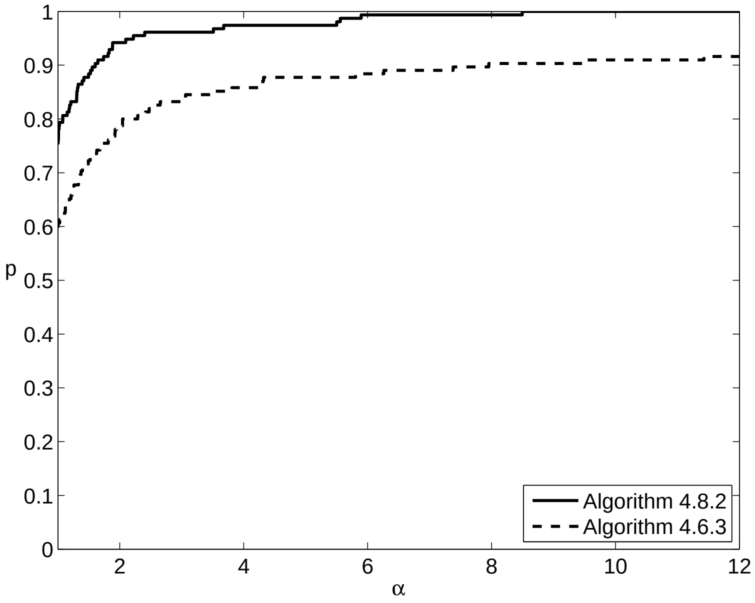

Figure 4.6: Same data as in Figure 4.5 presented as a performance profile.

```
3  $d_6 = \texttt{normest}(A_2, 3)^{1/6}$, $\eta_1 = \max\big(\texttt{normest}(A_2, 2)^{1/4}, d_6\big)$
4  if $\eta_1 \le \theta_3$ and $\texttt{ell}(A, 3) = 0$
5     $[X, W_3$, “LU fact.”$] = \mathbf{XL}(3, A, A_2)$ % Code Fragment 4.6.2.
6     Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(A)\|_1$.
      ...... To compute $L = L_{\exp}(A, E)$ for a given $E$:
7               $L = \mathbf{XL}(3, A, E, X, A_2, W_3$, “LU fact.”$)$
8     ...... To compute $L = L^\star_{\exp}(A, E)$ for a given $E$:
9               $L = \mathbf{XL}(3, A, E^*, X, A_2, W_3$, “LU fact.”$)$
10              $L = L^*$
11    goto line 58
12 end
13 $A_4 = A_2^2$, $d_4 = \|A_4\|_1^{1/4}$
14 $\eta_2 = \max\big(d_4, d_6\big)$
15 if $\eta_2 \le \theta_5$ and $\texttt{ell}(A, 5) = 0$
16    $[X, W_5$, “LU fact.”$] = \mathbf{XL}(5, A, A_2, A_4)$
17    Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(A)\|_1$.
      ...... To compute $L = L_{\exp}(A, E)$ for a given $E$:
18              $L = \mathbf{XL}(5, A, E, X, A_2, A_4, W_5$, “LU fact.”$)$
19    ...... To compute $L = L^\star_{\exp}(A, E)$ for a given $E$:
20              $L = \mathbf{XL}(5, A, E^*, X, A_2, A_4, W_5$, “LU fact.”$)$
21              $L = L^*$
22    goto line 58
23 end
24 $A_6 = A_2 A_4$, $d_6 = \|A_6\|_1^{1/6}$
25 $d_8 = \texttt{normest}(A_4, 2)^{1/8}$, $\eta_3 = \max(d_6, d_8)$
26 for $m = [7, 9]$
27     if $\eta_3 \le \theta_m$ and $\texttt{ell}(A, m) = 0$
```

```
28          $[X, \{A_{2k}\}_{k=4}^{(m-1)/2}, W_m, \text{“LU fact.”}] = \mathbf{XL}(m, A, \{A_{2k}\}_{k=1}^{3})$
            % The output matrices need to be stored (see Table 4.3).
29          Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(A)\|_1$.
            ...... To compute $L = L_{\exp}(A, E)$ for a given $E$:
30                    $L = \mathbf{XL}(m, A, E, X, \{A_{2k}\}_{k=1}^{(m-1)/2}, W_m, \text{“LU fact.”})$
31          ...... To compute $L = L_{\exp}^{\star}(A, E)$ for a given $E$:
32                    $L = \mathbf{XL}(m, A, E^*, X, \{A_{2k}\}_{k=1}^{(m-1)/2}, W_m, \text{“LU fact.”})$
33                    $L = L^*$
34       goto line 58
35    end
36    % Use degree $m = 13$.
37    $\eta_4 = \max\big(d_8, \texttt{normest}(A_4, A_6)^{1/10}\big)$
38    $\eta_5 = \min(\eta_3, \eta_4)$
39    $s = \max\big(\lceil \log_2(\eta_5/\theta_{13}) \rceil, 0\big)$
40    $s = s + \texttt{ell}(2^{-s}A, 13)$
41    $A \leftarrow 2^{-s}A$, $A_2 \leftarrow 2^{-2s}A_2$, $A_4 \leftarrow 2^{-4s}A_4$, $A_6 \leftarrow 2^{-6s}A_6$
42    $[X_s, W, \text{“LU fact.”}] = \mathbf{XL}(13, A, \{A_{2k}\}_{k=1}^{3})$
43    if $A$ is triangular
44       Invoke Code Fragment 2.2.1 and return $\{X_i\}_{i=1}^{s}$
45    else
46       for $i = s\colon -1\colon 1$
47          $X_{i-1} = X_i^2$
48       end
49    end
      % $\{X_i\}_{i=1}^{s}$ and the output matrices need to be stored (see Table 4.3).
50    $X = X_0$
51    Use Algorithm 1.6.12 to produce an estimate $\eta \approx \|L_{\exp}(\widetilde{A})\|_1$,
      where $\widetilde{A}$ denotes the original input matrix $A$.
      ...... To compute $L_{\exp}(\widetilde{A}, E)$ for a given $E$:
52                $E \leftarrow 2^{-s}E$
53                $L = \mathbf{XL}(13, A, E, X_s, \{A_{2k}\}_{k=1}^{3}, W, \text{“LU fact.”})$
54                for $i = s\colon -1\colon 1$
55                   $L \leftarrow X_i L + L X_i$
56                end
      ...... To compute $L_{\exp}^{\star}(\widetilde{A}, E)$ for a given $E$:
57                    Execute lines 52–56 with $E$ replaced by its conjugate
                      transpose and take the conjugate transpose of the result.
58    $\gamma = \eta\alpha/\|X\|_1$
```

With the same test matrices used in Figure 4.3, we invoked Algorithm 4.8.3 to estimate $\|K(A)\|_1$ and also computed $\|K(A)\|_1$ exactly by forming $K(A)$ using Algorithm 1.6.8, where $L_{\exp}(A, E)$ therein is computed by Algorithm 4.8.2. Figure 4.7 plots the norms and the estimates. The worst underestimation ratio is 0.61, which is same as the underestimation ratio produced by Algorithm 4.7.1. However, we observe by comparing the plots in the bottom of Figure 4.7 and Figure 4.3 that the number of matrices for which whose $\|K(A)\|_1$ is underestimated by Algorithm 4.8.3 is less than the number of matrices for which $\|K(A)\|_1$ is underestimated by Algorithm 4.7.1.

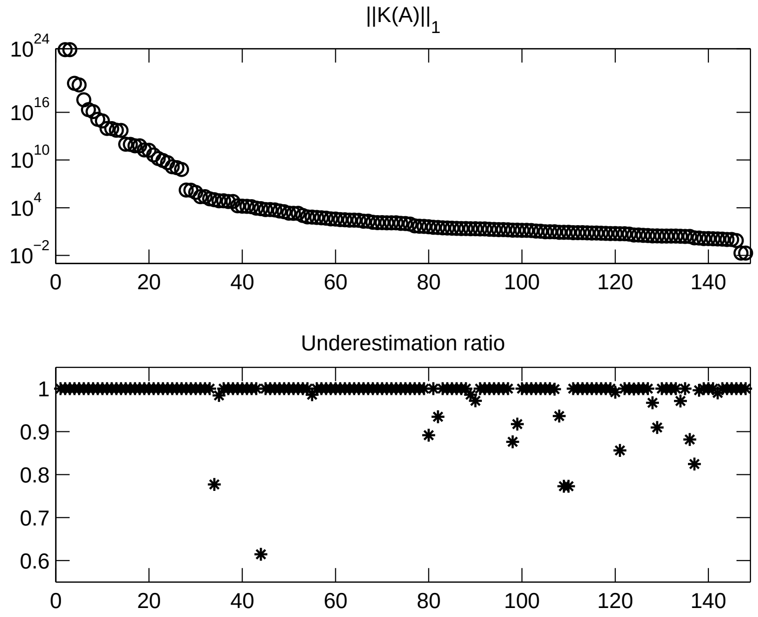

Figure 4.7: $\|K(A)\|_1$ and underestimation ratio $\eta/\|K(A)\|_1$, where η is the estimate of $\|K(A)\|_1$ produced by Algorithm 4.8.3.

Finally, for full comparison we use Algorithm 4.8.3 to reproduce the test shown in Figure 4.4, which was produced by Algorithm 4.7.1. Figure 4.8 displays the results.

Our conclusion is that Algorithm 4.8.3 behaves in a stable manner and reliably estimates the condition number. Moreover, it is more stable than Algorithm 4.7.1.

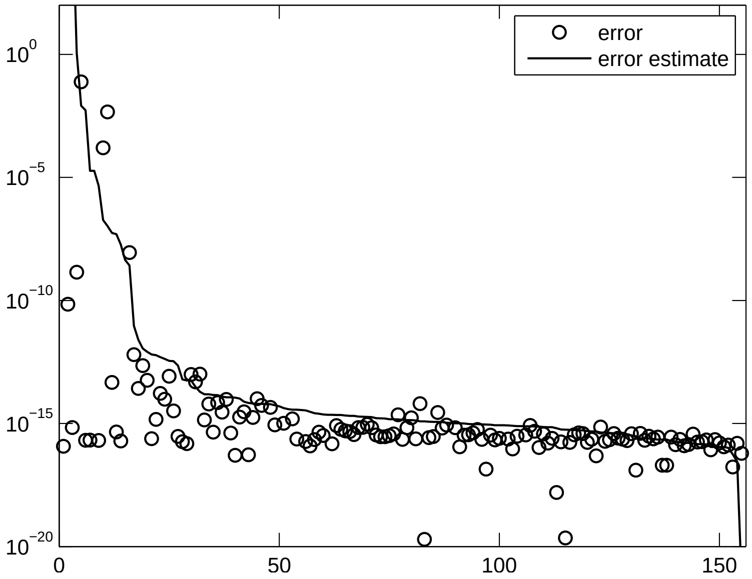

Figure 4.8: Normwise relative error for computed exponential and error estimate comprising condition number estimate times unit roundoff.

Chapter 5

The Complex Step Approximation to the Fréchet Derivative of a Matrix Function

5.1 Introduction

Given a Fréchet differentiable matrix function $f : \mathbb{C}^{n\times n} \rightarrow \mathbb{C}^{n\times n}$ (see Section 1.6), its Fréchet derivative determines the sensitivity of f to small perturbations in the input matrix. When calculating $f(A)$, it is desirable to be able to efficiently estimate $\operatorname{cond}(f, A)$, and from 1.6.4 and (1.8) we see that this will in general required the evaluation of $L_f(A, Z)$ for certain Z. Thus it is important to be able to compute or estimate the Fréchet derivative reliably and efficiently. A natural approach is to approximate the Fréchet derivative by the finite difference

$$L_f(A, E) \approx \frac{f(A + hE) - f(A)}{h}, \tag{5.1}$$

for a suitably chosen h. This approach has the drawback that h needs to be selected to balance truncation errors with errors due to subtractive cancellation in floating point arithmetic, and as a result the smallest relative error that can be obtained is of order $u^{1/2}$, where u is the unit roundoff [31, Sec. 3.4].

In this work we pursue a completely different approach. Like (5.1), it requires one additional function evaluation, but now at a complex argument:

$$L_f(A, E) \approx \operatorname{Im} \frac{f(A + ihE)}{h}, \tag{5.2}$$

where $i = \sqrt{-1}$. This complex step (CS) approximation is known in the scalar case but has not, to our knowledge, been applied previously to matrix functions. The approximation requires A and E to be real and f to be real-valued at real arguments. The advantage of (5.2) over (5.1) is that in principle (5.2) allows h to be chosen as small as necessary to obtain an accurate approximation to $L_f(A, E)$, without cancellation errors contaminating the result in floating point arithmetic. It also provides an approximation to $f(A)$ from this single function evaluation. Unlike (5.1), however, it requires the use of complex arithmetic.

In Section 5.2 we review the complex step approximation for scalars. We extend the approximation to the matrix case in Section 5.3 and show that it is second order accurate for analytic functions f. The computational cost is considered in Section 5.4. In Section 5.5 we show that the CS approximation is also second order accurate for the matrix sign function, which is not analytic. In Section 5.6 we show that good accuracy can be expected in floating point arithmetic for sufficiently small h, but that if the method for evaluating f uses complex arithmetic then catastrophic cancellation is likely to vitiate the approximation. Finally, numerical experiments are given in Section 5.7 to illustrate the advantages of the CS approximation over finite differences, the role of the underlying method for evaluating f, and the application of the approximation to condition number estimation.

5.2 Complex step approximation: scalar case

For an analytic function $f : \mathbb{R} \to \mathbb{R}$, the use of complex arithmetic for the numerical approximation of derivatives of f was introduced by Lyness [48] and Lyness and Moler [49]. The earliest appearance of the CS approximation itself appears to be in Squire and Trapp [66]. Later uses of the formula appear in Kelley [41, Sec. 2.5.2] and Cox and Harris [12], while Martins, Sturdza, and Alonso [39] and Shampine [62] investigate the implementation of the approximation in high level languages.

The scalar approximation can be derived from the Taylor series expansion, with $x_0, h \in \mathbb{R}$,

$$f(x_0 + ih) = \sum_{k=0}^{\infty} (ih)^k \frac{f^{(k)}(x_0)}{k!} = f(x_0) + ihf'(x_0) - h^2 \frac{f''(x_0)}{2!} + O(h^3). \tag{5.3}$$

Equating real and imaginary parts yields

$$f(x_0) = \operatorname{Re} f(x_0 + ih) + O(h^2), \qquad f'(x_0) = \operatorname{Im} \frac{f(x_0 + ih)}{h} + O(h^2). \tag{5.4}$$

Unlike in the finite difference approximation (5.1), subtractive cancellation is not intrinsic in the expression $\operatorname{Im} f(x_0+ih)/h$, and this approximation therefore offers the promise of allowing h to be selected based solely on the need to make the truncation error sufficiently small. Practical experience reported in the papers cited above has indeed demonstrated the ability of (5.4) to produce accurate approximations, even with h as small as 10^{-100}, which is the value used in software at the National Physical Laboratory according to [12].

In the next section we generalize the complex step approximation to real-valued matrix functions over $\mathbb{R}^{n\times n}$.

5.3 Complex step approximation: matrix case

Assume that the Fréchet derivative $L_f(A, E)$ is defined, a sufficient condition for which is that f is $2n - 1$ times continuously differentiable on an open subset of $\mathbb{R}$ or $\mathbb{C}$ containing the spectrum of A by Theorem 1.6.3. Replacing E by ihE in (1.7), where E is independent of h, and using the linearity of L_f, we obtain

$$f(A + ihE) - f(A) - ihL_f(A, E) = o(h).$$

Thus if $f : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ and $A, E \in \mathbb{R}^{n\times n}$ then

$$\lim_{h\to 0} \operatorname{Im} \frac{f(A + ihE)}{h} = L_f(A, E),$$

which justifies the CS approximation (5.2). However, this analysis does not reveal the rate of convergence of the approximation as $h \to 0$. To determine the rate we need a more careful analysis with stronger assumptions on f.

Denote by $L_f^{[j]}(A, E)$ the jth Fréchet derivative of f at A in the direction E, given by

$$L_f^{[j]}(A, E) = \frac{d^j}{dt^j} f(A + tE)\Big|_{t=0},$$

with $L_f^{[0]}(A, E) = f(A)$ and $L_f^{[1]} \equiv L_f$. The next result provides a Taylor expansion of f in terms of the Fréchet derivatives.

Theorem 5.3.1 *Let $f : \mathbb{C} \to \mathbb{C}$ have the power series expansion $f(x) = \sum_{k=0}^{\infty} a_k x^k$ with radius of convergence r. Let $A, E \in \mathbb{C}^{n\times n}$ such that $\rho(A+\mu E) < r$, where $\mu \in \mathbb{C}$. Then*

$$f(A + \mu E) = \sum_{k=0}^{\infty} \frac{\mu^k}{k!} L_f^{[k]}(A, E),$$

where

$$L_f^{[k]}(A, E) = \sum_{j=k}^{\infty} a_j L_{x^j}^{[k]}(A, E).$$

The Fréchet derivatives of the monomials satisfy the recurrence

$$L_{x^j}^{[k]}(A, E) = A L_{x^{j-1}}^{[k]}(A, E) + kE L_{x^{j-1}}^{[k-1]}(A, E). \tag{5.5}$$

Proof. Najfeld and Havel [56, pp. 349–350] show that

$$(A + \mu E)^j = \sum_{k=0}^{j} \frac{\mu^k}{k!} L_{x^j}^{[k]}(A, E)$$

and that the $L_{x^j}^{[k]}$ satisfy the recurrence (5.5). By the assumption on the spectral radius, we have

$$\begin{aligned} f(A + \mu E) &= \sum_{j=0}^{\infty} a_j (A + \mu E)^j \\ &= \sum_{j=0}^{\infty} a_j \left(\sum_{k=0}^{j} \frac{\mu^k}{k!} L_{x^j}^{[k]}(A, E) \right) \\ &= \sum_{k=0}^{\infty} \frac{\mu^k}{k!} \sum_{j=k}^{\infty} a_j L_{x^j}^{[k]}(A, E). \end{aligned}$$

By the sum rule for Fréchet derivatives [31, Thm. 3.2], the inner summation in the last expression is $L_f^{[k]}(A, E)$. □

Replacing μ in Theorem 5.3.1 by ih, where $h \in \mathbb{R}$, we obtain

$$f(A + ihE) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} h^{2k} L_f^{[2k]}(A, E) + i \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} h^{2k+1} L_f^{[2k+1]}(A, E).$$

To be able to extract the desired terms from this expansion we need $f : \mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$ and $A, E \in \mathbb{R}^{n \times n}$. Then

$$f(A) = \operatorname{Re} f(A + ihE) + O(h^2), \tag{5.6a}$$

$$L_f(A, E) = \operatorname{Im} \frac{f(A + ihE)}{h} + O(h^2). \tag{5.6b}$$

Theorem 5.3.1 can be used to develop approximations to higher Fréchet derivatives (cf. [47]), but we will not pursue this here.

The analyticity of f is sufficient to ensure a second order approximation, but it is not necessary. In Section 5.5 we consider the matrix sign function, which is not analytic, and show that the CS approximation error is nevertheless $O(h^2)$.

5.4 Cost analysis

The CS approximation has the major attraction that it is trivial to implement, as long as code is available for evaluating f at a complex argument. We now look at the computational cost, assuming that the cost of evaluating $f(A)$ is $O(n^3)$ flops, where a flop is a real scalar addition or multiplication.

First we note that the cost of multiplying two $n \times n$ real matrices is $2n^3$ flops. To multiply two $n \times n$ complex matrices requires $8n^3$ flops if complex scalar multiplications are done in the obvious way. However, by using a formula for multiplying two complex scalars in 3 real multiplications the cost can be reduced to $6n^3$ flops at the cost of weaker rounding error bounds [28], [29, Chap. 23]. For an algorithm for computing $f(A)$ whose cost is dominated by level 3 BLAS operations it follows [27], [7] that the cost of computing the CS approximation to $L_f(A, E)$ is 3–4 times that of the cost of computing $f(A)$ alone, though of course the CS approximation does yield approximations to both $f(A)$ and $L_f(A, E)$.

Next, we compare with another way of computing the Fréchet derivative, which is from Theorem 4.2.1

$$f\left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} f(A) & L_f(A, E) \\ 0 & f(A) \end{bmatrix}. \tag{5.7}$$

This formula requires the evaluation of f in real arithmetic at a $2n \times 2n$ matrix, which in principle is 8 times the cost of evaluating $f(A)$. However, it will usually be possible to reduce the cost by exploiting the block triangular, block Toeplitz structure of the argument. Hence this approach may be of similar cost to the CS approximation. In Section 4.6 we note a drawback of (5.7) connected with the scaling of E. Since $L_f(A, \alpha E) = \alpha L_f(A, E)$ the norm of E can be chosen at will, but the choice may affect the accuracy of a particular algorithm based on (5.7) and it is difficult to know what is the optimal choice.

Another comparison can be made under the assumption that f is a polynomial, which is relevant since a number of algorithms for evaluating $f(A)$ make use of polynomials or rational approximations. Let π_m be the number of matrix multiplications required to evaluate $f(A)$ by a particular scheme. Theorem 4.4.1 shows that for a wide class of schemes the extra cost of computing $L_f(A,E)$ via the scheme obtained by differentiating the given scheme for $f(A)$ is at most $2\pi_m$ if terms formed during the evaluation of $f(A)$ are re-used. The CS approximation is therefore likely to be more costly, but it requires no extra coding effort and is not restricted to polynomials.

5.5 Sign function

The matrix sign function is an example of a function that is not analytic, so the analysis in Section 5.3 showing a second order error for the CS approximation is not applicable. We prove in this section that the CS approximation nevertheless has an error of second order in h for the sign function.

For $A \in \mathbb{C}^{n\times n}$ with no eigenvalues on the imaginary axis, $\operatorname{sign}(A)$ is the limit of the Newton iteration

$$X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-1}\right), \qquad X_0 = A. \tag{5.8}$$

Moreover, the iterates E_k defined by

$$E_{k+1} = \frac{1}{2}\left(E_k - X_k^{-1}E_kX_k^{-1}\right), \qquad E_0 = E \tag{5.9}$$

converge to $L_{\text{sign}}(A,E)$. Both iterations converge quadratically; see [31, Thms. 5.6, 5.7]. The next theorem uses these iterations to determine the order of the error of the CS approximation.

Theorem 5.5.1 *Let $A, E \in \mathbb{R}^{n\times n}$ and let A have no eigenvalues on the imaginary axis. In the iteration*

$$Z_{k+1} = \frac{1}{2}\left(Z_k + Z_k^{-1}\right), \qquad Z_0 = A + ihE, \qquad h \in \mathbb{R} \tag{5.10}$$

the Z_k are nonsingular for all h sufficiently small and

$$\begin{aligned}\operatorname{Re}\operatorname{sign}(A+ihE) &= \lim_{k\to\infty}\operatorname{Re} Z_k = \operatorname{sign}(A) + O(h^2),\\ \operatorname{Im}\operatorname{sign}(A+ihE) &= \lim_{k\to\infty}\operatorname{Im}\frac{Z_k}{h} = L_{\text{sign}}(A,E) + O(h^2).\end{aligned}$$

Proof. Write $Z_k = M_k + iN_k \equiv \operatorname{Re} Z_k + i\operatorname{Im} Z_k$. It suffices to show that

$$M_k = X_k + O(h^2), \qquad N_k = hE_k + O(h^3), \tag{5.11}$$

where X_k and E_k satisfy (5.8) and (5.9), which we prove by induction. First, set $k = 1$ and assume that $\rho(EA^{-1}) < 1/h$, which is true for sufficiently small h. Then we have the expansion

$$(A+ihE)^{-1} = A^{-1}\sum_{j=0}^{\infty}(-ih)^j(EA^{-1})^j.$$

Therefore the first iteration of (5.10) gives

$$M_1 = \frac{1}{2}\left(A + A^{-1}\right) + O(h^2), \qquad N_1 = \frac{h}{2}\left(E - A^{-1}EA^{-1}\right) + O(h^3),$$

so that (5.11) holds for $k = 1$. Suppose that (5.11) holds for k. Then we can write $M_k = X_k + h^2 R_k$, for some matrix $R_k \in \mathbb{R}^{n\times n}$. Assuming $\rho(R_k X_k^{-1}) < 1/h^2$, which again is true for sufficiently small h, we have

$$M_k^{-1} = X_k^{-1}\sum_{j=0}^{\infty} h^{2j}(-R_k X_k^{-1})^j = X_k^{-1} + O(h^2). \tag{5.12}$$

Now assume $\rho(N_k M_k^{-1}) < 1$, which is true for sufficiently small h since $N_k = O(h)$. Then, using (5.12) and $(M_k + iN_k)^{-1} = M_k^{-1}\sum_{j=0}^{\infty}(-i)^j(N_k M_k^{-1})^j$, we have

$$\begin{aligned} M_{k+1} &= \frac{1}{2}\big(M_k + M_k^{-1} + M_k^{-1}\sum_{j=1}^{\infty}(-1)^j(N_k M_k^{-1})^{2j}\big) \\ &= \frac{1}{2}\left(X_k + X_k^{-1}\right) + O(h^2), \\ N_{k+1} &= \frac{1}{2}\big(N_k - M_k^{-1}N_k M_k^{-1} + M_k^{-1}\sum_{j=1}^{\infty}(-1)^{j+1}(N_k M_k^{-1})^{2j+1}\big) \\ &= \frac{h}{2}\left(E_k - X_k^{-1}E_k X_k^{-1}\right) + O(h^3), \end{aligned}$$

which completes the induction. □

Note that another approach to proving Theorem 5.5.1 would be to use existing perturbation theory for the matrix sign function, such as that of Sun [67]. However, the perturbation expansions in [67] make use of the Schur and Jordan forms and do not readily permit the real and imaginary parts to be extracted.

The cost of evaluating the E_k in (5.9) is twice the cost of evaluating the X_k (assuming an LU factorization of X_k is computed for (5.8) and re-used). The CS approximation provides an approximation to $L_{\text{sign}}(A, E)$ by iterating (5.8) with a complex starting matrix, so the cost is 3–4 times that for computing $\text{sign}(A)$ alone. Given the ease of implementing (5.9) one would probably not use the CS approximation with the Newton iteration. However, with other methods for evaluating $\text{sign}(A)$, of which there are many [31, Chap. 5], the economics may be different.

5.6 Accuracy

What accuracy can we expect from the CS approximation in floating point arithmetic? Equivalently, how accurately is the imaginary part of $f(A + ihE)$ computed when h is small, bearing in mind that the imaginary part is expected to be of order h, and hence much smaller than the real part? In order to obtain an accurate result it is necessary that the information contained in hE is accurately transmitted through to the imaginary part of $f(A + ihE)$, and this is most likely when the imaginary part does not undergo large growth and then reduction (due to subtractive cancellation) during the computation.

It is straightforward to show that the sum and product of two complex matrices with tiny imaginary part has tiny imaginary part and that the inverse of a matrix with tiny imaginary part has tiny imaginary part. It follows when a polynomial or rational function with real coefficients is evaluated at a matrix with tiny imaginary part the result has tiny imaginary part. Hence when we evaluate $f(A+ihE)$ using an algorithm for f based on polynomial or rational approximations with real coefficients there is no a priori reason to expect damaging cancellation within the imaginary part. In particular, there is no a priori lower bound on the accuracy that can be expected, unlike for the finite difference approximation (5.1), for which such a lower bound is of order $u^{1/2}$.

However, numerical instability is possible if the algorithm for $f(A)$ itself employs complex arithmetic, as we now show. Suppose we compute $C = \cos(A)$ by the simple algorithm [31, Alg 12.7]

$$X = e^{iA}, \tag{5.13a}$$
$$C = (X + X^{-1})/2. \tag{5.13b}$$

The CS approximation gives

$$L_{\cos}(A, E) \approx \operatorname{Im} \frac{\cos(A + ihE)}{h} = \operatorname{Im} \frac{e^{iA-hE} + e^{-iA+hE}}{2h}. \tag{5.14}$$

Making the simplifying assumption that A and E commute, in which case $L_{\cos}(A, E) = -E\sin(A)$ [31, Sec. 12.2], we have

$$\begin{aligned} e^{iA-hE} + e^{-iA+hE} &= e^{iA}e^{-hE} + e^{-iA}e^{hE} \\ &= \cos(A)(e^{-hE} + e^{hE}) + i\sin(A)(e^{-hE} - e^{hE}), \end{aligned}$$

and the CS approximation reduces to

$$L_{\cos}(A, E) \approx \frac{\sin(A)(e^{-hE} - e^{hE})}{2h}.$$

Thus $-E$ is being approximated by $(e^{-hE} - e^{hE})/(2h)$, and the latter expression suffers massive subtractive cancellation for small h. We illustrate in Figure 5.1 with A and E both the scalar 1. These computations, and those in the next section, were performed in MATLAB 7.10 (R2010a), with unit roundoff $u = 2^{-53} \approx 1.1 \times 10^{-16}$. Note that the CS approximation deteriorates once h decreases below 10^{-5}, yielding maximum accuracy of about 10^{-10}. This weakness of the CS approximation for scalar problems is noted by Martins, Sturdza, and Alonso [50].

The unwanted resonance between complex arithmetic in the underlying algorithm and the pure imaginary perturbation used by the CS approximation affects any algorithm based on the Schur form, such as those in [13], [22], [46]. Since $B := A+ihE$ is nonreal, the complex Schur form $B = QTQ^*$, with Q unitary and T upper triangular, must be used. In general, Q will have real and imaginary parts of similar norm (since A may have some nonreal eigenvalues), and likewise for T. The $O(h)$ imaginary part of $f(B) = Qf(T)Q^*$ is therefore the result of massive cancellation, which signals a serious loss of accuracy of the CS approximation in floating point arithmetic. The first experiment in the next section illustrates this phenomenon.

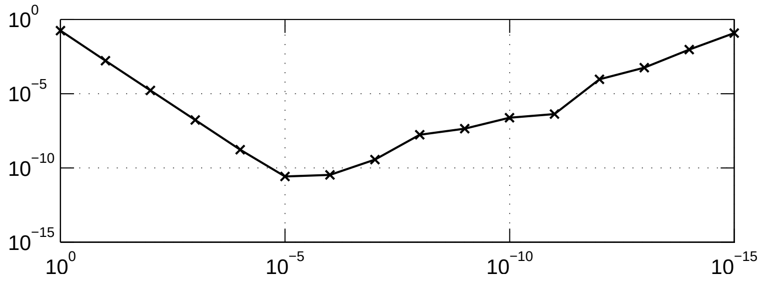

Figure 5.1: Relative errors for approximating $L_{\cos}(A, E)$ for scalars $A = E = 1$ using the CS approximation with (5.14) and $h = 10^{-k}$, $k = 0\colon 15$.

5.7 Numerical experiments

We now give some experiments to illustrate the CS approximation and its advantage over the finite difference approximation (5.1).

For our first experiment we take `A = gallery('triw',10)`, the unit upper triangular matrix with every superdiagonal element equal to -1, and a random matrix `E = randn(10)`. The function is $f(A) = e^A$, which we compute both by the MATLAB function `expm`, which implements the scaling and squaring method [30], and by the MATLAB function `funm`, which handles general matrix functions via the Schur–Parlett method [13] and treats the diagonal Schur form blocks specially in the case of the exponential.

For h ranging from 10^{-3} to 10^{-20}, Figure 5.2 plots the normwise relative error $\|L_{\exp}(A, E) - \widehat{L}\|_1 / \|L_{\exp}(A, E)\|_1$, where $\widehat{L}$ represents the approximate Fréchet derivative from the finite-difference approximation (5.1) or the CS approximation (5.2). The "exact" $L_{\exp}(A, E)$ is obtained via the relation (5.7) evaluated at 100 digit precision using Symbolic Math Toolbox.

In this example the CS approximation has full accuracy when using `expm` with $h \le 10^{-8}$, reflecting its $O(h^2)$ error (see (5.6b)). By contrast, the finite difference approximation returns its best result at around $h = 10^{-8}$ with error of $O(h)$, and then diverges as h decreases, just as the theory on the choice of h suggests [31, Sec. 3.4]. Equipping the CS approximation with `funm` leads to poor results, due to the complex arithmetic inherent in the Schur form (see Section 5.6), though the results are superior to those obtained with finite differences. Note that the fact that A has real eigenvalues does not help: as a result of A being highly nonnormal (indeed defective, with a single Jordan block), the perturbed matrix $B = A + ihE$ has eigenvalues with imaginary parts of order 10^{-2} for all the chosen h!

Interestingly, the error for the CS approximation with `expm` remains roughly constant at around 10^{-16} for h decreasing all the way down to 10^{-292}, at which point it starts to increase, reaching an error of 10^{-1} by the time h underflows to zero at around 10^{-324}.

The performance of the CS approximation is of course method-dependent. Figure 5.3 repeats the previous experiment except that we set $a_{15} = 10^6$ and compare `expm` with `expm_new`, the latter function using Algorithm 2.6.1 designed to avoid overscaling. The large off-diagonal element of A causes `expm` to overscale in its scaling phase, that is, to reduce $\|A\|$ much further than necessary in order to achieve an

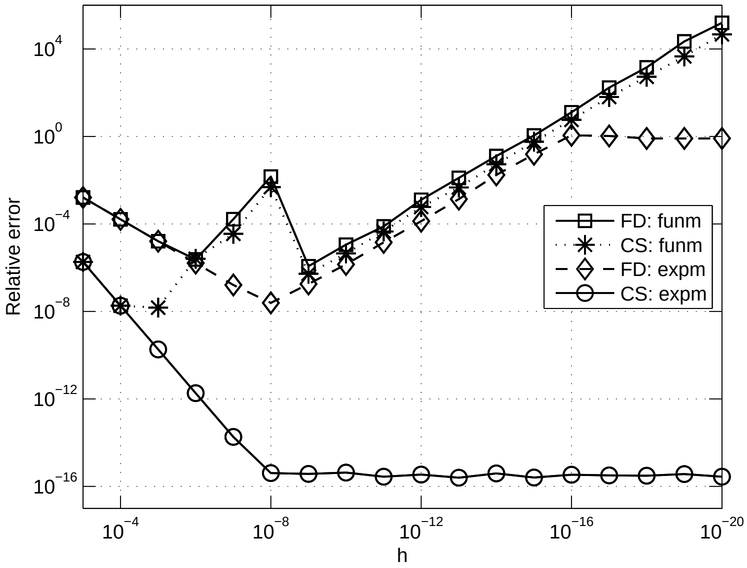

Figure 5.2: Relative errors for approximating $L_{\rm exp}(A,E)$ using the CS approximation and the finite difference (FD) approximation (5.1), for $h=10^{-k}$, $k=3\colon 20$.

accurate result: the relative error of the computed exponentials is of order 10^{-11} for `expm` and 10^{-16} for `expm_new`. We see from Figure 5.3 that there is a corresponding difference in the accuracy of the Fréchet derivative approximations. But the superiority of the CS approximation over the finite difference approximation is evident for both `expm` and `expm_new`.

Our next experiment involves a different function: the principal matrix square root, $A^{1/2}$. The Fréchet derivative $L_{\rm sqrt}(A,E)$ is the solution L of $LX+XL=E$, where $X=A^{1/2}$ [31, Sec. 6.1]. The product form of the Denman–Beavers iteration,

$$M_{k+1}=\frac{1}{2}\biggl(I+\frac{M_k+M_k^{-1}}{2}\biggr),\qquad M_0=A, \tag{5.15}$$

$$X_{k+1}=\frac{1}{2}X_k(I+M_k^{-1}),\qquad X_0=A, \tag{5.16}$$

is a variant of the Newton iteration, and $M_k\to I$ and $X_k\to A^{1/2}$ quadratically as $k\to\infty$ [31, Sec. 6.3]. With A the 8×8 Frank matrix (`gallery('frank',8)`), which has positive real eigenvalues, and `E = randn(8)`, we apply the CS approximation using this iteration as the means for evaluating $(A+ihE)^{1/2}$. Figure 5.4 shows the results, along with the errors from finite differencing. Again we see second order convergence of the CS approximations, which follows from the analyticity of the square root. Note, however, that the minimal relative error of order 10^4u. Since $\|L_{\rm sqrt}(A,E)\|_1\approx 9\times 10^3$ this is consistent with the maxim that the condition number of the condition number is the condition number [16].

Our final experiment illustrates the use of the CS approximation in condition estimation. As (1.6) shows, to estimate $\mathrm{cond}(f,A)$ we need to estimate $\|L_f(A)\|$, and this can be done by applying a matrix norm estimator to the Kronecker matrix form $K_f(A)\in\mathbb{C}^{n^2\times n^2}$ of $L_f(A)$, defined by $\mathrm{vec}(L_f(A,E))=K_f(A)\,\mathrm{vec}(E)$, where

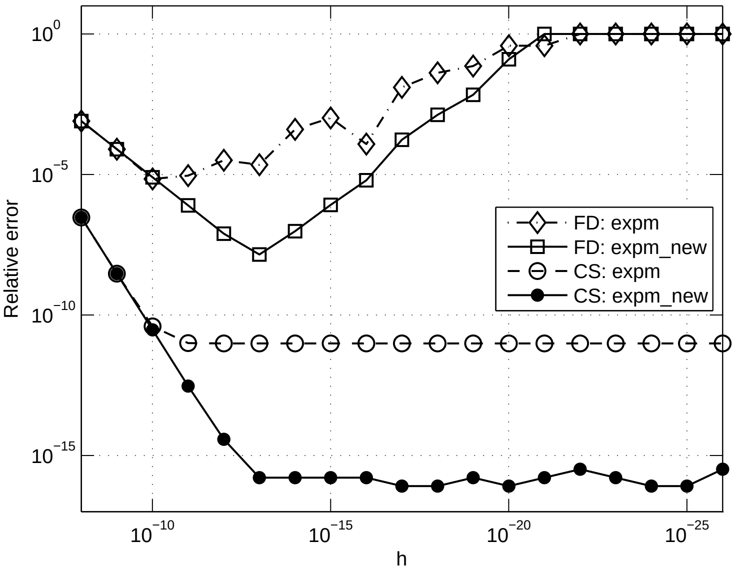

Figure 5.3: Relative errors for approximating $L_{\exp}(A, E)$ using the CS approximation and the finite difference (FD) approximation (5.1), for $h = 10^{-k}/\|A\|_1$, $k = 2\colon 21$.

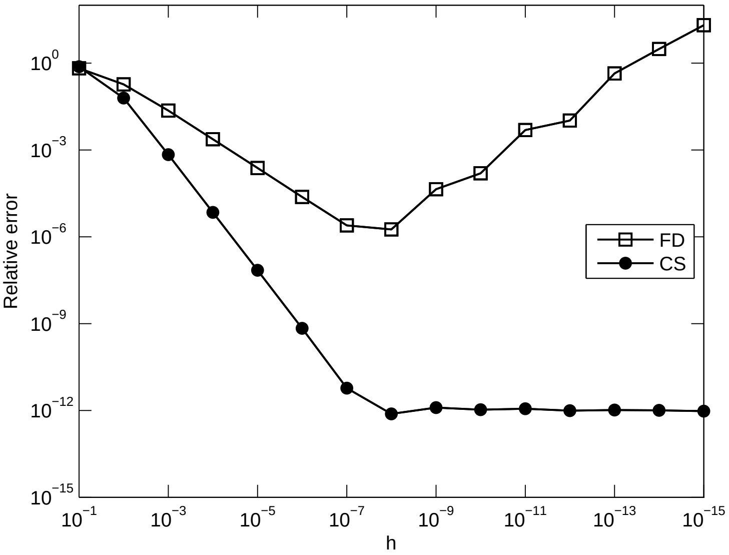

Figure 5.4: Relative errors for approximating $L_{\mathrm{sqrt}}(A, E)$ using the CS approximation and the finite difference (FD) approximation (5.1) with the product form of the Denman–Beavers iteration, for $h = 10^{-k}/\|A\|_1$, $k = 1\colon 15$.

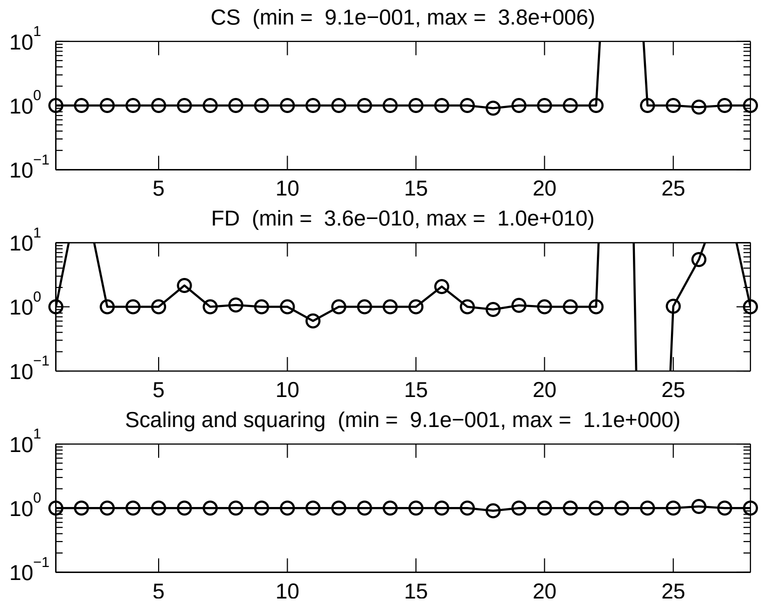

Figure 5.5: Ratios of estimate of $\|K_f(A)\|_1$ divided by true value for $f(A) = e^A$, computed using a block 1-norm estimator, where the Fréchet derivative is approximated by the CS approximation, the finite difference (FD) approximation (5.1), and Algorithm 4.6.3.

vec is the operator that stacks the columns of a matrix into one long vector [31, Chap. 3]. We will use the block 1-norm estimation algorithm of Higham and Tisseur [34], which requires the ability to form matrix products $K_f y \equiv \mathrm{vec}(L_f(A, E)$ and $K_f^T y \equiv \mathrm{vec}(L_f(A, E^T)^T)$, where $\mathrm{vec}(E) = y$ (where we are assuming $A \in \mathbb{R}^{n\times n}$ and $f : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$). We use a modified version of the function `funm_condest1` from the Matrix Function Toolbox [26], which interfaces to the MATLAB function `normest1` that implements the block 1-norm estimation algorithm. With f the exponential, evaluated by `expm`, we approximate the Fréchet derivative using three different approaches: the CS approximation, finite differences, and Algorithm 4.6.3, which is a specialized method based on scaling and squaring and Padé approximation. We take a collection of 28 real matrices from the literature on methods for e^A, which are mostly ill conditioned or badly scaled and are all of dimension 10 or less. For the finite difference approximation (5.1) we take the value $h = (u\|f(A)\|_1)^{1/2}/\|E\|_1$, which is optimal in the sense of balancing truncation error and rounding error bounds [31, Sec. 3.4]. For the CS approximation we take $h = \mathrm{tol}\|A\|_1/\|E\|_1$ with $\mathrm{tol} = u^2$; we found that for $\mathrm{tol} = u^{1/2}$ and $\mathrm{tol} = u$ the estimates were sometimes very poor on the most badly scaled problems. The exact $\|K_f(A)\|_1$ is obtained by explicitly computing $K_f(A)$ using `expm_cond` from the Matrix Function Toolbox [26]. The ratios of the estimate divided by $\|K_f(A)\|_1$ are shown in Figure 5.5. All should be at most 1, so a value larger than 1 is a sign of inaccurate Fréchet derivative approximations. The results show that the condition estimates obtained with the CS approximation are significantly more reliable than those from finite differences (one estimate of the wrong order of magnitude as opposed to four), but that neither is as reliable as when the Fréchet derivative is computed by a method specialized to the exponential.

Chapter 6

Conclusions

The propensity of the scaling and squaring method to overscaling has been known for over a decade. The new algorithm developed here, Algorithm 2.6.1, remedies this weakness in two different ways. First, it exploits triangularity, when present, to ensure that the diagonal and first off-diagonal are computed accurately during the squaring phase, benefitting all elements of the computed exponential. Second, it employs more refined truncation error bounds, based on the quantities $\|A^k\|^{1/k}$, which can be much smaller than the bounds based on $\|A\|$ that were used previously. These refined bounds enable the algorithm to produce a result that often has one or both of the advantages of being more accurate and having a lower computational cost than the original algorithm from [30] (Algorithm 2.3.1). A general conclusion of this work is that although matrix norms are sometimes described as a blunt instrument, they can extract much more information than might be thought about the behavior of a matrix power series, through the use of (estimates of) the norms of a small number of matrix powers.

A remaining open question is to understand, and ideally improve, the numerical stability of the squaring phase of the scaling and squaring method. Our treatment of the triangular case is a first step in this direction.

Our new algorithm, Algorithm 3.5.2 for the $e^A B$ problem (and its special case Algorithm 3.3.2), has a number of attractive features. Suppose, first, that $B \in \mathbb{R}^n$ is a vector. The algorithm spends most of its time computing matrix–vector products, with other computations occupying around 12% of the time. Thus it fully benefits from problems where matrix–vector products are inexpensive, or their evaluation is optimized. The algorithm is essentially direct rather than iterative, because the number of matrix–vector products is known after the initial norm estimation phase and depends only on the values $d_k = \|A^k\|_1^{1/k}$ for a few values of k (sometimes just $k = 1$). No convergence test is needed, except the test for early termination built into the evaluation of the truncated Taylor series. Our experiments demonstrate excellent numerical stability in floating point arithmetic, and this is supported (but not entirely explained) by the analysis in Section 3.4. The numerical reliability of the algorithm is emphasized by the fact that in Experiment 1 it has a better relative error performance profile than evaluation of $e^A b$ using the best current method for e^A. A particular strength of the algorithm is in the evaluation of e^{At} at multiple points on the interval of interest, because the scaling used by the algorithm naturally produces intermediate values, and the design of the algorithm ensures that when extremely dense output is required overscaling is avoided.

These benefits contrast with Krylov methods, of which we tested two particular examples. These methods are genuinely iterative, so their cost is difficult to predict and the choice of convergence test will influence the performance and reliability. They also typically require the selection or estimation of the size of the Krylov subspace. Moreover, the cost of the methods is not necessarily dominated by the matrix–vector products with A; depending on the method and the problem it can be dominated by the computations in the Arnoldi or Lanczos recurrence, and the cost of evaluating exponentials of smaller Hessenberg matrices is potentially significant.

For the case where B is a matrix with $n_0 > 1$ columns, Algorithm 3.5.2 is particularly advantageous because the logic of the algorithm is unchanged and the computational effort is now focused on products of $n \times n$ and $n \times n_0$ matrices. The Krylov codes `expv` and `phipm` must be called repeatedly on each column of B and thus cannot benefit from the greater efficiency of matrix products. An algorithm for $e^A B$ based on block Krylov subspaces might avoid this drawback, but no such algorithm is currently available.

The weakness of Algorithm 3.5.2 is its tendency for the cost to increase with increasing $\|A\|$ (though it is the d_k that actually determine the cost). This weakness is to some extent mitigated by preprocessing, but for symmetric A balancing has no effect and $d_k \approx \|A\|$ (there is equality for the 2-norm, but the algorithm uses the 1-norm). Note also that Algorithm 3.5.2 requires matrix–vector products with both A and A^* for the norm estimation. However, if A is non-Hermitian and known only implicitly and A^*x cannot be formed, we can revert to the use of $\|A\|_1$ in place of the $\alpha_p(A)$.

In summary, Algorithm 3.5.2 emerges as the best method for the $e^A B$ problem in our experiments. It has several features that make it attractive for black box use in a wide range of applications: its applicability to any A, its predictable cost after the initial norm estimation phase, its excellent numerical stability, and its ease of implementation.

We have established the notion that if a numerical algorithm for a matrix function based on rational approximation or iterative method) is available, it can be extended to an algorithm for simultaneously computing the matrix function itself and its Fréchet derivative. We extended the algorithm of Higham [30] (Algorithm 2.3.1) and the backward error analysis underling it to Algorithm 4.6.3 that calculates simultaneously the matrix exponential and its Fréchet derivative. Similarly, we extended Algorithm 2.6.1, which improves on the algorithm of Higham, to an Algorithm 4.8.2 for computing the matrix exponential and its Fréchet derivative. Both Algorithm 4.6.3 and Algorithm 4.8.2 were adapted to Algorithm 4.7.1 and Algorithm 4.8.3, respectively, that both compute the matrix exponential and return an estimate of its condition number.

The LAPACK Users' Guide states [5, p. 77] that "Our goal is to provide error bounds for most quantities computed by LAPACK." This is a desirable goal for any numerical algorithm, and in order to achieve it error analysis must be developed that yields a reasonably sharp error bound that can be efficiently estimated. For matrix function algorithms a complete error analysis is not always available, and for the forward error a bound of the form $\operatorname{cond}(f, A)u$ is the best we can expect in general. To date relatively little attention has been paid to combining evaluation of $f(A)$

with computation of the Fréchet derivative $L(A, E)$ and estimation of the condition number $\mathrm{cond}(f, A)$. We are currently applying and extending the ideas developed here to other transcendental functions such as the logarithm and the sine and cosine and will report on this work in future.

The CS approximation of Chapter 5 provides an attractive way to approximate Fréchet derivatives L_f when specialized methods for f but not L_f are available. This situation pertains, for example, to the functions $\psi_k(z) = \sum_{j=0}^{\infty} z^j/(j+k)!$, $k \geq 0$ [31, Sec. 10.7.4], related to the matrix exponential, for which efficient numerical methods are available [45], [64]. The CS approximation is trivial to implement assuming the availability of complex arithmetic. In floating point arithmetic its accuracy is not limited by the cancellation inherent in finite difference approximations, and indeed the accuracy of the computed approximation is in practice remarkably insensitive to the choice of the parameter h, as long as it is chosen small enough: typically $h \leq u^{1/2}\|A\|/\|E\|$ suffices thanks to the $O(h^2)$ approximation error.

The main weakness of the CS approximation is that it is prone to damaging cancellation when the underlying method for evaluating f employs complex arithmetic. But for many algorithms, such as those based on real polynomial and rational approximations or matrix iterations, this is not a concern.

The CS approximation is particularly attractive for use within a general purpose matrix function condition estimator. It is intended to update the function `funm_condest1` in the Matrix Function Toolbox [26] to augment the current finite difference approximation with the CS approximation, which will be the preferred option when it is applicable.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] Awad H. Al-Mohy and Nicholas J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. MIMS EPrint 2010.30, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, March 2010. 23 pp.

[5] 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.

[6] Athanasios C. Antoulas. *Approximation of Large-Scale Dynamical Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005. xxv+479 pp. ISBN 0-89871-529-6.

[7] Per Ling Bo Kågström and Charles F. Van Loan. GEMM-based level 3 BLAS: High performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 24(3):268–302, 1998.

[8] Matias Bossa, Ernesto Zacur, and Salvador Olmos. Algorithms for computing the group exponential of diffeomorphisms: Performance evaluation. In *Computer Vision and Pattern Recognition Workshops, 2008 (CVPRW '08)*, IEEE Computer Society, 2008, pages 1–8.

[9] Arthur Cayley. A memoir on the theory of matrices. *Philos. Trans. Roy. Soc. London*, 148:17–37, 1858.

[10] Tzu-Yi Chen and James W. Demmel. Balancing sparse matrices for computing eigenvalues. *Linear Algebra Appl*, 309:261–287, 2000.

[11] A. R. Collar. The first fifty years of aeroelasticity. *Aerospace (Royal Aeronautical Society Journal)*, 5:12–20, 1978.

[12] M. G. Cox and P. M. Harris. Numerical analysis for algorithm design in metrology. Software Support for Metrology Best Practice Guide No. 11, National Physical Laboratory, Teddington, UK, 2004. 75 pp.

[13] 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.

[14] Timothy A. Davis. University of Florida sparse matrix collection. `http://www.cise.ufl.edu/research/sparse/matrices/`.

[15] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. Manuscript available at `http://www.cise.ufl.edu/research/sparse/matrices/`, 2010. 28 pp.

[16] James W. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Numer. Math.*, 51:251–289, 1987.

[17] Luca Dieci and Alessandra Papini. Padé approximation for the exponential of a block triangular matrix. *Linear Algebra Appl.*, 308:183–202, 2000.

[18] Luca Dieci and Alessandra Papini. Conditioning of the exponential of a block triangular matrix. *Numer. Algorithms*, 28:137–150, 2001.

[19] J. J. Dongarra, J. J. Du Croz, I. S. Duff, and S. J. Hammarling. A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.

[20] Iain S. Duff, Michael A. Heroux, and Roldan Pozo. An overview of the Sparse Basic Linear Algebra Subprograms: The new standard from the BLAS Technical Forum. *ACM Trans. Math. Software*, 28(2):239–267, 2002.

[21] 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.

[22] Chun-Hua Guo and Nicholas J. Higham. A Schur–Newton method for the matrix pth root and its inverse. *SIAM J. Matrix Anal. Appl.*, 28(3):788–804, 2006.

[23] Gareth Hargreaves. *Topics in Matrix Computations: Stability and Efficiency of Algorithms.* PhD thesis, University of Manchester, Manchester, England, 2005. 204 pp.

[24] Gareth I. Hargreaves and Nicholas J. Higham. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms*, 40(4):383–400, 2005.

[25] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[26] Nicholas J. Higham. The Matrix Function Toolbox. `http://www.ma.man.ac.uk/~higham/mftoolbox`.

[27] Nicholas J. Higham. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Trans. Math. Software*, 16(4):352–368, 1990.

[28] Nicholas J. Higham. Stability of a method for multiplying complex matrices with three real matrix multiplications. *SIAM J. Matrix Anal. Appl.*, 13(3):681–687, 1992.

[29] 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.

[30] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[31] 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.

[32] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Rev.*, 51(4):747–764, 2009.

[33] Nicholas J. Higham and Awad H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19:159–208, 2010.

[34] 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.

[35] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[36] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. xiii+561 pp. ISBN 0-521-30586-1.

[37] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991. viii+607 pp. ISBN 0-521-30587-X.

[38] Giles Jewitt and J. Roderick McCrorie. Computing estimates of continuous time macroeconometric models on the basis of discrete data. *Computational Statistics & Data Analysis*, 49:397–416, 2005.

[39] Peter Sturdza Joaquim R. R. A. Martins and Juan J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Software*, 29(3):245–262, 2003.

[40] George A. Baker Jr. and Peter Graves-Morris. *Padé Approximants*, volume 59 of *Encyclopedia of Mathematics and Its Applications*. Second edition, Cambridge University Press, 1996. xiv+746 pp.

[41] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003. xiii+104 pp. ISBN 0-89871-546-6.

[42] Charles S. Kenney and Alan J. Laub. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.*, 10(2):191–209, 1989.

[43] Charles S. Kenney and Alan J. Laub. Polar decomposition and matrix sign function condition estimates. *SIAM J. Sci. Statist. Comput.*, 12(3):488–504, 1991.

[44] Charles S. Kenney and Alan 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.

[45] Souji Koikari. An error analysis of the modified scaling and squaring method. *Computers Math. Applic.*, 53:1293–1305, 2007.

[46] Souji Koikari. Algorithm 894: On a block Schur–Parlett algorithm for φ-functions based on the sep-inverse estimate. *ACM Trans. Math. Software*, 36 (2):Article 12, 2009.

[47] K. L. Lai and J. L. Crassidis. Extensions of the first and second complex-step derivative approximations. *J. Comput. Appl. Math.*, 219:276–293, 2008.

[48] J. N. Lyness. Numerical algorithms based on the theory of complex variable. In *Proceedings of the 1967 22nd National Conference*, ACM, New York, NY, USA, 1967, pages 125–133.

[49] J. N. Lyness and C. B. Moler. Numerical differentiation of analytic functions. *SIAM J. Numer. Anal.*, 4(2):202–210, 1967.

[50] Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The connection between the complex-step derivative approximation and algorithmic differentiation, 2001. AIAA paper AIAA-2001-0921.

[51] Roy Mathias. Evaluating the Frechet derivative of the matrix exponential. *Numer. Math.*, 63:213–226, 1992.

[52] Roy Mathias. A chain rule for matrix functions and applications. *SIAM J. Matrix Anal. Appl.*, 17(3):610–620, 1996.

[53] Borislav V. Minchev and Will M. Wright. A review of exponential integrators for first order semi-linear problems. Preprint 2/2005, Norwegian University of Science and Technology, Trondheim, Norway.

[54] 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.

[55] L. Morai and A. F. Pacheco. Algebraic approach to the radioactive decay equations. *Am. J. Phys.*, 71(7):684–686, 2003.

[56] Igor Najfeld and Timothy F. Havel. Derivatives of the matrix exponential and their computation. *Advances in Applied Mathematics*, 16:321–375, 1995.

[57] Jitse Niesen. `http://www.amsta.leeds.ac.uk/~jitse/software.html`, retrieved on February 17, 2010.

[58] Jitse Niesen and Will M. Wright. A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators. Technical report arxiv:0907.4631, 2009. 20 pp.

[59] Michael L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. xiv+104 pp. ISBN 0-89871-571-7.

[60] Michael S. Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[61] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 29(1):209–228, 1992.

[62] L. F. Shampine. Accurate numerical derivatives in MATLAB. *ACM Trans. Math. Software*, 33(4), 2007. Article 26, 17 pages.

[63] Roger B. Sidje. Expokit: A software package for computing matrix exponentials. *ACM Trans. Math. Software*, 24(1):130–156, 1998.

[64] Bård Skaflestad and Will M. Wright. The scaling and modified squaring method for matrix functions related to the exponential. *Appl. Numer. Math.*, 59:783–799, 2009.

[65] The Control and Systems Library SLICOT. `http://www.slicot.org/`.

[66] William Squire and George Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Rev.*, 40(1):110–112, 1998.

[67] Ji-guang Sun. Perturbation analysis of the matrix sign function. *Linear Algebra Appl.*, 250:177–206, 1997.

[68] Lloyd N. Trefethen, J. A. C. Weideman, and Thomas Schmelzer. Talbot quadratures and rational approximations. *BIT*, 46(3):653–670, 2006.

[69] Charles F. Van Loan. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal.*, 14(6):971–981, 1977.

[70] Charles F. Van Loan. Computing integrals involving the matrix exponential. *IEEE Trans. Automat. Control*, AC-23(3):395–404, 1978.

[71] David S. Watkins. A case where balancing is harmful. *Electron. Trans. Numer. Anal.*, 23:1–4, 2006.

[72] Daniel E. Whitney. More about similarities between Runge–Kutta and matrix exponential methods for evauating transient response. *Proc. IEEE*, 57:2053–2054, 1969.

[73] Thomas G. Wright. Eigtool. `http://www.comlab.ox.ac.uk/pseudospectra/eigtool/`.

[74] Ding Yuan and Warnick Kernan. Explicit solutions for exit-only radioactive decay chains. *J. Appl. Phys.*, 101:094907 1–12, 2007.