Arbitrary Precision Algorithms for Computing the Matrix Cosine and its Fréchet Derivative

Al-Mohy, Awad and Higham, Nicholas J. and Liu, Xiaobo

2021

MIMS EPrint: **2021.13**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

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

ISSN 1749-9097

ARBITRARY PRECISION ALGORITHMS FOR COMPUTING THE MATRIX COSINE AND ITS FRÉCHET DERIVATIVE*

AWAD H. AL-MOHY†, NICHOLAS J. HIGHAM‡, AND XIAOBO LIU‡

Abstract. Existing algorithms for computing the matrix cosine are tightly coupled to a specific precision of floating-point arithmetic for optimal efficiency so they do not conveniently extend to an arbitrary precision environment. We develop an algorithm for computing the matrix cosine that takes the unit roundoff of the working precision as input, and so works in an arbitrary precision. The algorithm employs a Taylor approximation with scaling and recovering and it can be used with a Schur decomposition or in a decomposition-free manner. We also derive a framework for computing the Fréchet derivative, construct an efficient evaluation scheme for computing the cosine and its Fréchet derivative simultaneously in arbitrary precision, and show how this scheme can be extended to compute the matrix sine, cosine, and their Fréchet derivatives all together. Numerical experiments show that the new algorithms behave in a forward stable way over a wide range of precisions. The transformation-free version of the algorithm for computing the cosine is competitive in accuracy with the state-of-the-art algorithms in double precision and surpasses existing alternatives in both speed and accuracy in working precisions higher than double.

Key words. multiprecision algorithm, multiprecision arithmetic, matrix cosine, matrix exponential, matrix function, Fréchet derivative, double angle formula, Taylor approximation, forward error analysis, MATLAB

AMS subject classifications. 15A16, 65F30, 65F60

1. Introduction.

Matrix functions has been the subject of much research because of their many applications in science and engineering. The matrix exponential is the most studied function thanks to its crucial role in representing the solutions of linear first order differential equations. The matrix sine and cosine, which can be defined for $A \in \mathbb{C}^{n\times n}$ by their Maclaurin series

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

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

where I denotes the identity matrix of order n, play an analogous role for second order differential equations. For example, the second order system

$$y''(t) + Ay(t) = g(t), \quad y(0) = y_0, \quad y'(0) = y_0', \tag{1.1}$$

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

$$y(t) = \cos(\sqrt{A}t)y_0 + (\sqrt{A})^{-1}\sin(\sqrt{A}t)y_0' + \int_0^t (\sqrt{A})^{-1}\sin\big(\sqrt{A}(t-s)\big)g(s)\mathrm{d}s,$$

*Version dated August 16, 2021.

Funding: The work of the first author was supported by the Deanship of Scientific Research at King Khalid University through Research Groups Program No. R.G.P.1/113/40. The work of the second author was supported by Engineering and Physical Sciences Research Council grant EP/P020720/1 and the Royal Society.

†Department of Mathematics, King Khalid University, P.O. Box 9004, Abha, Saudi Arabia (ahalmohy@kku.edu.sa).

‡Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk, xiaobo.liu@manchester.ac.uk).

where $\sqrt{A}$ denotes any square root of A [37]. For generalizations of the system (1.1) and other applications see [5] and the references therein.

In recent years there has been a growing interest in multiprecision algorithms for computing matrix functions. Several algorithms that work in arbitrary precision have been developed, including algorithms for the matrix exponential [8], [16] and the matrix logarithm [15], and the algorithm [23] for general matrix functions. We note that the cosine and sine of a matrix can be computed via the the matrix exponential by exploiting the matrix analogue of Euler's formula, $\mathrm{e}^{\mathrm{i}A} = \cos A + \mathrm{i}\sin A$, and this idea is implemented in the mpmath library [28], but complex arithmetic needs to be used even for a real A. The Multiprecision Computing Toolbox [31] offers functions that can evaluate in arbitrary precision the sine and cosine of a matrix using the Schur–Parlett algorithm [9]. We are not aware of any specialized algorithm for computing the matrix cosine in arbitrary precision.

The need for arbitrary precision algorithms for matrix trigonometric functions arises from their inclusion in many languages and libraries that attempt to offer arbitrary precision implementations of various functions with both scalar and matrix arguments, including the software mentioned above, as well as the Julia language [7] and Python's SymPy [30], for example. Furthermore, from the aspect of algorithm development, a reference solution computed in higher precision is required to estimate the forward error of algorithms for these matrix functions.

In this work we develop a new arbitrary precision algorithm for computing the matrix cosine. The algorithm uses a Taylor approximant to $\cos(2^{-s}X)$ in conjunction with the double angle recurrence $\cos(2X) = 2\cos^2(X) - I$, and we refer to this process as scaling and recovering. The algorithmic parameters s and the degree of the approximant are determined from a relative forward error bound for the approximant. The algorithm takes the working precision as an input argument and can compute the Fréchet derivative $L_{\cos}(A, E)$ (defined in section 5) simultaneously.

We begin in section 2 by reviewing previous work on computing the matrix cosine and explaining why existing algorithms are not suitable for arbitrary precision arithmetic. In section 3 we derive a bound on norm of the forward error of a Taylor approximant to the matrix cosine. Based on this error bound we develop an algorithm for evaluating the matrix cosine in arbitrary precision in section 4. In section 5 we derive a framework for computing $L_{\cos}(A, E)$ by Fréchet differentiating our algorithm for $\cos A$, and we construct an efficient evaluation scheme for computing $\cos A$ and $L_{\cos}(A, E)$ simultaneously. An algorithm for computing $\cos A$ and $L_{\cos}(A, E)$ in arbitrary precision is developed. We also discuss an extension of the evaluation scheme to the matrix sine and its Fréchet derivative. We then test the algorithms developed in the previous sections experimentally and compare their performance against alternative approaches in section 6. Conclusions are drawn in section 7.

Throughout the work we denote by $\|\cdot\|$ any consistent matrix norm, by $\mathbb{N}$ the set of nonnegative integers, and by $\mathbb{N}^+$ the set of positive integers. We denote by u the unit roundoff of the floating-point arithmetic.

2. Previous work. The focus in the literature has been on computing the matrix cosine rather than the matrix sine, as the sine can be obtained with a cosine algorithm by using the identity $\sin A = \cos(A - \frac{\pi}{2}I)$. The most popular and successful method for computing the cosine of a matrix is the scaling and recovering algorithm. It uses a rational or polynomial approximation to $\cos(2^{-s}X)$ in conjunction with scaling and recovering [21, Thm. 12.1]. The algorithm was first suggested by Serbin and Blalock [38], though they did not propose a concrete scheme for choosing

the algorithmic parameters.

Higham and Smith [26] develop an algorithm that scales the matrix such that $\|2^{-s}A\|_\infty \leq 1$ and employs a diagonal Padé approximant of fixed degree 8, where ad hoc analysis shows that this choice provides full normwise relative accuracy in double precision. Diagonal Padé approximants are preferred over non-diagonal ones, as symmetries in the coefficients of the numerator and denominator can be utilized for efficient evaluation of the approximant. Hargreaves and Higham [19] derive an algorithm that chooses the degree of the diagonal Padé approximant adaptively to minimize the computational cost subject to achieving a desired absolute error bound. Since then the strategy of using variable degree approximants has been widely adopted. Sastre er al. [36] propose an algorithm that uses Taylor series approximations with sharper absolute error bounds derived using ideas similar to those in [3, sect. 4]. The derivation of these algorithms is based on forward error bounds. Al-Mohy, Higham, and Relton [5] develop algorithms that are based on backward error analysis and Padé approximants to $\sin x$ and e^x, and they can compute the matrix sine and cosine separately or simultaneously. Another algorithm that can calculate the two functions simultaneously is proposed by Seydaoglu, Bader, Blanes, and Casas [39]; it chooses from some Taylor polynomial approximations of fixed degree and relies on precomputed constants. Other algorithms have been developed for computing the matrix cosine based on Taylor series [6], [34], [35], with improvements on the error bounds or the cost of evaluation of the approximating polynomials. There are also algorithms for evaluating the matrix cosine based on approximating functions other than Taylor and Padé approximants, for example, algorithms based on Bernoulli matrix polynomials [10] and Hermite matrix polynomials [11].

The algorithms mentioned above require computing symbolically in high precision certain constants that depend on the working precision, and these constants are crucial for selecting algorithmic parameters since they appear in the truncation error bounds or are the coefficients of the approximating functions. Therefore, none of these algorithms conveniently extends to an arbitrary precision environment since it is impractical to carry out this procedure when the accuracy at which the function should be evaluated is known only at run time. Hence a new approach is required for computing the matrix cosine in arbitrary precision.

3. Forward error analysis for the matrix cosine. Padé approximation has been widely adopted in algorithms, especially arbitrary precision algorithms, for evaluating matrix functions, including the matrix logarithm [15] and the matrix exponential [16]. In comparison with the exponential and logarithm functions, relatively few results are available concerning Padé approximants of the cosine function. In particular, we are not aware of a proof of existence of the Padé approximants for arbitrary degrees. Magnus and Wynn [29] give the coefficients of the Padé approximants of the scalar cosine function in terms of determinants of matrices whose entries are binomial coefficients, but these expressions are not useful for deriving a general error bound. For this reason, we employ the scaling and recovering idea and bound the relative forward error of the truncated Taylor approximant to the cosine. The techniques used in [16, sect. 3] for bounding the forward error of a Taylor approximant as an approximation to the matrix exponential do not generalize to the matrix cosine, because the terms in its Taylor expansion alternate in sign, but we can derive computable error bounds by using to the hyperbolic cosine.

Let

$$t_{2m}^c(A) := \sum_{i=0}^{m} \frac{(-1)^i}{(2i)!} A^{2i}, \quad t_{2m}^{ch}(A) := \sum_{i=0}^{m} \frac{1}{(2i)!} A^{2i} \tag{3.1}$$

denote the Taylor approximants of order $2m$ to $\cos A$ and $\cosh A$, respectively and let $B = A^2$. Then we have

$$\begin{aligned} \|\cos A - t_{2m}^c(A)\| &= \left\| \sum_{i=m+1}^{\infty} \frac{(-1)^i}{(2i)!} A^{2i} \right\| = \left\| \sum_{i=m+1}^{\infty} \frac{(-1)^i}{(2i)!} B^{i} \right\| \le \sum_{i=m+1}^{\infty} \frac{1}{(2i)!} \alpha_m(B)^i \\ &= \sum_{i=m+1}^{\infty} \frac{1}{(2i)!} \left(\sqrt{\alpha_m(B)}\right)^{2i} = \cosh\left(\sqrt{\alpha_m(B)}\right) - t_{2m}^{ch}\left(\sqrt{\alpha_m(B)}\right) \end{aligned} \tag{3.2}$$

by [3, Thm. 4.2(a)], where

$$\begin{aligned} &\alpha_m(B) \in \mathcal{A}_m(B) \\ &\qquad := \{\max\left(\|B^d\|^{1/d}, \|B^{d+1}\|^{1/(d+1)}\right) : d \in \mathbb{N}^+, d(d-1) \le m+1\}. \end{aligned} \tag{3.3}$$

For nonnormal matrices this bound can be much smaller than a simpler bound based on a single power of A, such as [21, eq. (4.9)].

We note that elements in $\mathcal{A}_m(B)$ are of the form $\|B^d\|^{1/d}$ for some $d \in \mathbb{N}^+$, and the size of $\mathcal{A}_m(B)$ depends on m. In fact, we could instead apply [3, Thm. 4.2(b)] in (3.2), as Al-Mohy does in [1, eq. (3.2)], and this would lead to exactly the same bound. Nadukandi and Higham [32] show that the use of

$$\widetilde{\alpha}_m(B) := \min\{\max\left(\|B^a\|^{1/a}, \|B^b\|^{1/b}\right) : a, b \in \mathbb{N}^+,\ \gcd(a,b) = 1,\ ab - a - b < m+1\}$$

in place of $\alpha_m(B)$ results in a more refined bound, but it requires considerably more computation, which can be undesirable in high precision, as discussed in [16, sect. 3.1]. For this reason we will choose an $\alpha_m(B)$ from $\mathcal{A}_m(B)$ defined in (3.3), but it should be noted that all the results in this section remain true with $\alpha_m(B)$ replaced by $\widetilde{\alpha}_m(B)$.

Ideally, in designing an algorithm for computing the matrix cosine we would like to use in the bound (3.2) the quantity

$$\alpha_m^{\mathtt{opt}}(B) = \min\{\, \alpha_m : \alpha_m \in \mathcal{A}_m(B) \,\},$$

in order to obtain the sharpest bounds, since these bounds are obviously increasing in $\alpha_m(B)$. However, to find $\alpha_m^{\mathtt{opt}}(B)$ we would need to search over $\mathcal{D} := \{d \in \mathbb{N}^+ : d(d-1) \le m+1\}$, and this set has $\lfloor (1+\sqrt{4m+5})/2 \rfloor$ elements. This makes computation of $\alpha_m^{\mathtt{opt}}(B)$ unpractical since the value of m can be large for an algorithm aiming for an arbitrary precision environment. More importantly, it has been observed [3] that for nonnormal matrices the sequence $\{\|B^k\|^{1/k}\}$ is typically roughly decreasing despite possible considerably nonmonotonic behavior, so it is reasonable and effective to employ the considerably cheaper approximation

$$\alpha_m^*(B) = \max\left(\|B^{d^*}\|^{1/d^*}, \|B^{d^*+1}\|^{1/(d^*+1)}\right), \quad d^* = \max_d \mathcal{D} = \left\lfloor \frac{1+\sqrt{4m+5}}{2} \right\rfloor, \tag{3.4}$$

and this strategy has been shown to be effective for computing the matrix exponential in arbitrary precision by Fasi and Higham [16].

4. A multiprecision algorithm for the matrix cosine. In this section we build a novel algorithm for computing the matrix cosine in arbitrary precision floating-point arithmetic based upon the Taylor approximant t_{2m}^c of (3.1) together with the scaling and recovering idea. There are two algorithmic parameters: m, which relates to the order of approximation, and s, the number of scalings in $X = 2^{-s}A$ (or $Y = 4^{-s}B$, as we work with $B = A^2$), to be determined in order to guarantee that

$$\|\cos X - t_{2m}^c(X)\| \lesssim u\|\cos X\|. \tag{4.1}$$

By (3.2), a sufficient condition for (4.1) to hold is

$$\cosh\left(\sqrt{\alpha_m^*(4^{-s}B)}\right) - t_{2m}^{ch}\left(\sqrt{\alpha_m^*(4^{-s}B)}\right) \lesssim u\|\cos(2^{-s}A)\|. \tag{4.2}$$

We employ the Paterson–Stockmeyer method [33], which is the customary choice in the literature, to evaluate

$$t_{2m}^c(X) = \sum_{i=0}^{m} \frac{(-1)^i}{(2i)!}X^{2i} = \sum_{i=0}^{m} \frac{(-1)^i}{(2i)!}(4^{-s}B)^i =: p_m^c(4^{-s}B), \tag{4.3}$$

which is a polynomial (in $B = A^2$) of degree m. It is important to note that, in choosing the degree m and hence the corresponding approximants, only those that maximize the approximation degree for a given number of matrix multiplications are worth considering. For evaluating polynomial approximants p_m^c to the cosine by means of the Paterson–Stockmeyer method, the sequence of optimal degrees is [14, eq. (14)]

$$\mathtt{m}_i := \left\lfloor \frac{(i+2)^2}{4} \right\rfloor, \quad i \in \mathbb{N}.$$

To evaluate the approximant $p_{\mathtt{m}_i}^c(B)$, it is known that at least the first $\nu = \lfloor\sqrt{\mathtt{m}_i}\rfloor$ powers of B will be required [18, Thm. 1.7.4]. Hence we can form the first ν powers of B immediately after the degree m is chosen, which can be used to reduce the computational cost of evaluating $\alpha_m^*(4^{-s}B)$. Note that $\|(4^{-s}B)^d\|^{1/d} = 4^{-s}\|B^d\|^{1/d}$ and thus $\alpha_m^*(4^{-s}B) = 4^{-s}\alpha_m^*(B)$, so we could compute the norm of powers of B and then perform scaling as required. More computational effort can be saved in finding $\alpha_m^*(B)$ by estimating the 1-norm of powers of B since it is enough to evaluate accurately the order of magnitude of $\alpha_m^*(B)$. We adapt the numerical scheme used by Fasi and Higham [16, Frag. 4.5] for estimating $\|B^d\|_1$, $d \in \mathbb{N}^+$. The algorithm efficiently computes B^dW using available powers of B, where $W \in \mathbb{C}^{n\times t}$, with $t \ll n$, and integrates this process with the block 1-norm estimation algorithm NORMEST1 proposed by Higham and Tisseur [27] that repeatedly computes the action of B on W, without explicitly forming any powers of B. This algorithm requires only $O(n^2)$ flops.

The technique proposed by Fasi and Higham [16, sect. 4.1] can be exploited to obtain a sharper bound at almost no extra cost, by reusing quantities computed during previous steps of the algorithm. Since the algorithm considers the approximants in nondecreasing order of cost, the value of $d^*(\mathtt{m}_i)$ in (3.4) is nondecreasing in i. Hence, in the process of seeking suitable a degree parameter we can use a variable $\alpha_{\min}$ to keep track of the smallest value of $\alpha_{\mathtt{m}_i}^*(B)$ computed up to now, and update it when a new value $\alpha_{\mathtt{m}_j}^*(B) < \alpha_{\min}$ is found for some $j > i$, and in practical calculation we use this $\alpha_{\min}$ to replace $\alpha_m^*(B)$.

On the other hand, we do not know $\|\cos(2^{-s}A)\| = \|\cos X\|$ a priori, so we could use a lower bound for the norm of $\cos X$, such as those presented in [21, Thm. 12.3]. In

fact, we can even derive a sharper bound by exploiting the result in [3, Thm. 4.2(a)]: with $Y = X^2$,

$$\|\cos X\| = \left\| I + \sum_{i=1}^{\infty} \frac{(-1)^i}{(2i)!} X^{2i} \right\| \geq 1 - \left\| \sum_{i=1}^{\infty} \frac{(-1)^i}{(2i)!} Y^i \right\|$$
$$\geq 1 - \sum_{i=1}^{\infty} \frac{\left(\sqrt{\alpha_m(Y)}\right)^{2i}}{(2i)!} = 2 - \cosh\left(\sqrt{\alpha_m(Y)}\right).$$

However, to use this bound or those in [21, Thm. 12.3] it is required that $\theta < \cosh^{-1}(2)$, for $\theta = \sqrt{\alpha_m(Y)}$ or $\theta = \sqrt{\|Y\|}$. This condition on the norm of the scaled matrix $Y = 4^{-s}A^2$ can require a very large s (when $\|A\|$ is large), which means a large number of double angle recurrence steps. This potentially rigid restriction on s is undesirable, especially for an algorithm aiming for arbitrary precision. Alternatively, an absolute bound can be used in developing the algorithm, for example [19], [26], [36], and clearly this is reasonable if $\|Y\|$ is not too large. In our algorithm we truncate the Taylor series to obtain the practical approximation

$$\cos(2^{-s}A) \approx \sum_{i=0}^{\ell} \frac{(-1)^i}{(2i)!} (2^{-s}A)^{2i} = \sum_{i=0}^{\ell} \frac{(-4^{-s})^i}{(2i)!} \mathcal{B}_i, \quad \ell = \text{length}(\mathcal{B}), \tag{4.4}$$

where $\mathcal{B} = \{I, B, B^2, \dots\}$ is an array that stores the powers of $B = A^2$, and length($\mathcal{B}$) is the number powers in $\mathcal{B}$ with positive exponents. This is the best approximation we currently have to $\cos(2^{-s}A)$. In practice, we can evaluate (4.4) in a lower precision since it suffices to obtain the correct order of magnitude of $\|\cos(2^{-s}A)\|$ in the bound (4.1), and in fact this is necessary for better efficiency considering that we have to recompute the coefficients in (4.4) when s is changed. We update $\mathcal{B}$ when it does not contain the first $\lfloor\sqrt{m}\rfloor$ powers of B, so the value of length($\mathcal{B}$) varies with the degree m. Since the estimate (4.4) uses only the powers of B that are available in $\mathcal{B}$, it requires only $O(n^2)$ flops.

The function EvalBound in Fragment 4.1 shows how the bound (4.2) can be evaluated efficiently using the techniques discussed above. We use some extra precision in forming the sum in the t_{2m}^{ch} term in the error bound (4.2) to guarantee sufficient accuracy, and we found this strategy makes no noticeable difference to the speed. In our implementation we only compute and store these scalar coefficients at run time when the order increases from $\mathtt{m}_i$ to $\mathtt{m}_{i+1}$, so each of the coefficients is calculated at most once. Within the 1-norm estimating function normest1 in EvalBound, we have used the lambda syntax from lambda calculus for an anonymous function: $\lambda x.f(x)$ denotes a function that replaces all the occurrences of x in the body of f with the value of its input argument.

For a chosen combination of s and m if the bound (4.2) is not satisfied we can either increase m from $\mathtt{m}_i$ to $\mathtt{m}_{i+1}$ to use a Taylor approximant of higher order or increment the scaling parameter s, to reduce the truncation error of approximation. Both options will increase the dominant part of the computational cost by one matrix multiplication. Although increasing s will increase the number of matrix squarings that will occur during the recovering phase of the algorithm, which is a potentially significant source of rounding errors for the algorithm, we still need to choose s such that the norm of $X = 2^{-s}A$ is sufficiently small in order for the Taylor approximation of $\cos X$ to be computed stably and accurately. On the other hand, when $\|2^{-s}A\| \gg 1$ both the actual error and the bound (3.2) can decrease extremely slowly as m

Fragment 4.1: Error bound checking for the matrix cosine.

1 **function** EvalBound($B \in \mathbb{C}^{n\times n}$, $m, s \in \mathbb{N}$)
  ▷ *Check (4.2) using elements in $\mathcal{B}$. Lines 2–3 are executed in precision u, line 10 is executed in precision $u^{1.2}$, and other lines are executed in double precision.*
2 **for** $i \leftarrow \text{length}(\mathcal{B}) + 1$ **to** $\lfloor\sqrt{m}\rfloor$ **do**
3 $\mathcal{B}_i = \mathcal{B}_{i-1}B$
4 $d^* \leftarrow \lfloor\frac{1+\sqrt{4m+5}}{2}\rfloor$
5 **if** $b_{d^*} = -\infty$ **then**
6 $b_{d^*} \leftarrow \text{normest1}(\lambda x.\text{EvalPowVec}(d^*, x))^{1/d^*}$
7 **if** $b_{d^*+1} = -\infty$ **then**
8 $b_{d^*} \leftarrow \text{normest1}(\lambda x.\text{EvalPowVec}(d^*+1, x))^{1/(d^*+1)}$
9 $\alpha_{\min} \leftarrow \min\{\max\{b_{d^*}, b_{d^*+1}\}, \alpha_{\min}\}$
10 $\delta_{\text{nxt}} \leftarrow \cosh(\sqrt{4^{-s}\alpha_{\min}}) - t^{ch}_{2\mathtt{m}_i}(\sqrt{4^{-s}\alpha_{\min}})$
11 $M \leftarrow \sum_{i=0}^{\text{length}(\mathcal{B})} \frac{(-4^{-s})^i}{(2i)!}\mathcal{B}_i$
12 $\phi \leftarrow \text{normest1}(\lambda x.Mx)$
13 **return** δ_{nxt}, ϕ

14 **function** EvalPowVec($W \in \mathbb{C}^{n\times t}$, $d \in \mathbb{N}$)
  ▷ *Compute B^dW using elements in $\mathcal{B}$.*
15 $\ell \leftarrow \text{length}(\mathcal{B})$
16 **while** $d > 0$ **do**
17 **for** $i \leftarrow 1$ **to** $\lfloor d/\ell \rfloor$ **do**
18 $W \leftarrow \mathcal{B}_\ell W$
19 $d \leftarrow d \mod \ell$
20 $\ell \leftarrow \min\{\ell - 1, d\}$
21 **return** W

increases, leading to the use of an approximant of degree much higher then necessary, which in turns results in loss of accuracy in floating-point arithmetic and unnecessary computation. It sometimes can be cheaper (and even more accurate) to perform a stronger scaling on A and use a lower order approximant.

Facing this flexibility in selecting the algorithmic parameters, algorithms aiming for a fixed precision environment usually choose to consider the approximants only up to a certain order, or set a scaling threshold $\eta > 0$ and keep increasing s until $\|2^{-s}A\| \le \eta$ is satisfied. The multiprecision algorithm for the matrix exponential [16] determines both parameters at run time by monitoring the decay rate of the bound on the truncation error of the approximant as m increases, and this heuristic proves to be effective. Let us denote the truncation error bound associated with Taylor approximant of order $2\mathtt{m}_i$ of this algorithm by

$$\delta_i = \cosh\left(\sqrt{\alpha^*_{\mathtt{m}_i}(4^{-s}B)}\right) - t^{ch}_{2\mathtt{m}_i}\left(\sqrt{\alpha^*_{\mathtt{m}_i}(4^{-s}B)}\right). \tag{4.5}$$

In the algorithm we increment s when $\delta_{i-1} < \delta_i^k$, $k \in \mathbb{N}^+$, that is, when the bound on the absolute error does not decay at least at the order k as m increases from $\mathtt{m}_{i-1}$ to $\mathtt{m}_i$. We found in practice that $k = 3$ is the best choice for accuracy.

Fragment 4.2: Modified Paterson–Stockmeyer algorithm for the cosine.

function PSEvalCos($B \in \mathbb{C}^{n\times n}$, $m, s \in \mathbb{N}$)
▷ *Evaluate* $\sum_{i=0}^{m} c_i(4^{-s}B)^i$ *using elements of* $\mathcal{B}$.
for $i \leftarrow 0$ **to** m **do**
  $c_i \leftarrow (-1)^i/(2i)!$
$\nu \leftarrow \lfloor\sqrt{m}\rfloor$
$\mu \leftarrow \lfloor m/\nu \rfloor$
for $i \leftarrow \mathrm{length}(\mathcal{B}) + 1$ **to** ν **do**
  $\mathcal{B}_i \leftarrow \mathcal{B}_{i-1}B$
$C \leftarrow \sum_{j=0}^{m-\mu\nu} c_{\mu\nu+j}4^{-sj}\mathcal{B}_j$
for $i \leftarrow \mu - 1$ **down to** 0 **do**
  $C \leftarrow 4^{-s\nu}C\mathcal{B}_\nu + \sum_{j=0}^{\nu-1} c_{\nu i+j}4^{-sj}\mathcal{B}_j$
return C

Fragment 4.3: Recomputation of the diagonals.

function RecompDiags($A, C \in \mathbb{C}^{n\times n}$)
▷ *Compute main diagonal and first superdiagonal of* $C \approx \cos(A)$ *for upper triangular or real upper quasitriangular* A.
for $i = 1$ **to** n **do**
  if $i = n - 1$ **or** $i \le n - 2$ **and** $a_{i+2,i+1} = 0$ **then**
    if $a_{i+1,i} = 0$ **then**
      Recompute c_{ii}, $c_{i,i+1}$, $c_{i+1,i+1}$ using [5, eqs. (3.1), (3.3)].
    else
      Recompute c_{ii}, $c_{i,i+1}$, $c_{i+1,i}$, $c_{i+1,i+1}$ using [5, eq. (3.6)].
    $i \leftarrow i + 1$
  else
    $c_{ii} \leftarrow \cos a_{ii}$
return C

Once a combination of Taylor approximant order m and scaling parameter s is found, the algorithm computes an approximation to $\cos(2^{-s}A)$ by evaluating the polynomial $p_m^c(4^{-s}B)$ using the modified Paterson–Stockmeyer method PSEvalCos given in Fragment 4.2, and finally recovers $\cos A$ by applying s steps of the double angle recurrence. If A is upper quasi-triangular, then in order to reduce the rounding errors introduced during the recovering phase and improve accuracy of the final result, the algorithm recomputes the diagonal and first superdiagonal of the intermediate matrices in the recovering stage from the elements of A, as discussed in [5], and this is accomplished by RecompDiags in Fragment 4.3. The algorithm can optionally make use of preprocessing (and postprocessing) techniques as discussed in [19], [26].

We now present the complete precision-independent scaling and recovering algorithm for the matrix cosine, which is given in Algorithm 4.4. In addition to the matrix $A \in \mathbb{C}^{n\times n}$, the algorithm takes the following input arguments.

- The arbitrary precision floating-point parameter $u > 0$ that specifies the unit roundoff of the working precision of the algorithm.

Algorithm 4.4: Multiprecision algorithm for the matrix cosine.

Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes an approximation C to $\cos A$ in floating-point arithmetic with unit roundoff u using a scaling and recovering method based on Taylor approximants. The pseudocode of EvalBound is given in Fragment 4.1, that of PSEvalCos in Fragment 4.2, and that of RecompDiags in Fragment 4.3. The function isSchurForm returns true if A is upper triangular or real and quasi upper triangular, and otherwise false.

```
B ← A^2
B_0 ← I
B_1 ← B
α_min ← ∞
δ_pre ← ∞
b ← [−∞, −∞, ...]
s ← 0
i ← 1
[δ_nxt, φ] ← EvalBound(B, m_i, s)
while δ_nxt > uφ and i < N do
    if δ_pre < δ_nxt^k then
        s ← s + 1
    else
        i ← i + 1
    δ_pre ← δ_nxt
    [δ_nxt, φ] ← EvalBound(B, m_i, s)
C ← PSEvalCos(B, m_i, s)
if isSchurForm(A) then
    C ← RecompDiags(2^{-s}A, C)
for j ← 1 to s do
    C ← 2C^2 − I
    if isSchurForm(A) then
        C ← RecompDiags(2^{-s+j}A, C)
return C
```

- The positive integer $m_{\max}$ determines the maximum order of the approximants p_m^c in (4.3) that the algorithm can consider. The algorithm will try the orders $m = \mathtt{m}_i$ ascendingly for $i = 1\colon N$ such that $\mathtt{m}_N \le m_{\max} < \mathtt{m}_{N+1}$.

In the algorithm the variables $\mathcal{B}$, b, and $\alpha_{\min}$ are assumed to be available within all the code fragments (that is, their scope is global in the codes). We use the notation $[x, x, \dots]$ to denote a vector whose elements are all initialized to x and whose length is unimportant, and such a vector b is defined to store the approximated values of $\|B^d\|_1^{1/d}$, $d \in \mathbb{N}^+$, so the 1-norm of each power of B is estimated at most once.

Overall, Algorithm 4.4 requires about $2\sqrt{\mathtt{m}_i} + s$ matrix multiplications, or a total of $(4\sqrt{\mathtt{m}_i} + 2s)n^3$ flops in the highest order in precision u, where $2\sqrt{\mathtt{m}_i}$ multiplications are for forming the required powers of B and evaluating the polynomial $p_m^c(4^{-s}B)$, and s multiplications are for performing the final recovering phase.

4.1. Schur variant. If A is normal ($A^*A = AA^*$) and a multiprecision implementation of the QR algorithm [17, sect. 7.5] is available, then we should simply diagonalize A in precision u to obtain $A = QDQ^*$ with Q unitary and D diagonal and the compute $\cos A = Q\cos(D)Q^*$. More generally, for nonnormal A a (real) Schur decomposition can be computed before invoking our multiprecision algorithm. More specifically, we compute $A = QTQ^*$, where Q and T are, respectively, unitary and upper triangular if A has complex entries and orthogonal and upper quasi-triangular if A has real entries; then we compute $\cos A = Q\cos(T)Q^*$. This Schur variant of the algorithm requires $\big(28 + (4\sqrt{\mathtt{m}_i} + 2s)/3\big)n^3$ flops in precision u.

5. Computing the Fréchet derivative. The Fréchet derivative of a matrix function f at $A \in \mathbb{C}^{n\times n}$ is a linear operator $L_f(A,\cdot)$ satisfying

$$f(A+E) = f(A) + L_f(A,E) = o(\|E\|)$$

for all $E \in \mathbb{C}^{n\times n}$. It appears in an expression for the condition: number [21, sect. 3.1]

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

where

$$\|L_f(A)\| := \max_{G\neq 0}\frac{\|L_f(A,G)\|}{\|G\|}.$$

The condition number measures the first order sensitivity of $f(A)$ to small perturbations in A.

Recall that the truncation error for a Taylor approximant of degree $2m$ to $\cos X$ is

$$\cos X - t^c_{2m}(X) = \sum_{i=m+1}^{\infty} c_i X^{2i}, \quad c_i := \frac{(-1)^i}{(2i)!}. \tag{5.1}$$

Fréchet differentiating both sides of (5.1) at $X = 2^{-s}A$ in the direction $E_s := 2^{-s}E$ gives the truncation error for an approximation to the Fréchet derivative:

$$L_{\cos}(X,E_s) - L_{t^c_{2m}}(X,E_s) = \sum_{i=m+1}^{\infty} c_i L_{x^{2i}}(X,E_s). \tag{5.2}$$

From (5.2) we can approximate $L_{\cos}(X,E_s)$ by $L_{t^c_{2m}}(X,E_s)$ with a controllable truncation error. We will discuss in detail the computation of $L_{t^c_{2m}}(X,E_s)$, the Fréchet derivative of a power series, in the next subsection.

Now we derive the basic framework for computing $\cos A$ and $L_{\cos}(A,E)$ simultaneously given that the approximated values of $\cos X \equiv \cos(2^{-s}A)$ and $L_{\cos}(X,E_s) \equiv L_{\cos}(2^{-s}A, 2^{-s}E)$ are available. Fréchet differentiating the double angle formula $\cos(2A) = 2\cos^2(A) - I$ and employing the chain rule, we have the relation

$$L_{\cos}(2A,2E) = L_{2x^2-1}\big(\cos A, L_{\cos}(A,E)\big).$$

Then using the linearity of the Fréchet derivative and the sum and product rules [21, Sec. 3.2], we obtain

$$L_{\cos}(2A,2E) = 2\big(\cos A L_{\cos}(A,E) + L_{\cos}(A,E)\cos A\big).$$

Using this relation we can construct the following recurrence relation, which yields $C_0 := \cos A$ and $L_0 := L_{\cos}(A, E)$ simultaneously:

$$\left.\begin{aligned} L_s &= L_{\cos}(2^{-s}A, 2^{-s}E),\\ C_s &= \cos(2^{-s}A),\\ L_{k-1} &= 2(C_k L_k + L_k C_k)\\ C_{k-1} &= 2C_k^2 - I \end{aligned}\right\} k = s:-1:1.$$

5.1. Error analysis and evaluation scheme. We can derive an error bound for the approximation $L_{\cos}(X, E_s) \approx L_{t_{2m}^c}(X, E_s)$. Taking norms on both sides of (5.2) gives

$$\begin{aligned} \|L_{\cos}(X,E_s) - L_{t_{2m}^c}(X,E_s)\| &\le \sum_{i=m+1}^{\infty} 2i|c_i|\|E_s\|\|X\|^{2i-1} = \|E_s\| \sum_{i=m}^{\infty} \frac{\|X\|^{2i+1}}{(2i+1)!}\\ &= \|E_s\|\Big(\sinh(\|X\|) - \sum_{i=0}^{m-1} \frac{\|X\|^{2i+1}}{(2i+1)!}\Big), \end{aligned} \tag{5.3}$$

where we have used the result $\|L_{x^i}(X, E_s)\| \le i\|E_s\|\|X\|^{i-1}$ [2, Thm. 3.2]. One advantage of this forward error bound for the Fréchet derivative of t_{2m}^c is that it can be used in a multiprecision environment. Note that the error bound is based on $\|X\|$, whereas the error bound for t_{2m}^c itself is based on $\alpha_m^*(X^2)$. Since the bound based on $\|X\|$ can be arbitrarily weak we will base our algorithm for computing $\cos A$ and $L_{\cos}(A, E)$ on the α_m-based bound (4.2). We will test experimentally whether this produces an accurate Fréchet derivative. A similar situation holds in the works [4] for the matrix logarithm and [22] for the matrix factional powers, where the algorithms which aim for double precision only are based on backward α_m-based error bounds for the functions themselves.

Now we derive an evaluation scheme for computing $\cos(X)$ and $L_{\cos}(X, E_s)$ in a way that reuses matrix operations from the computation of $\cos(X)$ in the computation of $L_{\cos}(X, E_s)$. Fréchet differentiating both sides of $t_{2m}^c(X) = p_m^c(Y)$ from (4.3), where $Y = X^2$, at X in direction E_s and using the chain rule, we have

$$L_{t_{2m}^c}(X, E_s) = L_{p_m^c}\big(X^2, L_{x^2}(X, E_s)\big) = L_{p_m^c}(Y, XE_s + E_sX).$$

Recall that $p_m^c(Y)$ is evaluated by the Paterson–Stockmeyer method. To be more specific, we rewrite the polynomial as

$$p_m^c(Y) = \sum_{i=0}^{\mu} Z_i (Y^\nu)^i, \quad \mu = \lfloor m/\nu \rfloor, \tag{5.4}$$

where

$$Z_i = \begin{cases} \sum_{j=0}^{\nu-1} c_{\nu i+j} Y^j, & i = 0, \dots, \mu-1,\\ \sum_{j=0}^{m-\mu\nu} c_{\nu i+j} Y^j, & i = \mu. \end{cases}$$

Note that the powers of $Y = 4^{-s}B$ up to the νth are available by ν matrix scalings given that we have formed those powers of B. Fréchet differentiating both sides of (5.4) at Y in direction $\widetilde{E}_s := XE_s + E_sX$ and employing the product rule, we have

$$L_{p_m^c}(Y, \widetilde{E}_s) = \sum_{i=0}^{\mu} L_{z_i}(Y, \widetilde{E}_s)(Y^\nu)^i + \sum_{i=1}^{\mu} Z_i M_{\nu i},$$

Fragment 5.1: Modified Paterson–Stockmeyer scheme for the matrix cosine and its Fréchet derivative.

function PSEvalCosLc($A, B, E \in \mathbb{C}^{n\times n}$, $m, s \in \mathbb{N}$)
▷ *Compute simultaneously* $C \approx \cos(2^{-s}A)$ *and* $L \approx L_{\cos}(2^{-s}A, 2^{-s}E)$.
for $i = 0$ **to** m **do**
  $c_i \leftarrow (-1)^i/(2i)!$
$\nu \leftarrow \lfloor\sqrt{m}\rfloor$
$\mu \leftarrow \lfloor m/\nu\rfloor$
for $i \leftarrow \mathrm{length}(\mathcal{B}) + 1$ **to** ν **do**
  $\mathcal{B}_i \leftarrow \mathcal{B}_{i-1}B$
for $i \leftarrow 0$ **to** $\mu - 1$ **do**
  $\mathcal{Z}_i \leftarrow \sum_{j=0}^{\nu-1} c_{\nu i+j}4^{-sj}\mathcal{B}_j$
$\mathcal{Z}_\mu \leftarrow \sum_{j=0}^{m-\mu\nu} c_{\nu i+j}4^{-sj}\mathcal{B}_j$
$\mathcal{M}_1 \leftarrow 4^{-s}(AE + EA)$
for $j \leftarrow 2$ **to** ν **do**
  $\mathcal{M}_j \leftarrow 4^{-s}\mathcal{M}_{j-1}B + 4^{-s(j-1)}\mathcal{B}_{j-1}\mathcal{M}_1$
$\mathcal{N}_1 \leftarrow \mathcal{M}_\nu$
$P \leftarrow \mathcal{N}_1$
for $i \leftarrow 2$ **to** μ **do**
  $P \leftarrow 4^{-s\nu}\mathcal{B}_\nu P$
  $\mathcal{N}_i \leftarrow 4^{-s\nu}\mathcal{N}_{i-1}\mathcal{B}_\nu + P$
$C \leftarrow \mathcal{Z}_\mu$
$L \leftarrow \sum_{j=1}^{m-\mu\nu} c_{\nu i+j}\mathcal{M}_j$
for $i \leftarrow \mu - 1$ **down to** 0 **do**
  $C \leftarrow 4^{-s\nu}C\mathcal{B}_\nu + \mathcal{Z}_i$
  $L \leftarrow 4^{-s\nu}L\mathcal{B}_\nu + \sum_{j=1}^{\nu-1} c_{\nu i+j}\mathcal{M}_j$
$L \leftarrow L + \sum_{i=1}^{\mu} \mathcal{Z}_i\mathcal{N}_i$
return C, L

where

$$L_{z_i}(Y, \widetilde{E}_s) = \begin{cases} \sum_{j=1}^{\nu-1} c_{\nu i+j}M_j, & i = 0, \dots, \mu - 1, \\ \sum_{j=1}^{m-\mu\nu} c_{\nu i+j}M_j, & i = \mu, \end{cases}$$

and $M_j := L_{y^i}(Y, \widetilde{E}_s)$ satisfies the recurrence relation

$$M_j = M_{\ell_1}Y^{\ell_2} + Y^{\ell_1}M_{\ell_2}, \quad M_1 = \widetilde{E}_s, \tag{5.5}$$

where $j = \ell_1 + \ell_2$ with positive integers ℓ_1 and ℓ_2 [2, Thm. 3.2]. Hence, it is efficient to compute explicitly and store Z_i in computing $p_m^c(Y) \approx \cos(2^{-s}A)$ as we can reuse these coefficient matrices for computing $L_{p_m^c}(Y, \widetilde{E}_s)$. In addition, M_j for $j = 1, 2, \dots, \nu - 1$ and $j = \nu, 2\nu, \dots, \mu\nu$ are needed. Using (5.5) we can compute

$$\begin{aligned} M_j &= M_{j-1}Y + Y^{j-1}M_1, && j = 2 : \nu, \\ M_{i\nu} &= M_{(i-1)\nu}Y^\nu + Y^{(i-1)\nu}M_\nu, && i = 2 : \mu, \end{aligned} \tag{5.6}$$

where all the required powers of Y are available if both the right-hand sides of (5.4) and (5.6) are evaluated via explicit powers. However, we found in practice that

Algorithm 5.2: Multiprecision algorithm for the matrix cosine and its Fréchet derivative.

Given $A \in \mathbb{C}^{n\times n}$ and $E \in \mathbb{C}^{n\times n}$ this algorithm computes simultaneously $C \approx \cos A$ and $L \approx L_{\cos}(A,E)$ in floating-point arithmetic with unit roundoff u using a scaling and recovering method based on Taylor approximants. The pseudocode of PSEvalCosLc is given in Fragment 5.1, and that of RecompDiags in Fragment 4.3.

```
1  Execute lines 1–16 in Algorithm 4.4.
2  [C, L] ← PSEvalCosLc(A, B, E, m_i, s)
3  if isSchurForm(A) then
4      C ← RecompDiags(2^{-s}A, C)
5  for j ← 1 to s do
6      L ← 2(CL + LC)
7      C ← 2C^2 − I
8      if isSchurForm(A) then
9          C ← RecompDiags(2^{-s+j}A, C)
10 return C, L
```

evaluating the right-hand side of (5.4) via explicit powers produces a less accurate approximation for $\cos(2^{-s}A)$ than using Horner's method. We hence use Horner's method and form the extra powers $Y^{i\nu}$, $i = 2,\dots,\mu-1$ implicitly when required in (5.6).

We summarize in Fragment 5.1 our scheme for computing simultaneously $\cos(2^{-s}A)$ and $L_{\cos}(2^{-s}A, 2^{-s}E)$ where we have introduced the arrays $\mathcal{Z}_i = Z_i$, $i = 0,\dots,\mu$, $\mathcal{M}_j = M_j$, $j = 1,\dots,\nu$, and $\mathcal{N}_i = M_{i\nu}$, $i = 1,\dots,\mu$.

Exploiting Fragment 5.1, we obtain Algorithm 5.2, the overall algorithm for computing $\cos A$ and $L_{\cos}(A,E)$ simultaneously. The total cost of Algorithm 5.2 in precision u in the highest order is the $(4\sqrt{\mathtt{m}_i}+2s)n^3$ flops cost of Algorithm 4.4 plus the extra cost for computing $L_{\cos}(A,E)$, which consists of about $6\sqrt{\mathtt{m}_i}$ matrix multiplications for computing the required coefficient matrices M_j and forming the approximated $L_{\cos}(2^{-s}A, 2^{-s}E)$, and $2s$ matrix multiplications in the recovering recurrence for $L_{\cos}(A,E)$, namely an extra cost of $(12\sqrt{\mathtt{m}_i}+4s)n^3$ flops. This algorithm also requires about $3\sqrt{\mathtt{m}_i}n^2$ additional memory locations for the storage of the Z_i and M_i for the computation of $L_{\cos}(A,E)$.

If a Schur decomposition $T = Q^*AQ$ is computed before invoking Algorithm 5.2, then for the Fréchet derivative we need apply to the direction E the same transformation and undo the transformation at the end [21, Prob. 3.2], arriving at $L_{\cos}(A,E) = QL_{\cos}(T, Q^*EQ)Q^*$. If a real Schur decomposition is computed and E is full, the Schur variant of Algorithm 5.2 requires an extra cost of $\big(8+(12\sqrt{\mathtt{m}_i}+4s)/2\big)n^3$ flops in precision u for computing the Fréchet derivative. For normal A we can simply employ an explicit formula obtained from the Daleckiĭ–Kreĭn theorem [21, Thm. 3.11] for computing the Fréchet derivative.

In some situations, such as in condition estimation, when several Fréchet derivatives $L_{\cos}(A,E)$ are needed at a fixed A and different direction we need only compute the parameters s and m once since they depend only on A.

5.2. Extension to the sine and its Fréchet derivative. It is straightforward to bound the truncation error of a Taylor approximant to the matrix sine function in a similar way to (3.2), by employing the hyperbolic sine. We have

$$\|\sin A - t^s_{2m+1}(A)\| \le \|A\| \left\| \sum_{i=m+1}^{\infty} \frac{(-1)^i}{(2i+1)!} B^i \right\| \le \frac{\|A\|}{\sqrt{\alpha_m(B)}} \sum_{i=m+1}^{\infty} \frac{\left(\sqrt{\alpha_m(B)}\right)^{2i+1}}{(2i+1)!}$$

$$= \frac{\|A\|}{\sqrt{\alpha_m(B)}} \Big(\sinh\left(\sqrt{\alpha_m(B)}\right) - t^{sh}_{2m+1}\left(\sqrt{\alpha_m(B)}\right) \Big), \tag{5.7}$$

where

$$t^s_{2m+1}(A) := \sum_{i=0}^{m} \frac{(-1)^i}{(2i+1)!} A^{2i+1}, \quad t^{sh}_{2m+1}(A) := \sum_{i=0}^{m} \frac{1}{(2i+1)!} A^{2i+1} \tag{5.8}$$

are the Taylor approximants of order $2m+1$ to $\sin A$ and $\sinh A$, respectively. Based on the t^s_{2m+1} of (5.8) together with the triple angle recurrence $\sin(3A) = 3\sin A - 4\sin^3(A)$ we can design a multiprecision algorithm for the matrix sine similarly to that in section 4. Moreover, an algorithm for computing the matrix sine and cosine at the same time can be easily developed exploiting the idea in [5], where computational savings are possible by reusing the powers of $B = A^2$ in the matrix array $\mathcal{B}$.

On the other hand, we can evaluate the matrix sine and its Fréchet derivative simultaneously without computing $\cos A$. Fréchet differentiating the triple angle formula $\sin(3A) = \sin A\big(3I - 4\sin^2(A)\big)$ and employing the product rule, we arrive at

$$\begin{aligned} L_{\sin}(3A, 3E) = {} & L_{\sin}(A, E)\big(3I - 4\sin^2(A)\big) \\ & - 4\sin A\big(\sin A L_{\sin}(A, E) + L_{\sin}(A, E)\sin A\big). \end{aligned} \tag{5.9}$$

If we scale A by 3^s for some $s \in \mathbb{N}$, using (5.9) we obtain the following relation which produces $S_0 := \sin A$ and $\widetilde{L}_0 := L_{\sin}(A, E)$ simultaneously:

$$\left.\begin{aligned} \widetilde{L}_s &= L_{\sin}(3^{-s}A, 3^{-s}E), \quad S_s = \sin(3^{-s}A), \\ \widetilde{L}_{k-1} &= \widetilde{L}_k(3I - 4S_k^2) - 4S_k(S_k\widetilde{L}_k + \widetilde{L}_k S_k) \\ S_{k-1} &= S_k(3I - 4S_k^2) \end{aligned}\right\} k = s : -1 : 1,$$

where S_k can be evaluated by a Taylor approximant with truncation error bounded above by (5.7), and an approximated $\widetilde{L}_k$ is obtainable by Fréchet differentiating S_k in the direction $3^{-s}E$.

Ultimately, it is possible to construct an algorithm to compute efficiently the matrix cosine and sine functions and their Fréchet derivatives all together. Fréchet differentiating both sides of the double angle formulae

$$\cos(2A) = I - 2\sin^2(A), \quad \sin(2A) = 2\sin A\cos A$$

gives

$$\begin{aligned} L_{\cos}(2A, 2E) &= -2\big(\sin A L_{\sin}(A, E) + L_{\sin}(A, E)\sin A\big), \\ L_{\sin}(2A, 2E) &= 2\big(\sin A L_{\cos}(A, E) + L_{\sin}(A, E)\cos A\big), \end{aligned}$$

from which we can obtain the following recurrence for computing $C_0 = \cos A$, $S_0 = \sin A$, $L_0 = L_{\cos}(A, E)$, and $\widetilde{L}_0 = L_{\sin}(A, E)$ all together.

$$\begin{aligned} &L_s = L_{\sin}(2^{-s}A, 2^{-s}E), \quad \widetilde{L}_s = L_{\sin}(2^{-s}A, 2^{-s}E), \\ &C_s = \cos(2^{-s}A), \quad S_s = \sin(2^{-s}A), \\ &\left.\begin{aligned} L_{k-1} &= -2(S_k\widetilde{L}_k + \widetilde{L}_k S_k) \\ \widetilde{L}_{k-1} &= 2(S_k L_k + \widetilde{L}_k C_k) \\ S_{k-1} &= 2S_k C_k \\ C_{k-1} &= I - 2S_k^2 \end{aligned}\right\} k = s:-1:1. \end{aligned}$$

6. Numerical experiments. All our experiments are performed using the 64-bit version of MATLAB 2021a on a laptop equipped with an Intel i7-6700HQ processor running at 2.60GHz and with 16GB of RAM. The code uses the Advanpix Multiprecision Computing Toolbox (version 4.8.3.14463) [31], which allows the user to specify the number of *decimal* digits d of working precision by using the command `mp.Digits(d)`.

The test matrices, whose size ranges between 4 and 41, are nonnormal and are selected from Anymatrix [24] and the literature of matrix functions [5], [16], [26]; those from the matrix function literature are collected in an Anymatrix group that is available from https://github.com/Xiaobo-Liu/matrices-mp-cosm. Normal matrices are excluded since they can be easily handled by diagonalization, as we have discussed in the previous sections. Most of test matrices have only real elements and are set to be of size 16×16. To examine the algorithms for complex matrices we also test the above matrices multiplied by the imaginary unit i. In total 198 matrices are used in the experiments, and we denote by $\mathcal{F}$ the set containing these matrices. The MATLAB code for our algorithms and experiments is available from https://github.com/Xiaobo-Liu/mp-cosm.

We compare the following codes for computing the cosine. The first three codes are for double precision only, and are used to test whether our algorithm is competitive in double precision.

- `cosm`, the algorithm by Al-Mohy, Higham, and Relton [5, Alg. 4.1], and is intended for double precision only.
- `cosm_tay`, the algorithm by Sastre et al. [34], which uses the scaling and recovering method based on truncated Taylor series, and is intended for double precision only.
- `cosm_pol`, the algorithm by Sastre et al. [35], which uses Taylor polynomial approximations of fixed degree with precomputed coefficients, and is intended for double precision only.

The next four codes are for arbitrary precision.

- `cosm_adv`, the (overloaded) `cosm` function provided by the Advanpix Multiprecision Computing Toolbox [31].
- `cosm_exp`, the algorithm [21, Alg. 12.7] that computes the cosine via the identity involving the exponential

$$\cos A = \begin{cases} \mathrm{Re}(\mathrm{e}^{\mathrm{i}A}), & \text{if } A \text{ is real,} \\ \frac{1}{2}(\mathrm{e}^{\mathrm{i}A} + \mathrm{e}^{-\mathrm{i}A}), & \text{if } A \text{ is complex,} \end{cases} \tag{6.1}$$

  where the exponential is computed by `expm` [3] and the multiprecision algorithm for the matrix exponential [16], respectively, in double precision and other precisions.

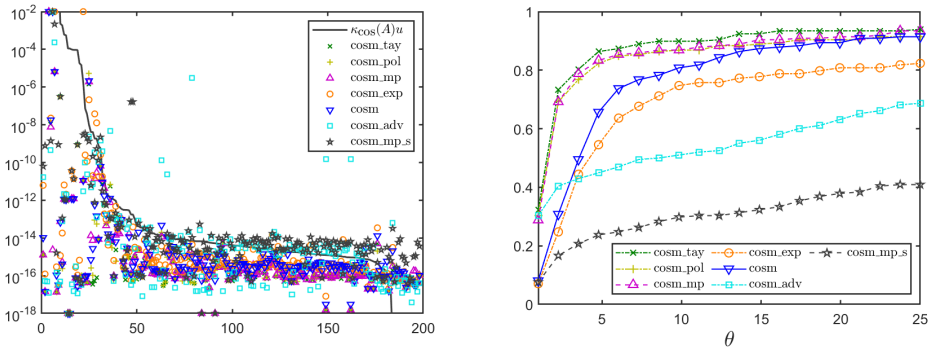

Fig. 6.1: Left: forward error of the algorithms on the matrices in $\mathcal{F}$ in double precision, where the solid line is $\kappa_{\cos}(A)u$. Right: corresponding performance profiles.

- `cosm_mp`, our implementation of Algorithm 4.4 with $m_{\max} = 500$.
- `cosm_mp_s`, our implementation of the Schur variant of Algorithm 4.4 with $m_{\max} = 500$ (the real Schur decomposition is used where possible).

We found in practice that the preprocessing techniques in general made little difference to accuracy of our algorithm (this is also found in [19], for example). Therefore, we did not perform preprocessing in the tests. We also compared `cosm_mp` with its counterpart based on an absolute error bound and found that the former is faster and more accurate in practice. In our implementation of Algorithm 4.4 we set $k = 3$, which controls the switch between increasing m and s as the truncation error decays.

We assess the quality of a computed solution $\widetilde{X}$ by an algorithm running with d digits of precision in terms of the 1-norm relative forward error $\|X - \widetilde{X}\|_1/\|X\|_1$, where the reference solution X is computed in $2d$ digits of precision using `cosm_mp` with $m_{\max} = 2500$. We gauge the forward stability of the algorithms by comparing the forward error with $\kappa_{\cos}(A)u$, where $\kappa_{\cos}(A)$ is the 1-norm condition number [21, Chap. 3] of the matrix cosine of A. We estimate it in double precision by applying the `funm_condest1` function provided by the Matrix Function Toolbox [20] to `cosm`.

To improve the plots of forward error, we map any errors outside the displayed range onto the nearest edge (top or bottom) of the plot. We also present the results in the form of performance profiles [13], and use the technique of Dingle and Higham [12] to rescale errors smaller than u.

6.1. Accuracy of the computed cosine in double precision. Our first experiments compare the accuracy of `cosm_mp` and `cosm_mp_s` with `cosm`, `cosm_exp`, `cosm_tay`, `cosm_pol`, and `cosm_adv` in IEEE double precision, with `cosm_adv` running with 16 decimal digits of precision simulated by Advanpix [31].

Figure 6.1 presents the comparison in accuracy between the algorithms on the test matrices, sorted by decreasing condition number. In the performance profiles, the y-coordinates of a given method represents the frequency of matrices for which its relative error is within a factor θ of the error of the algorithm that produces the most accurate result. We observe that our implementation of `cosm_mp` in double precision is competitive in accuracy with the most accurate algorithms that are optimized for double precision. We also note that among the algorithms `cosm_adv` is overall the least accurate and can be unstable, as it sometimes provides a forward error far above

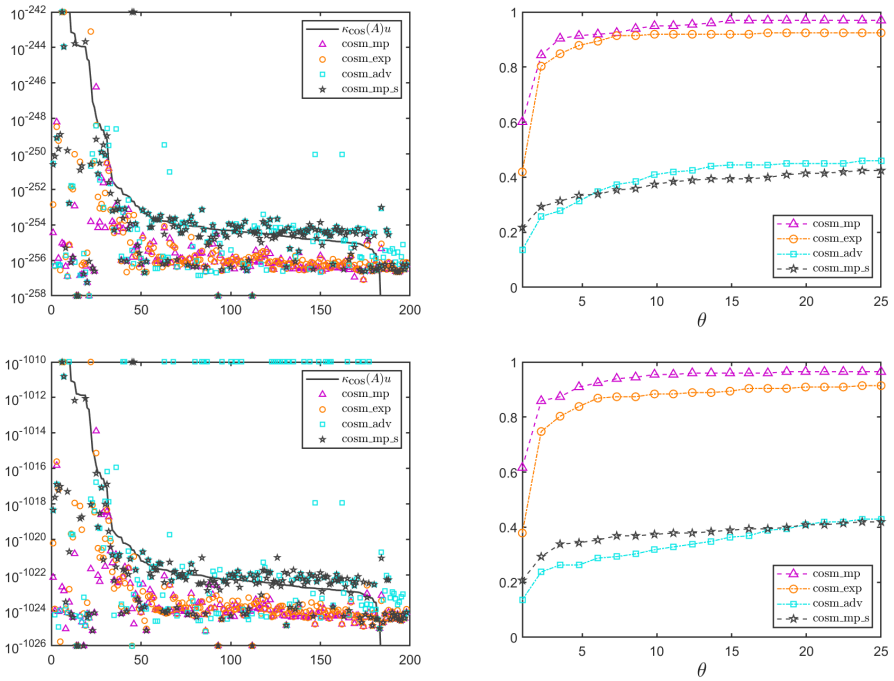

Fig. 6.2: Left: forward error of the algorithms on the matrices in $\mathcal{F}$ in d digits of precision, where the solid line is $\kappa_{\cos}(A)u$. Right: corresponding performance profiles. Top: $d = 256$. Bottom: $d = 1024$.

$\kappa_{\cos}(A)u$.

6.2. Accuracy in higher precision. Now we examine the accuracy of our algorithms in higher precision. We compare the relative forward errors of `cosm_adv`, `cosm_exp`, `cosm_mp`, and `cosm_mp_s` running at 256 and 1024 decimal digits of precision on the test matrices, and report the same data in the form of performance profiles.

As reported in Figure 6.2, `cosm_mp` delivers superior accuracy to its counterparts and gives the best accuracy in more than 60 percent of the cases. The exponential-based algorithm `cosm_exp` is only slightly less accurate than `cosm_mp`. The Schur-based algorithm `cosm_mp_s` is distinctively less accurate than its Schur-free counterpart. We also note that `cosm_adv` achieves the worst overall accuracy in the experiments and shows signs of forward instability, especially when the number of decimal digits is increased from 256 to 1024, as it gives errors much larger than $\kappa_{\cos}(A)u$ in many cases.

6.3. Speed comparison for computing the cosine. We also compared the execution times of our implementations of `cosm_mp` and `cosm_mp_s` with the other algorithms in double and higher precisions. For this purpose it is sensible to test the algorithms on matrices of different sizes, so we take from $\mathcal{F}$ the matrices whose size is variable in the tests. We denote the new set of 104 matrices by $\mathcal{V}$.

Figure 6.3 shows that, in double precision, `cosm_tay`, `cosm_pol`, `cosm_exp`, and `cosm` are the fastest algorithms and have close performance in computation time. These double-precision-oriented algorithms employ a rational approximant of degree

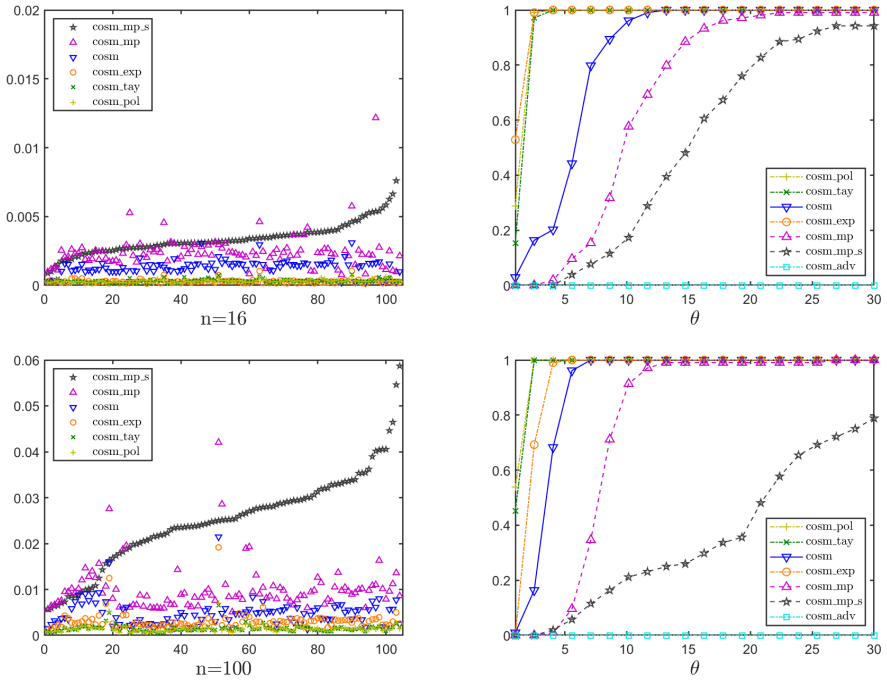

Fig. 6.3: Execution times (in seconds) and the corresponding performance profile of the algorithms for computing matrices in $\mathcal{V}$ in double precision on matrices of different size. The execution times of `cosm_adv` is not plotted because their appreciably larger magnitude makes it destructive and inconvenient to be compared against the others.

chosen from a fixed set based on error bounds with precomputed coefficients and are highly optimized in selecting the degree and scaling parameter, so in general `cosm_mp` and `cosm_mp_s` are not expected to be as efficient. However, from the performance profiles we observe that `cosm_mp` has relatively better performance as n increases from 16 to 100. This is because the $O(n^2)$ flops required by `cosm_mp` in evaluating the error bound, which is extra compared with the above double-precision-oriented algorithms, are expensive for small matrices but become negligible for large n. The Schur-based algorithm `cosm_mp_s` becomes relatively slower as n grows. We also note that `cosm_adv` is appreciably slower than the rest of the algorithms in both cases.

Then we compare the execution times of `cosm_adv`, `cosm_exp`, `cosm_mp`, and `cosm_mp_s` in precisions higher than double. Figure 6.4 reports the execution times and corresponding performance profiles of these algorithms in 256 digits of precision, where matrices of size $n = 16$ and $n = 100$ are used. It is observed that `cosm_mp` is substantially faster than the other algorithms, being the fastest algorithm on about 80 percent of the matrices in both sets. The Schur-based algorithm `cosm_mp_s` is in general faster than `cosm_exp` for $n = 16$, but its performance deteriorates for $n = 100$. `cosm_adv` is the least efficient algorithm whose behavior is unsteady as it can be much slower than other algorithms on certain matrices. We repeated the above experiments in a working precision of 1024 digits, finding similar behavior of the algorithms.

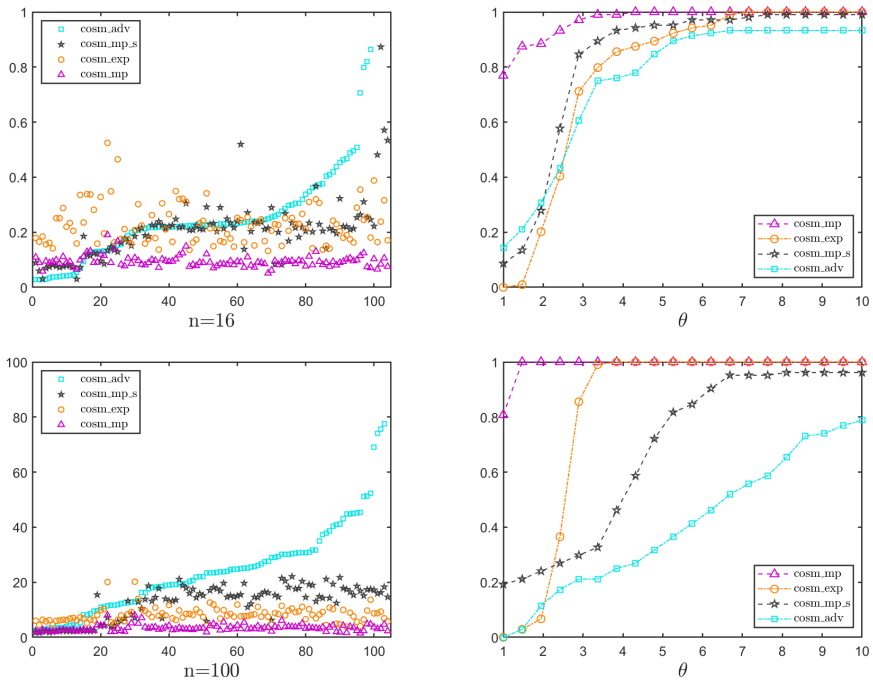

Fig. 6.4: Execution time (in seconds) and corresponding performance profiles of the algorithms in 256 digits of precision on matrices of different sizes.

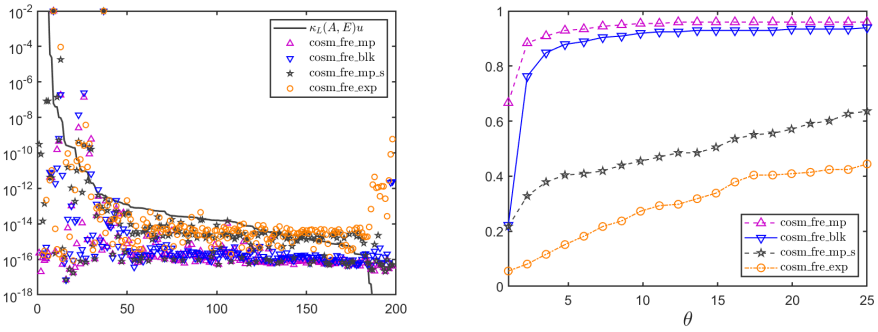

Fig. 6.5: Left: Forward error in $L_{\cos}(A, E)$ on the matrices in $\mathcal{F}$ in double precision, where the solid line is $\kappa_L(A, E)u$. Right: Corresponding performance profiles.

6.4. Accuracy of the computed Fréchet derivative. In this section we examine the accuracy of Algorithm 5.2 for computing the Fréchet derivative in multiprecision arithmetic on the 198 matrices in set $\mathcal{F}$. For each A, we used a different E, generated to have pseudorandom elements drawn from the standard normal distribution and normalised such that $\|E\|_1 = 1$. We evaluate in the 1-norm the relative

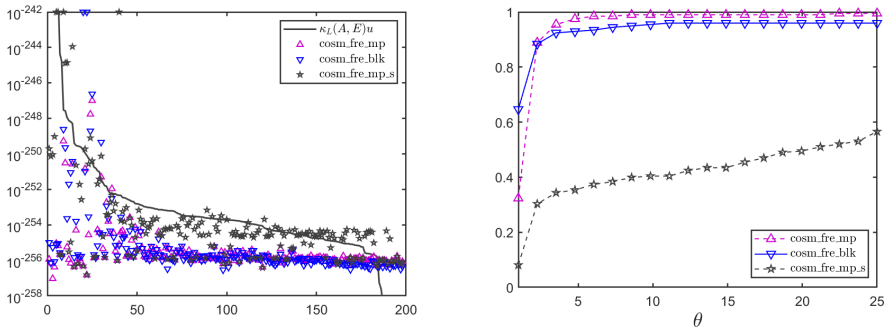

Fig. 6.6: Left: Forward error in $L_{\cos}(A,E)$ on the matrices in $\mathcal{F}$ in 256 digits of precision, where the solid line is $\kappa_L(A,E)u$. Right: Corresponding performance profiles.

forward error of the computed Fréchet derivative. To obtain the reference solution $L_{\cos}(A,E)$ we apply Algorithm 4.4 with $m_{\max} = 2500$ in twice the working precision to the $2n \times 2n$ matrix $\left[\begin{smallmatrix} A & E \\ 0 & A \end{smallmatrix}\right]$ and exploit the property, for arbitrary f [21, eq. (3.16)],

$$f\left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} f(A) & L_f(A,E) \\ 0 & f(A) \end{bmatrix}. \tag{6.2}$$

We tested the following four schemes for computing the Fréchet derivative:

- `cosm_fre_blk`, Algorithm 4.4 with $m_{\max} = 500$ applied to the block 2×2 matrix in (6.2).
- `cosm_fre_mp`, our implementation of Algorithm 5.2 with $m_{\max} = 500$.
- `cosm_fre_mp_s`, our implementation of the real Schur variant of Algorithm 5.2 with $m_{\max} = 500$.
- `cosm_fre_exp`, which computes $L_{\cos}(A,E) = \frac{\mathrm{i}}{2}(L_{\exp}(\mathrm{i}A,E) - L_{\exp}(-\mathrm{i}A,E))$, which is obtained by applying the chain rule to the complex case of (6.1), by invoking the algorithm [2] for computing the Fréchet derivative of the matrix exponential, and is intended for double precision only.

As in the previous experiments in the implementation of Algorithm 4.4 and Algorithm 5.2 we set $k = 3$, which controls the switch between increasing m and s as the truncation error decays. We also measure the forward stability of these algorithms by comparing the error with $\mathrm{cond}_L(A,E)u$, where $\mathrm{cond}_L(A,E)$ is the condition number of the Fréchet derivative, defined as

$$\mathrm{cond}_L(A,E) = \lim_{\epsilon \to 0} \sup_{\substack{\|\Delta A\| \le \epsilon \|A\| \\ \|\Delta E\| \le \epsilon \|E\|}} \frac{\|L_{\cos}(A+\Delta A, E+\Delta E) - L_{\cos}(A,E)\|}{\epsilon \|L_{\cos}(A,E)\|}.$$

We estimate $\mathrm{cond}_L(A,E)$ using an algorithm of Higham and Relton [25].

We observe from Figure 6.5 that `cosm_fre_mp` and `cosm_fre_blk` are competitive in terms of accuracy. This also reflects the robustness of Algorithm 4.4 for computing the matrix cosine. However, `cosm_fre_blk` has eight times the cost and four times the storage requirement of `cosm_fre_mp`, and its performance may depend on the scaling of the perturbation E, which is undesirable [22, sect. 4.3]. All the algorithms except `cosm_fre_exp` behave in a forward stable manner in most of cases. The exponential-

based algorithm `cosm_fre_exp` is in general the least accurate and can be unstable on some very well-conditioned problems.

Finally, we examine the accuracy of the algorithms in precisions higher than double. Figure 6.6 shows a similar trend to that in the double precision. The Schur-free algorithms `cosm_fre_mp` and `cosm_fre_blk` are most accurate and have close performance, and all the three algorithms are reasonably forward stable. We repeated the above experiments in a working precision of 1024 digits, finding similar behavior of the algorithms.

7. Conclusions. Existing algorithms for computing the matrix cosine are all designed for double precision arithmetic and typically require certain precomputed constants that are specific to double precision arithmetic, so they do not conveniently extend to an arbitrary precision environment. In this work we have developed multiprecision algorithms for computing the matrix cosine that can also compute the Fréchet derivative. The algorithms employ a forward error bound on the Taylor approximant to $\cos A$ that combines the hyperbolic cosine function with the quantity $\sqrt{\alpha_m(A^2)}$. We have also derived a framework for computing the Fréchet derivative, constructed an efficient evaluation scheme for computing the cosine and its Fréchet derivative simultaneously in arbitrary precision, and shown how this scheme can be extended to compute the matrix sine, cosine, and their Fréchet derivatives all together.

Experiments show that our new algorithms behave in a forward stable manner in floating-point arithmetic. The transformation-free version of the new algorithm for computing the cosine is competitive in accuracy with the state-of-the-art algorithms in double precision and is the fastest and most accurate among all candidates in working precisions higher than double. The fact that the Fréchet derivative computation in Algorithm 5.2 is based on an error bound that is valid only for the $\cos A$ computation does not appear to affect the accuracy of the computed Fréchet derivative. The new algorithms have been shown to have excellent accuracy on various test matrices as well as their variants multiplied by the imaginary unit i; so the algorithms are also good candidates for computing the matrix hyperbolic cosine function and its Fréchet derivative from the identity $\cosh(A) = \cos(\mathrm{i}A)$.

The analysis and techniques here can be adapted for evaluating other matrix trigonometric and hyperbolic functions in arbitrary precision arithmetic, such as those treated in [1] and the wave-kernel functions investigated in [32] and their Fréchet derivatives. Another possible future direction is to extend our algorithms to compute the action of these functions on a matrix in arbitrary precision as it is actually the matrix-vector products that are required in the solutions of wave equations.

REFERENCES

[1] A. H. Al-Mohy, *A truncated Taylor series algorithm for computing the action of trigonometric and hyperbolic matrix functions*, SIAM J. Sci. Comput., 40 (2018), pp. A1696–A1713, https://doi.org/10.1137/17M1145227.

[2] 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 (2009), pp. 1639–1657, https://doi.org/10.1137/080716426.

[3] A. H. Al-Mohy and N. J. Higham, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989, https://doi.org/10.1137/09074721X.

[4] A. H. Al-Mohy, N. J. Higham, and S. D. Relton, *Computing the Fréchet derivative of the matrix logarithm and estimating the condition number*, SIAM J. Sci. Comput., 35 (2013), pp. C394–C410, https://doi.org/10.1137/120885991.

[5] A. H. Al-Mohy, N. J. Higham, and S. D. Relton, *New algorithms for computing the matrix*

sine and cosine separately or simultaneously, SIAM J. Sci. Comput., 37 (2015), pp. A456–A487, https://doi.org/10.1137/140973979.

[6] P. Alonso, J. Ibáñez, J. Sastre, J. Peinado, and E. Defez, *Efficient and accurate algorithms for computing matrix trigonometric functions*, J. Comput. Appl. Math., 309 (2017), pp. 325–332, https://doi.org/10.1016/j.cam.2016.05.015.

[7] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Rev., 59 (2017), pp. 65–98, https://doi.org/10.1137/141000671.

[8] M. Caliari and F. Zivcovich, *On-the-fly backward error estimate for matrix exponential approximation by Taylor algorithm*, J. Comput. Appl. Math., 346 (2019), pp. 532–548, https://doi.org/10.1016/j.cam.2018.07.042.

[9] P. I. Davies and N. J. Higham, *A Schur–Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 464–485, https://doi.org/10.1137/S0895479802410815.

[10] E. Defez, J. Ibáñez, J. M. Alonso, and P. Alonso-Jordá, *On Bernoulli series approximation for the matrix cosine*, Math. Meth. Appl. Sci., (2020), pp. 1–15, https://doi.org/10.1002/mma.7041.

[11] E. Defez, J. Ibáñez, J. Peinado, J. Sastre, and P. Alonso-Jordá, *An efficient and accurate algorithm for computing the matrix cosine based on new Hermite approximations*, J. Comput. Appl. Math., 348 (2019), pp. 1–13, https://doi.org/10.1016/j.cam.2018.08.047.

[12] N. J. Dingle and N. J. Higham, *Reducing the influence of tiny normwise relative errors on performance profiles*, ACM Trans. Math. Software, 39 (2013), pp. 24:1–24:11, https://doi.org/10.1145/2491491.2491494.

[13] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213, https://doi.org/10.1007/s101070100263.

[14] M. Fasi, *Optimality of the Paterson–Stockmeyer method for evaluating matrix polynomials and rational matrix functions*, Linear Algebra Appl., 574 (2019), pp. 182–200, https://doi.org/10.1016/j.laa.2019.04.001.

[15] M. Fasi and N. J. Higham, *Multiprecision algorithms for computing the matrix logarithm*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 472–491, https://doi.org/10.1137/17M1129866.

[16] M. Fasi and N. J. Higham, *An arbitrary precision scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 1233–1256, https://doi.org/10.1137/18M1228876.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, fourth ed., 2013.

[18] G. Hargreaves, *Topics in matrix computations: Stability and efficiency of algorithms*, PhD thesis, University of Manchester, Manchester, England, Aug. 2005.

[19] G. I. Hargreaves and N. J. Higham, *Efficient algorithms for the matrix cosine and sine*, Numer. Algorithms, 40 (2005), pp. 383–400, https://doi.org/10.1007/s11075-005-8141-0.

[20] N. J. Higham, *The Matrix Computation Toolbox*. http://www.maths.manchester.ac.uk/~higham/mctoolbox.

[21] N. J. Higham, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, https://doi.org/10.1137/1.9780898717778.

[22] N. J. Higham and L. Lin, *An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1341–1360, https://doi.org/10.1137/130906118.

[23] N. J. Higham and X. Liu, *A multiprecision derivative-free Schur–Parlett algorithm for computing matrix functions*, MIMS EPrint 2020.19, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Sept. 2020, http://eprints.maths.manchester.ac.uk/2824/. Revised June 2021. To appear in SIAM J. Matrix Anal. Appl.

[24] N. J. Higham and M. Mikaitis, *Anymatrix: Extendable MATLAB Matrix Collections*. https://github.com/mmikaitis/anymatrix.

[25] N. J. Higham and S. D. Relton, *Estimating the condition number of the Fréchet derivative of a matrix function*, SIAM J. Sci. Comput., 36 (2014), pp. C617–C634, https://doi.org/10.1137/130950082.

[26] N. J. Higham and M. I. Smith, *Computing the matrix cosine*, Numer. Algorithms, 34 (2003), pp. 13–26, https://doi.org/10.1023/A:1026152731904.

[27] N. J. Higham and F. Tisseur, *A block algorithm for matrix* 1*-norm estimation, with an application to* 1*-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201, https://doi.org/10.1137/S0895479899356080.

[28] F. Johansson et al., *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*. http://mpmath.org/.

[29] A. Magnus and J. Wynn, *On the padé table of* $\cos z$, Proc. Amer. Math. Soc., 47 (1975), pp. 361–367, https://doi.org/10.2307/2039747.

[30] A. Meurer, C. P. Smith, M. Paprocki, O. Čertik, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, *SymPy: Symbolic computing in Python*, PeerJ Comput. Sci., 3 (2017), p. e103, https://doi.org/10.7717/peerj-cs.103.

[31] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. http://www.advanpix.com.

[32] P. Nadukandi and N. J. Higham, *Computing the wave-kernel matrix functions*, SIAM J. Sci. Comput., 40 (2018), pp. A4060–A4082, https://doi.org/10.1137/18M1170352.

[33] M. S. Paterson and L. J. Stockmeyer, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM J. Comput., 2 (1973), pp. 60–66, https://doi.org/10.1137/0202007.

[34] J. Sastre, J. Ibánez, P. Alonso, J. Peinado, and E. Defez, *Two algorithms for computing the matrix cosine function*, Appl. Math. Comput., 312 (2017), pp. 66–77, https://doi.org/10.1016/j.amc.2017.05.019.

[35] J. Sastre, J. Ibánez, P. Alonso-Jordá, J. Peinado, and E. Defez, *Fast Taylor polynomial evaluation for the computation of the matrix cosine*, J. Comput. Appl. Math., 354 (2019), pp. 641–650, https://doi.org/10.1016/j.cam.2018.12.041.

[36] J. Sastre, J. Ibánez, P. Ruiz, and E. Defez, *Efficient computation of the matrix cosine*, Appl. Math. Comput., 219 (2013), pp. 7575–7585, https://doi.org/10.1016/j.amc.2013.01.043.

[37] S. M. Serbin, *Rational approximations of trigonometric matrices with application to second-order systems of differential equations*, Appl. Math. Comput., 5 (1979), pp. 75–92, https://doi.org/10.1016/0096-3003(79)90011-0.

[38] S. M. Serbin and S. A. Blalock, *An algorithm for computing the matrix cosine*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 198–204, https://doi.org/10.1137/0901013.

[39] M. Seydaoglu, P. Bader, S. Blanes, and F. Casas, *Computing the matrix sine and cosine simultaneously with a reduced number of products*, arXiv:2010.00465 [math.NA], 2020, https://arxiv.org/abs/2010.00465.