A Truncated Taylor Series Algorithm for Computing the Action of Trigonometric and Hyperbolic Matrix Functions

Article *in* SIAM Journal on Scientific Computing · January 2018

DOI: 10.1137/17M1145227

Awad Al-Mohy
King Khalid University

SIAM J. SCI. COMPUT.
Vol. 40, No. 3, pp. A1696–A1713

A TRUNCATED TAYLOR SERIES ALGORITHM FOR COMPUTING THE ACTION OF TRIGONOMETRIC AND HYPERBOLIC MATRIX FUNCTIONS[*]

AWAD H. AL-MOHY[†]

Abstract. A new algorithm is derived for computing the action $f(tA)B$, where A is an $n \times n$ matrix, B is $n \times n_0$ with $n_0 \ll n$, and f is cosine, sinc, sine, hyperbolic cosine, hyperbolic sinc, or hyperbolic sine function. In the case of f being even, the computation of $f(tA^{1/2})B$ is possible without explicitly computing $A^{1/2}$, where $A^{1/2}$ denotes any matrix square root of A. The algorithm offers six independent output options given t, A, B, and a tolerance. For each option, actions of a pair of trigonometric or hyperbolic matrix functions are simultaneously computed. The algorithm scales the matrix A down by a positive integer s, approximates $f(s^{-1}tA^{\sigma})B$, where σ is either 1 or 1/2, by using a truncated Taylor series, and finally uses the recurrences of the Chebyshev polynomials of the first and second kind to recover $f(tA^{\sigma})B$. The selection of the scaling parameter and the degree of Taylor polynomial is based on a forward error analysis and a sequence of the form $\|A^k\|^{1/k}$ in such a way that the overall computational cost of the algorithm is minimized. Shifting is used where applicable as a preprocessing step to reduce the scaling parameter. The algorithm works for any matrix A, and its computational cost is dominated by the formation of products of A with $n \times n_0$ matrices that could take advantage of the implementation of level-3 BLAS. Our numerical experiments show that the new algorithm behaves in a forward stable fashion and in most problems outperforms the existing algorithms in terms of CPU time, computational cost, and accuracy.

Key words. matrix cosine, matrix sine, sinc function, hyperbolic cosine, hyperbolic sine, Taylor series, ordinary differential equation, variation of the constants formula, trigonometric integrators, Chebyshev polynomials, MATLAB

AMS subject classifications. 15A60, 65F30

DOI. 10.1137/17M1145227

1. Introduction.

The matrix cosine and sine functions appear in the solution of the system of second order differential equations

$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} + Ay = g(y(t)), \quad y(0) = y_0, \quad y'(0) = y'_0. \tag{1.1}$$

The exact solution of this system and its derivative is given by the variation of the constants formula [10, 27]

$$\begin{aligned} y(t) = \cos(tA^{1/2})y_0 + t\,\mathrm{sinc}(tA^{1/2})y'_0 \\ + \int_0^t (t-\tau)\,\mathrm{sinc}((t-\tau)A^{1/2})g(y(\tau))\mathrm{d}\tau, \end{aligned} \tag{1.2}$$

$$\begin{aligned} y'(t) = -A^{1/2}\sin(tA^{1/2})y_0 + \cos(tA^{1/2})y'_0 \\ + \int_0^t (t-\tau)\cos((t-\tau)A^{1/2})g(y(\tau))\mathrm{d}\tau, \end{aligned} \tag{1.3}$$

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section August 28, 2017; accepted for publication (in revised form) March 16, 2018; published electronically June 5, 2018.
http://www.siam.org/journals/sisc/40-3/M114522.html
Funding: This work was supported by King Khalid University.
[†]Department of Mathematics, King Khalid University, Abha, Saudi Arabia (ahalmohy@kku.edu.sa).

where $A^{1/2}$ denotes any matrix square root of A and sinc : $\mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ is defined as

$$\operatorname{sinc} X = \sum_{k=0}^{\infty} \frac{(-1)^k X^{2k}}{(2k+1)!}. \tag{1.4}$$

The matrix function sinc clearly satisfies the relation $X\operatorname{sinc}X = \sin X$. The first term of (1.3) can be rewritten using the equality

$$A^{1/2}\sin(tA^{1/2}) = tA\operatorname{sinc}(tA^{1/2}).$$

This is important to remove any ambiguity that a square root of A is needed. We will see below how the actions of $\cos(tA^{1/2})$ and $\operatorname{sinc}(tA^{1/2})$ can be simultaneously computed without explicitly computing $A^{1/2}$, whereas it is impossible to evaluate the action of $\sin(tA^{1/2})$ without forming $A^{1/2}$ explicitly because sin is an odd function.

The variation of the constants formula forms the basis of numerical schemes to solve the problem. For instance, at time $t_n = nh$, $y(t_n)$ and $y'(t_n)$ can be numerically approximated by y_n and y'_n, respectively, via the trigonometric scheme

$$y_{n+1} = \cos(hA^{1/2})y_n + h\operatorname{sinc}(hA^{1/2})y'_n + \frac{h^2}{2}\operatorname{sinc}(hA^{1/2})\widehat{g}(y_n), \tag{1.5}$$

$$y'_{n+1} = -hA\operatorname{sinc}(hA^{1/2})y_n + \cos(hA^{1/2})y'_n + \frac{h}{2}\cos(hA^{1/2})\widehat{g}(y_n) + \frac{h}{2}\widehat{g}(y_{n+1}), \tag{1.6}$$

where $\widehat{g}(y) = \psi(hA^{1/2})g(\phi(hA^{1/2})y)$ provided that ψ and ϕ are suitably chosen continuous filter functions; see [9, sect. 2], [10, sect. 2], or [13, sect. XIII.2.2]. Many filter functions are proposed in the literature and most of them involve several actions of $\operatorname{sinc}(hA^{1/2})$ to evaluate $\widehat{g}(y)$. For example Hairer and Lubich [12] chose $\psi = \operatorname{sinc}$ and $\phi = 1$, while Grimm and Hochbruck proposed $\psi = \operatorname{sinc}^2$ and $\phi = \operatorname{sinc}$ [10].

The system (1.1) arises from semidiscretization of some second order PDE's by finite difference or finite element methods [24]. The hyperbolic matrix functions: $\cosh A$, $\sinh A$, and $\operatorname{sinch} A$, where $\operatorname{sinch} A = \operatorname{sinc}(\mathrm{i}A)$, arise in the solution of coupled hyperbolic systems of PDE's [22]. They also have application in communicability analysis in complex networks [7]. Some applications, as in the scheme (1.5)–(1.6), for instance, don't require $f(A)$ explicitly but rather the action of it on a vector: $f(A)b$. When A is large and sparse, $f(A)$ may be too expensive to compute or store, and it could also be much denser than A.

The computation of the action of the matrix exponential has received significant research attention; see [2] and the references therein. However, this is not the case for trigonometric and hyperbolic matrix functions. One possible reason is that the second order system (1.1) can be presented in a block form of a first order system of ODE's, and the matrix exponential is used to solve the problem as in (6.1). Grimm and Hochbruck [11] proposed the use of a rational Krylov subspace method instead of the standard one for certain problems for computing $\cos(tA^{1/2})b$ and $\operatorname{sinc}(tA^{1/2})b$.

Recently, Higham and Kandolf [19] derived an elegant algorithm for computing the action of trigonometric and hyperbolic matrix functions. They adapted the existing algorithm of Al-Mohy and Higham [2], `expmv`, for computing the action of the matrix exponential so that the simultaneous evaluation of $\cos(A)B$ and $\sin(A)B$ (or $\cosh(A)B$ and $\sinh(A)B$) requires a single action of e^A on the matrix $[B, B]/2 \in \mathbb{C}^{n\times 2n_0}$. The algorithm inherits the backward stability of `expmv` as the authors successfully associated the backward error bounds for the trigonometric and hyperbolic matrix functions to that of the matrix exponential. Furthermore, the algorithm of Higham and Kandolf is a generalization of `expmv` that allows simultaneous computation of $e^{t_i A}B$ for different values of t_i. They adapted the time scalar parameter t of `expmv` and replaced it by a "time matrix" T of which particular settings yield the actions of the trigonometric and hyperbolic matrix functions.

The calculation of $\cos A$ and $\sin A$ for dense A of medium size is well-studied. Serbin and Blalock [25] proposed an algorithm for $\cos A$. It begins by approximating $\cos(2^{-s}A)$ by a Taylor or Padé approximant, where s is a nonnegative integer, and then applies the double angle formula $\cos(2A) = 2\cos^2(A) - I$ on the approximant s times to recover the original matrix cosine. An algorithm by Higham and Smith [20] uses the [8/8] Padé approximant with the aid of a forward error analysis to specify the scaling parameter s. Hargreaves and Higham [14] develop an algorithm with a variable choice of the degree of Padé approximants based on forward error bounds in such a way that the computational cost is minimized. They also derive an algorithm that computes $\cos A$ and $\sin A$ simultaneously. Recently, Al-Mohy, Higham, and Relton [3] derived new backward stable algorithms for computing $\cos A$ and $\sin A$ separably or simultaneously using Padé approximants and rational approximations obtained from Padé approximants to the exponential function. They use the triple angle formula to have an independent algorithm for $\sin A$. In spite of the fact that the algorithms based on the double and triple angle formulas for computing $\cos A$ and $\sin A$, respectively, prove to be a great success, it doesn't seem that these formulas can be adapted to compute the action of these matrix functions.

In this paper we derive a new algorithm for computing the action of the trigonometric and hyperbolic matrix functions of the form $f(tA)B$ and $f(tA^{1/2})B$ without computing $A^{1/2}$. The form $f(tA^{1/2})B$ appears in the variation of constants formula (1.2)–(1.3). In contrast, the algorithm of Higham and Kandolf neither seems ready to compute $f(tA^{1/2})B$ without explicitly providing $A^{1/2}$ nor can it directly return $\mathrm{sinc}(tA)B$ or $\mathrm{sinch}(tA)B$.

The paper is organized as follows. In section 2 we exploit the recurrences of the Chebyshev polynomials and explain how the actions of trigonometric and hyperbolic matrix functions can be computed. In section 3 we present forward error analysis using truncated Taylor series and computational cost analysis to determine optimal scaling parameters and degrees of Taylor polynomials for various tolerances. Preprocessing by shifting and termination criterion is discussed in section 4. We write our algorithm in section 5 and then give numerical experiments in section 6. Finally we draw some concluding remarks in section 7.

2. Computing the actions $f(tA)B$ and $f(tA^{1/2})B$.

In this section we exploit trigonometric formulas and derive recurrences to computing the action of the matrix functions $\cos X$, $\mathrm{sinc}X$, $\sin X$, $\cosh X$, $\mathrm{sinch}\, X$, and $\sinh X$ on a thin matrix B. For an integer k we have

$$\cos(kX) + \cos((k-2)X) = 2\cos(X)\cos((k-1)X). \tag{2.1}$$

Multiplying the sides of this equation from the right by B and setting $C_k(X,B) = \cos(kX)B$ (denoted for simplicity by C_k) for $k \geq 0$ yield the three term recurrence

$$C_k + C_{k-2} = 2\cos(X)C_{k-1} = 2\,C_1(X, C_{k-1}), \quad k \geq 2. \tag{2.2}$$

Observe that for $Y = \cos(X)$ the matrix $\cos(kX)$ is a polynomial of degree k in Y. Precisely, $\cos(kX) = T_k(Y)$, where $T_k(Y)$ is *the Chebyshev polynomial of the first kind* that satisfies the recurrence: $T_k(Y) + T_{k-2}(Y) = 2YT_{k-1}(Y)$, $T_0 = I$, $T_1 = Y$ [23]. Thus, $C_k = T_k(Y)B$. The heaviest computational work in the recurrence (2.2) lies in $C_1(X, C_{k-1})$ for all $k \geq 1$. Let r be a rational approximation to the cosine function, which we assume to be good near the origin, and choose a positive integer $s \geq 1$ so that $\cos(s^{-1}A)$ is well-approximated by $r(s^{-1}A)$. Thus,

$$C_1(s^{-1}A, C_{k-1}) = \cos(s^{-1}A)C_{k-1} \approx r(s^{-1}A)C_{k-1}.$$

The recurrence (2.2) with $X = s^{-1}A$ yields

$$C_s(s^{-1}A, B) = \cos(A)B.$$

We choose for r a truncated Taylor series

$$r_m(x) = \sum_{j=0}^{m} \frac{(-1)^j x^{2j}}{(2j)!}$$

and compute the matrix $V = r_m(s^{-1}A)B$ using consecutive matrix products as shown by the next pseudocode.

Code Fragment 2.1.

```
1  $V = B$
2  for $k = 1{:}\,m$
3      $\beta = 2k$, $\gamma = 2k - 1$
4      $B = AB$
5      $B = (AB)\left(s^2\beta\gamma\right)^{-1}$
6      $V = V + (-1)^k B$
7  end
```

Similarly we approximate $\operatorname{sinc} x$ by truncating the Taylor series in (1.4) as

$$\widetilde{r}_m(x) = \sum_{j=0}^{m} \frac{(-1)^j x^{2j}}{(2j+1)!}.$$

The matrix $V := \widetilde{r}_m(s^{-1}A)B$ can be evaluated using Code Fragment 2.1 after replacing γ in line 3 by $\gamma = 2k + 1$. To evaluate $r_m(s^{-1}A^{1/2})B$ or $\widetilde{r}_m(s^{-1}A^{1/2})B$, we need only delete line 4 of Code Fragment 2.1.

Next, to compute $\operatorname{sinc}(A)B$ consider the three term recurrence

$$S_k - S_{k-2} = 2C_k, \quad k \geq 2, \quad S_0 = B, \quad S_1 = 2C_1 = 2\cos(X)B. \tag{2.3}$$

It is the recurrence that yields the *Chebyshev polynomials of the second kind* [23]. By induction on k, it easy to verify that

$$\sin(X)S_{k-1} = \sin(kX)B. \tag{2.4}$$

Assume for a temporarily fixed positive integer $q \geq 2$ that (2.4) holds for all k with $q \geq k \geq 2$. The inductive step follows from

$$\begin{aligned}\sin(X)S_{q+1} &= 2\sin(X)C_{q+1} + \sin(X)S_{q-1}\\ &= 2\sin(X)\cos((q+1)X)B + \sin(qX)B\\ &= \big[\sin((q+2)X) - \sin(qX)\big]B + \sin(qX)B = \sin((q+2)X)B.\end{aligned}$$

Since (2.4) holds for every X we conclude that

$$\operatorname{sinc}(X)S_{k-1} = k \operatorname{sinc}(kX)B. \tag{2.5}$$

For $X = s^{-1}A$ the recurrences (2.2) and (2.3) can intertwine the computation of $C_s = \cos(A)B$ and S_{s-1}. The matrix $\operatorname{sinc}(A)B$ can be recovered by computing the action $\operatorname{sinc}(s^{-1}A)S_{s-1} \approx \widetilde{r}_m(s^{-1}A)S_{s-1}$ that can be achieved by a single execution of Code Fragment 2.1 with $V = S_{s-1}$ and $\gamma = 2k+1$ in line 3. Observe that the calculation of S_{s-1} via (2.3) involves only $s-2$ additions of $n \times n_0$ matrices provided that C_k, $1 \le k \le s$, are already computed from (2.2). Such operations are negligible. However, we can eliminate about half of these operations by observing that

$$\frac{1}{2}S_{s-1} = \begin{cases} C_1 + C_3 + C_5 \cdots + C_{s-1} & \text{if } s \text{ is even,} \\ \frac{1}{2}C_0 + C_2 + C_4 + \cdots + C_{s-1} & \text{if } s \text{ is odd,} \end{cases} \tag{2.6}$$

which can be easily derived from (2.3). It is worth pointing out that we do not store the sequence $\{C_k\}$ and then form S_{s-1} but rather we compute it via overwriting a variable S, say, during the calculation of C_k as in lines 45 and 46 of Algorithm 5.1.

To avoid unnecessary repetition, the analogy between the identities of the trigonometric and hyperbolic functions allows us to replace cos in (2.2) and (2.3) by cosh, and replace sinc in (2.5) by sinch so that the recurrence relations return $\cosh(A)B$, $\operatorname{sinch}(A)B$, and $\sinh(A)B$.

3. Forward error analysis and computational cost analysis.

We use the truncated Taylor series r_m and $\widetilde{r}_m$ to approximate the cos and sinc functions, respectively. Given a matrix $A \in \mathbb{C}^{n\times n}$ and tolerance tol, we need to determine the positive integer s so that

$$\|\cos(s^{-1}A^\sigma) - r_m(s^{-1}A^\sigma)\| \le \text{tol}, \tag{3.1}$$

where σ is either 1 or 1/2. We have

$$\cos(s^{-1}A^\sigma) - r_m(s^{-1}A^\sigma) = \sum_{j=m+1}^{\infty} \frac{(-1)^j (s^{-1}A^\sigma)^{2j}}{(2j)!}.$$

By [1, Thm. 4.2(b)] and since the tail of the Taylor series of the cosine is an even function, we obtain

$$\begin{aligned} \|\cos(s^{-1}A^\sigma) - r_m(s^{-1}A^\sigma)\| &\le \sum_{j=m+1}^{\infty} \frac{\alpha_p(s^{-1}A^\sigma)^{2j}}{(2j)!} \\ &= \cosh(\alpha_p(s^{-1}A^\sigma)) - \sum_{j=0}^{m} \frac{\alpha_p(s^{-1}A^\sigma)^{2j}}{(2j)!} =: \rho_m(\alpha_p(s^{-1}A^\sigma)), \end{aligned} \tag{3.2}$$

where

$$\alpha_p(X) = \max\big(d_{2p}, d_{2p+2}\big), \quad d_k = \|X^k\|^{1/k}, \tag{3.3}$$

and p is any positive integer satisfying the constraint $m+1 \ge p(p-1)$. In addition, it is straightforward to verify that

$$\|\operatorname{sinc}(s^{-1}A^\sigma) - \widetilde{r}_m(s^{-1}A^\sigma)\| \le \sum_{j=m+1}^{\infty} \frac{\alpha_p(s^{-1}A^\sigma)^{2j}}{(2j+1)!} \le \rho_m(\alpha_p(s^{-1}A^\sigma)). \tag{3.4}$$

Similarly, the forward errors of the approximations of cosh and sinch by Taylor polynomials have exactly the same bound ρ_m.

Next we analyze the computational cost and determine how to choose the scaling parameter and the degree of Taylor polynomial. Define

$$\theta_m = \max\{\, \theta : \rho_m(\theta) \le \mathrm{tol}\,\}. \tag{3.5}$$

Thus, given m and p, if s is chosen so that $s^{-1}\alpha_p(A^\sigma) \le \theta_m$, then the inequality $\rho_m(\alpha_p(s^{-1}A^\sigma)) \le \mathrm{tol}$ will be satisfied and therefore the absolute forward error will be bounded by tol. Table 3.1 lists selected values of θ_m for tol $= 2^{-10}$ (half precision), tol $= 2^{-24}$ (single precision), and tol $= 2^{-53}$ (double precision). These values were determined as described in [18, App.]. For each m, the optimal value of the scaling parameter s is given by $s = \max(\lceil \alpha_p(A^\sigma)/\theta_m \rceil, 1)$. The computational cost of evaluating C_s in view of Code Fragment 2.1 is $2\sigma ms$ matrix–matrix multiplications of the form AB. That is, $2\sigma n_0 ms$ matrix–vector products since B has n_0 columns. By (2.6), S_{s-1} is obtained with a negligible cost. $\mathrm{sinc}(A)B$ can be then recovered by a single invocation of Code Fragment 2.1 for $V = S_{s-1}$ and $\gamma = 2k+1$; this requires only $2\sigma n_0 m$ matrix–vector products. After that, one multiplication is needed to recover $\sin(A)B$ from $\mathrm{sinc}(A)B$; that is, n_0 matrix–vector products. We build our cost analysis on the assumption that the output of our algorithm is $\cos(A^\sigma)B$ and $\mathrm{sinc}(A^\sigma)B$. Note that when $\sigma = 1/2$, $\sin(A^\sigma)B$ cannot be obtained without computing $A^{1/2}$. Thus, the total cost is

$$2\sigma n_0 m(s+1) = 2\sigma n_0 m\big(\max(\lceil \alpha_p(A^\sigma)/\theta_m \rceil, 1) + 1\big) \tag{3.6}$$

matrix–vector products. We observe that this quantity tends to be decreasing as m increases though the decreasing is not necessarily monotonic. The sequence $\{m/\theta_m\}$ is strictly decreasing, while the sequence $\{\alpha_p(X)\}$ has a generally nonincreasing trend for any X. Thus, the larger m is, the less the cost. However, a large value of m could lead to unstable calculation of Taylor polynomials $r_m(A^\sigma)B$ for large $\|A^\sigma\|$ in floating point arithmetic. Thus, we impose a limit $m_{\max}$ on m and seek m_* that minimizes the computational cost over all p such that $p(p-1) \le m_{\max} + 1$. For the moment we drop the max in (3.6), whose purpose is simply to cater for nilpotent A^σ with $A^{\sigma j} = 0$ for $j \ge 2p$. Moreover, we remove constant terms since they essentially do not affect the optimization for the value of m_*. Thus, we consider the sequence

$$\widehat{C}_m(A^\sigma) = m\lceil \alpha_p(A^\sigma)/\theta_m \rceil$$

to be minimized subject to some constraints. Note that $\|A^\sigma\| \ge d_2 \ge d_{2k}$ in (3.3) for all $k \ge 1$ and so

$$\|A^\sigma\| \ge \alpha_1(A^\sigma) = d_2 = \|A^{2\sigma}\|^{1/2} \ge \alpha_p(A^\sigma) \tag{3.7}$$

for all $p \ge 1$. Hence, we do not need to consider the case $p = 1$ when minimizing $\widehat{C}_m(A^\sigma)$ since $\widehat{C}_m(A^\sigma) \le m\lceil \alpha_1(A^\sigma)/\theta_m \rceil$. Let $p_{\max}$ denote the largest positive integer p such that $p(p-1) \le m_{\max} + 1$. Let m_* be the smallest value of m at which the minimum

$$\widehat{C}_{m_*}(A^\sigma) = \min\big\{\, m\lceil \alpha_p(A^\sigma)/\theta_m \rceil : 2 \le p \le p_{\max},\ \ p(p-1) - 1 \le m \le m_{\max} \,\big\} \tag{3.8}$$

is attained [2, eq. (3.11)]. The optimal scaling parameter then is

$$s = \max(\widehat{C}_{m_*}(A^\sigma)/m_*, 1).$$

Table 3.1
Selected constants θ_m for $\mathrm{tol} = 2^{-10}$ *(half),* $\mathrm{tol} = 2^{-24}$ *(single), and* $\mathrm{tol} = 2^{-53}$ *(double).*

$2m$	6	10	14	18	22	26	30	34	38	42	46	50
half	1.6e0	3.0e0	4.4e0	5.8e0	7.3e0	8.8e0	1.0e1	1.2e1	1.3e1	1.5e1	1.6e1	1.8e1
single	4.7e-1	1.3e0	2.4e0	3.6e0	4.9e0	6.2e0	7.6e0	9.0e0	1.0e1	1.2e1	1.3e1	1.5e1
double	3.8e-2	2.5e-1	6.8e-1	1.3e0	2.1e0	3.0e0	4.1e0	5.1e0	6.3e0	7.5e0	8.7e0	1.0e1

Our experience and observation indicate that $p_{\max} = 5$ and $m_{\max} = 25$ are appropriate choices for our algorithm. However, the algorithm supports user-specified values of $p_{\max}$ and $m_{\max}$. The MATLAB code of the algorithm allows $m_{\max}$ to be taken up to 50, and consequently the choice of $p_{\max}$ must not exceed 7.

The forward error analysis and cost analysis are valid for any matrix norm, but it is most convenient to use the 1-norm since it is easy to be efficiently estimated using the block 1-norm estimation algorithm of Higham and Tisseur [21]. We estimate the quantities $d_k = \|A^{\sigma k}\|_1^{1/k}$, where k is even as defined in (3.3), which are required to form $\alpha_p(A^\sigma)$. The algorithm of Higham and Tisseur estimates $\|A^{\sigma k}\|_1$ via about two actions of $A^{\sigma k}$ and two actions of $(A^*)^{\sigma k}$, all on matrices of ℓ columns, where the positive integer ℓ is a parameter (typically set to 1 or 2). The number σk is a positive integer since k is even, so fractional powers of A are completely avoided. Therefore, obtaining $\alpha_p(A^\sigma)$ for $p = 2\colon p_{\max}$ costs approximately

$$8\sigma\ell \sum_{p=2}^{p_{\max}+1} p = 4\sigma\ell p_{\max}(p_{\max}+3) \tag{3.9}$$

matrix–vector products. Thus, in view of (3.6), if it happens that

$$2\sigma n_0 m_{\max}\big(\|A\|_1^\sigma/\theta_{m_{\max}} + 1\big) \le 4\sigma\ell p_{\max}(p_{\max}+3),$$

or equivalently

$$\|A\|_1^\sigma \le \theta_{m_{\max}}\left(\frac{2\ell}{n_0 m_{\max}} p_{\max}(p_{\max}+3) - 1\right), \tag{3.10}$$

then the computational cost of evaluating C_s and $\widetilde{r}_m(s^{-1}A^\sigma)S_{s-1}$ with m determined by using $\|A\|_1^\sigma$ or $\|A^{2\sigma}\|_1^{1/2}$ in place of $\alpha_p(A^\sigma)$ in (3.8) is no larger than the cost (3.9) of computing the sequence $\{\alpha_p(A^\sigma)\}$. Thus, we should certainly use $\|A\|_1^\sigma$ if $\sigma = 1$ or $\|A\|_1^{1/2}$ if $\sigma = 1/2$ in place of $\alpha_p(A^\sigma)$ for each p in light of the inequalities in (3.7).

In the case $\sigma = 1$, we still have another chance to avoid estimating $\alpha_p(A)$ for $p > 2$. If the inequality (3.10) is unsatisfied, the middle bound d_2 in (3.7) can be estimated and its actual cost, ν matrix–vector products, can be calculated. We check again if the bound

$$d_2 \le \theta_{m_{\max}}\left(\frac{2\ell}{n_0 m_{\max}} p_{\max}(p_{\max}+3) - \nu - 1\right) \tag{3.11}$$

holds. We sum up our analysis for determining the parameters m_* and s in the following code.

Code Fragment 3.1 ($[m_*, s, \Theta_\sigma] = \mathrm{parameters}(A, \sigma, \mathrm{tol})$). *This code determines m_* and s given A, σ,* tol, *$m_{\max}$, and $p_{\max}$. Let Θ_σ denote the number of the actual matrix–vector products needed to estimate the sequence $\{\alpha_p(A^\sigma)\}$.*

```
1  if (3.10) is satisfied
2    $m_* = \operatorname{argmin}_{1\le m\le m_{\max}} m\lceil \|A\|_1^\sigma/\theta_m\rceil$
3    $s = \lceil \|A\|_1^\sigma/\theta_{m_*}\rceil$
4    goto line 16
5  end
6  if $\sigma = 1$
7    Compute $d_2$.
8    if (3.11) is satisfied
9      $m_* = \operatorname{argmin}_{1\le m\le m_{\max}} m\lceil d_2/\theta_m\rceil$
10     $s = \lceil d_2/\theta_{m_*}\rceil$
11     goto line 16
12   end
13 end
14 Let $m_*$ be the smallest $m$ achieving the minimum in (3.8).
15 $s = \max\big(\widehat{C}_{m_*}(A^\sigma)/m_*, 1\big)$
16 end
```

As explained in [2, sect. 3], if we wish to compute $f(tA^\sigma)B$ for several values of t, we need not invoke Code Fragment 3.1 for each $t^{1/\sigma}A$. The trick is that since $\alpha_p(tA^\sigma) = |t|\alpha_p(A^\sigma)$, we can precompute the matrix $M \in \mathbb{R}^{(p_{\max}-1)\times m_{\max}}$ given by

$$M_{pm} = \begin{cases} \dfrac{\alpha_p(A^\sigma)}{\theta_m}, & 2 \le p \le p_{\max},\ \ p(p-1)-1 \le m \le m_{\max}, \\ 0 & \text{otherwise}, \end{cases} \tag{3.12}$$

and then for each t obtain $\widehat{C}_{m_*}(tA^\sigma)$ as the smallest nonzero element in the matrix $\lceil |t|M\rceil \operatorname{diag}(1,2,\dots,m_{\max})$, where m_* is the column index of the smallest element. The benefit of basing the selection of the scaling parameter on $\alpha_p(A)$ instead of $\|A\|$ is that $\alpha_p(A)$ can be much smaller than $\|A\|$ for highly nonnormal matrices.

4. Preprocessing and termination criterion.

In this section we discuss several strategies to improve the algorithm stability and reduce its computational cost. The algorithmic scaling parameter s plays an important role in that the smaller the s, the better the stability of the algorithm in general, and the lower the computational cost. That is why we rely on $\alpha_p(A)$ instead of merely using $\|A\|$ to produce the scaling parameter. Al-Mohy and Higham [2, sect. 3.1] proposed an argument reduction and a termination criterion. They found empirically that the shift $\mu = n^{-1}\operatorname{trace}(A)$ [17, Thm. 4.21] that minimizes the Frobenius norm of the matrix $\widetilde{A} = A - \mu I$ leads to values of $\alpha_p(\widetilde{A})$ smaller than $\alpha_p(A)$. We use this shift here if the required outputs are $\cos(A)B$ and $\sin(A)B$ or $\cosh(A)B$ and $\sinh(A)B$. There are cases where shifting is impossible to recover. This happens when the required outputs include $\operatorname{sinc}(A)B$, $\operatorname{sinch}(A)B$, or any form of $f(A^{1/2})B$.

We can recover the original cosine and sine of A from the computed cosine and sine of $\widetilde{A}$ using the formulas

$$\cos A = \cos\mu\cos\widetilde{A} - \sin\mu\sin\widetilde{A}, \quad \sin A = \cos\mu\sin\widetilde{A} + \sin\mu\cos\widetilde{A}. \tag{4.1}$$

The functions $\cosh A$ and $\sinh A$ have analogous formulas containing $\cosh\mu$ and $\sinh\mu$, which could *overflow* for large enough $|\operatorname{Re}(\mu)|$. The same problem arises for $\cos\mu$ and $\sin\mu$ if $|\operatorname{Im}(\mu)|$ is large enough. Al-Mohy and Higham successfully overcome this problem in their algorithm for the matrix exponential by undoing the effect of the scaled shift right after the inner loop of [2, Alg. 3.2]. It is possible to do so for trigonometric and hyperbolic matrix functions. We can undo the effect of the scaled shift in $\cos(s^{-1}\widetilde{A})C_{k-1}$ for each k in the recurrence (2.2) using the formula in (4.1), which requires $\sin(s^{-1}\widetilde{A})C_{k-1}$. The next code shows how $\sin(s^{-1}\widetilde{A})C_{k-1}$ can be formed using the already generated power actions $\widetilde{A}^{2k}B$.

Code Fragment 4.1. *Given $\widetilde{A} = A - \mu I \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times n_0}$, and a suitable chosen scaling parameter s, this code returns $C = r_m(s^{-1}A)B \approx \cos(s^{-1}A)B$.*

```
1  $V = B$, $Z = B$
2  for $k = 1\colon m$
3      $\beta = 2k$, $\gamma = 2k-1$, $q = 1/(2k+1)$
4      $B = \widetilde{A}B$
5      $B = (\widetilde{A}B)\left(s^2\beta\gamma\right)^{-1}$
6      $V = V + (-1)^k B$
7      $Z = Z + (-1)^k qB$
8  end
9  $C = \cos(\mu/s)V - s^{-1}\sin(\mu/s)\widetilde{A}Z$
```

The recovery of $\sin(s^{-1}A)S_{s-1}$ (recall (2.4)) can be obtained by a single execution of Code Fragment 4.1 for $V = Z = S_{s-1}$ after setting $\gamma = 2k+1$ and $q = 2k+1$ in line 3. Thus, $\sin(s^{-1}A)S_{s-1} \approx s^{-1}\cos(\mu/s)\widetilde{A}V + \sin(\mu/s)Z$. Comparing Code Fragment 2.1 with Code Fragment 4.1, assuming the same scaling parameter s, undoing the shift requires n_0 matrix–vector products for each $k = 1\colon s$ bringing the total of the extra cost to $(s+1)n_0$ matrix–vector products: sn_0 for C_s and n_0 to recover $\sin(A)B$ from $\sin(s^{-1}\widetilde{A})S_{s-1}$ using (2.4) and the formula in (4.1). However, the scaling parameter s selection based on $\widetilde{A}$ is potentially smaller than the selection based on A, making the overall cost of the algorithm potentially smaller.

For the early termination of the evaluation of Taylor polynomials, we use the criterion proposed by Al-Mohy and Higham [2, eq. (3.15)] implemented in line 34 of Algorithm 5.1 below.

5. Algorithm.

In this section we write in detail our algorithm for computing the trigonometric and hyperbolic matrix functions of the forms $f(tA)B$ and $f(tA^{1/2})B$.

Algorithm 5.1 ($[C, S]$ = `funmv`$(t, A, B, \text{tol}, $`option`$)$). *Given $t \in \mathbb{C}$, $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times n_0}$, and a tolerance* tol*, this algorithm computes C and S for any chosen option of the table. The parameters σ, k_0, and* shift *are set to their corresponding values of the last column depending on the chosen case.*

Option	Outputs		$(\sigma, k_0, \text{shift})$
1	$C \approx \cos(tA)B$	$S \approx \sin(tA)B$	$(1, 1, 1)$
2	$C \approx \cosh(tA)B$	$S \approx \sinh(tA)B$	$(1, 0, 1)$
3	$C \approx \cos(tA)B$	$S \approx \operatorname{sinc}(tA)B$	$(1, 1, 0)$
4	$C \approx \cosh(tA)B$	$S \approx \operatorname{sinch}(tA)B$	$(1, 0, 0)$
5	$C \approx \cos(tA^{1/2})B$	$S \approx \operatorname{sinc}(tA^{1/2})B$	$(\frac{1}{2}, 1, 0)$
6	$C \approx \cosh(tA^{1/2})B$	$S \approx \operatorname{sinch}(tA^{1/2})B$	$(\frac{1}{2}, 0, 0)$

```
1  if shift, $\mu = \text{trace}(A)/n$, $A = A - \mu I$, end
2  if $t\|A\|_1 = 0$
```

```
  $m_* = 0$, $s = 1$   % the case $tA = 0$.
else
  $[m_*, s, \Theta_\sigma] = \mathrm{parameters}(t^{1/\sigma}A, \sigma, \mathrm{tol})$ % Code Fragment 3.1
end
undoin $= 0$, undout $= 0$   % undo shifting inside or outside the loop.
if option 1 and $|\mathrm{Im}(t\mu)| > 0$
  $\phi_1 = \cos(t\mu/s)$, $\phi_2 = \sin(t\mu/s)$, undoin $= 1$
elseif option 1 and $t\mu \in \mathbb{R}\backslash\{0\}$
  $\phi_1 = \cos(t\mu)$, $\phi_2 = \sin(t\mu)$, undout $= 1$
elseif option 2 and $|\mathrm{Re}(t\mu)| > 0$
  $\phi_1 = \cosh(t\mu/s)$, $\phi_2 = \sinh(t\mu/s)$, undoin $= 1$
elseif option 2 and $t\mu \in \mathbb{C}\backslash\mathbb{R}$
  $\phi_1 = \cosh(t\mu)$, $\phi_2 = \sinh(t\mu)$, undout $= 1$
end
$C_0 = 0 \in \mathbb{R}^{n\times n_0}$
if $2|s$, $C_0 = B/2$, end
$S = C_0$, $C_1 = B$
for $i = 1{:}\, s+1$
    if $i = s+1$
      $S = 2S$, $C_1 = S$
    end
    $V = C_1$, $Z = C_1$, $B = C_1$
    $c_1 = \|B\|_\infty$
    for $k = 1{:}\, m_*$
        $\beta = 2k$
        if $i \le s$, $\gamma = \beta - 1$, $q = 1/(\beta+1)$ else $\gamma = \beta + 1$, $q = \gamma$, end
        if $\sigma = 1$, $B = AB$, end
        $B = (AB)\big((t/s)^2/(\beta\gamma)\big)$
        $c_2 = \|B\|_\infty$
        $V = V + (-1)^{k_0 k}B$
        if undoin, $Z = Z + ((-1)^{k_0 k}q)B$, end
        if $c_1 + c_2 \le \mathrm{tol}\|V\|_\infty$, break, end
        $c_1 = c_2$
    end
    if undoin
      if $i \le s$
        $V = V\phi_1 + A(Z((-1)^{k_0}t\phi_2/s))$
      else
        $V = A(V(t\phi_1/s)) + Z\phi_2$
      end
    end
    if $i = 1$, $C_2 = V$, elseif $i \le s$, $C_2 = 2V - C_0$, end    % using (2.2).
    if $i \le s-1$ and ($2|s$ xor $2|i$)
      $S = S + C_2$    % using (2.6).
    end
    $C_0 = C_1$, $C_1 = C_2$
end
$C = C_2$
if undoin
```

```
52      $S = V$
53  elseif option 1 or option 2
54      $S = A(V(t/s))$
55  else
56      $S = V/s$
57  end
58  if undout
59      $C = \phi_1 C + ((-1)^{k_0}\phi_2)S$
60      $S = \phi_1 S + \phi_2 C_2$
61  end
```

Due to the stopping criterion in line 34, assume that the inner loop is terminated when k takes values m_i, $i = 1\colon s+1$. Thus, the total cost of the algorithm is

$$2\sigma n_0 \sum_{i=1}^{s+1} m_i + (\texttt{undoin}) n_0 (s+1) + (\text{shift} - \texttt{undoin}) n_0 + \Theta_\sigma \tag{5.1}$$

matrix–vector multiplications. Since m_i and Θ_σ are bounded by m_* and (3.9), respectively, an upper bound of the computational cost of the algorithm can be obtained after the execution of Code Fragment 3.1 in line 5. This advantage allows users to estimate the overhead of the algorithm.

Next we discuss an important point that the matrix $A^{1/2}$ might be available or easy to compute in some applications; for instance, A could be diagonal. Assume we want to compute $\cos(tA^{1/2})b$ and $\text{sinc}(tA^{1/2})b$. Thus, `funmv` can be implemented in two settings: `funmv`$(t, A^{1/2}, b)$ (option 3) or `funmv`(t, A, b) (option 5). We analyze the computational cost of those two implementations, where the selection of σ plays the important role. Note that the parameter $\Theta_\sigma > 0$ if Code Fragment 3.1 is forced to estimate $\alpha_p(A^\sigma)$. Consequently, both implementations yield the same scaling parameter s. However, in view of the computational cost of $\alpha_p(A^\sigma)$ in (3.9) and the computational cost of the algorithm in (5.1), `funmv`$(t, A^{1/2}, b)$ (option 3) costs almost twice the cost of `funmv`(t, A, b) (option 5). An example is instructive. Take $A = \text{diag}(1, 2, \ldots, 100)$, $b = [1, 1, \ldots, 1]^T$, and $t = 10$. `funmv`(t, A, b) (option 5) selects $s = 11$ and requires 308 matrix–vector products of which $\Theta_{1/2} = 20$, whereas `funmv`$(t, A^{1/2}, b)$ (option 3) selects $s = 11$ and requires 618 matrix–vector products of which $\Theta_1 = 42$.

Note that it is possible to recover $\sin(tA)B$ and $\sinh(tA)B$ from $\text{sinc}(tA)B$ and $\text{sinch}(tA)B$ in options 3 and 4, respectively, since no shift is performed. However, $\sin(tA^{1/2})B$ and $\sinh(tA^{1/2})B$ cannot be recovered from $\text{sinc}(tA^{1/2})B$ and $\text{sinch}(tA^{1/2})B$ in options 5 and 6, respectively, because that requires the computation of the matrix $A^{1/2}$.

6. Numerical experiments.

In this section we perform some numerical tests to illustrate the accuracy and efficiency of Algorithm 5.1. We use MATLAB R2015a on a machine with Core i7. But for experiments where CPU time is important, we limit MATLAB to a single processor. The experiments involve the following algorithms:

1. `funmv`: the MATLAB code of Algorithm 5.1.
2. `trigmv` and `trighmv`: MATLAB codes implementing the recently authored algorithm by Higham and Kandolf [19, Alg. 3.2]. `trigmv` returns the actions $\cos(A)b$ and $\sin(A)b$ while `trighmv` returns the actions $\cosh(A)b$ and $\sinh(A)b$. The codes are available at https://bitbucket.org/kandolfp/trigmv.
3. `expmv`: MATLAB code for the algorithm of Al-Mohy and Higham [2, Alg. 3.2]

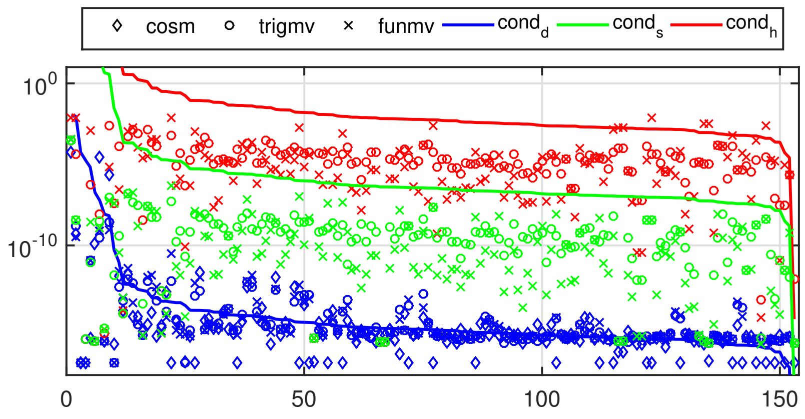

FIG. 6.1. *Experiment* 1. *Normwise relative errors in computing* $\cos(A)b$ *using different precisions.* cond_d*,* cond_s*, and* cond_h *represent* $\mathrm{cond}(\cos, A)$ *multiplied by* 2^{-53}*,* 2^{-24}*, and* 2^{-10}*, respectively.*

computing the action of the matrix exponential $e^A b$. The code is available at https://github.com/higham/expmv.

4. `cosm` and `sinm`: [3, Alg's 4.2 and 5.2] of Al-Mohy, Higham, and Relton for explicitly computing $\cos A$ and $\sin A$, respectively. The multiplication by b follows to obtain $\cos(A)b$ or $\sin(A)b$. The MATLAB codes of the algorithms are available at https://github.com/sdrelton/cosm_sinm.

Experiment 1. In this experiment we test the stability of `funmv` (option 1 and option 2) comparing with `trigmv`, `trighmv`, and `cosm`. We use the test matrices described in [1, sect. 6] and [2, sect. 6]. For each matrix A of these test matrices, a vector b is randomly generated. We approximate $x := \cos(A)b$ by $\widehat{x}$ using `funmv` (option 1) and `trigmv` with the tolerances of half, single, and double precisions. The approximation of x by `cosm` is carried out in double precision since the algorithm is only intended for that. The "exact" x is computed at 100 digit precision with the Symbolic Math Toolbox. The relative forward errors $\|x - \widehat{x}\|_2/\|x\|_2$ for each tolerance is plotted in Figure 6.1, where the solid lines represent the condition number of the matrix cosine, $\mathrm{cond}(\cos, A)$, multiplied by the associate tolerance tol and sorted in descending order. The condition number with respect to the Frobenius norm is estimated using the code `funm_condest_fro` from the Matrix Function Toolbox [16].

Figure 6.2 displays a performance profile [6] using the code from [15, sect. 26.4], to which we apply the strategy from [5] to avoid skewing the results by tiny relative errors smaller than the unit roundoff. The profile presents the double precision data plotted in Figure 6.1 which includes the data of `cosm`. For each method, the parameter p is the proportion of problems in which the error is within a factor of α of the smallest error over all methods. The experiment reveals that our algorithm behaves as stable as the existing algorithms. The performance profile shows that `cosm` produces better accuracy in general while `funmv` and `trigmv` have similar behavior.

We repeat the experiment for $\cosh(A)b$ using `funmv` (option 2), `trighmv`, and `cosm` with argument iA. The results are reported in Figures 6.3 and 6.4. All the algorithms behave in a stable manner, but the performance profile sorts the three algorithms based on the accuracy whereby the higher the curve, the better the accuracy in general.

The figures corresponding to $\sin(A)b$ and $\sinh(A)b$ are quite similar to those of $\cos(A)b$ and $\cosh(A)b$, respectively; which is why we don't report them here.

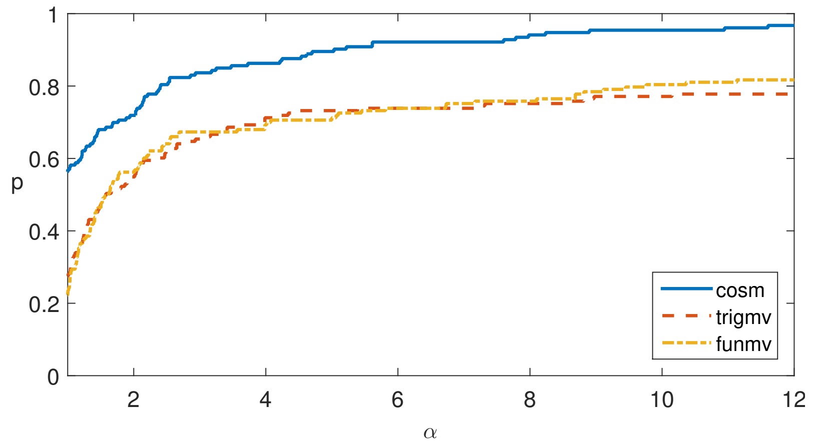

FIG. 6.2. *Double precision data of Figure* 6.1 *presented as a performance profile.*

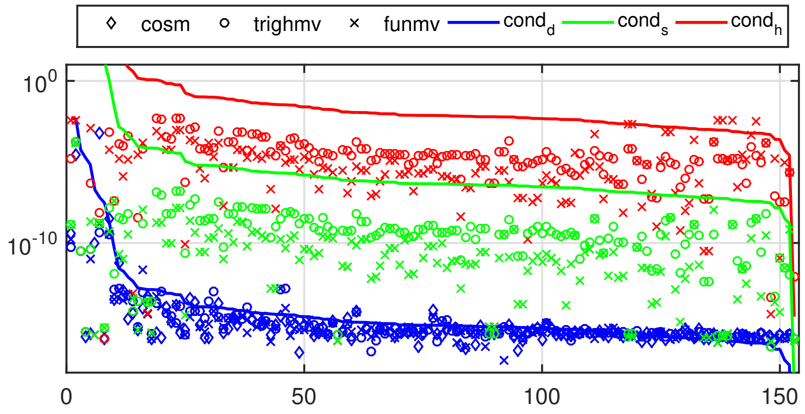

FIG. 6.3. *Experiment* 1. *Normwise relative errors in computing* $\cosh(A)b$ *using different precisions.* cond_d, cond_s, *and* cond_h *represent* $\text{cond}(\cosh, A)$ *multiplied by* 2^{-53}, 2^{-24}, *and* 2^{-10}, *respectively.*

In addition, we have tried options 3 and 4 of `funmv` to compute $\cos(A)b$ and $\cosh(A)b$, respectively. The stability behaviors hold but the performance profile for the relative errors related to $\cos(A)b$ shows that `trigmv` produces better accuracy in general whereas the profile for $\cosh(A)b$ still shows that `funmv` (option 4) produces better accuracy than `trighmv`. However, options 3 and 4 cater for particular settings where sinc and sinch are required. Thus, `funmv` wouldn't benefit from the preprocessing and more computational cost is expected. Hence, options 1 and 2 are the best settings for `funmv` if the required outputs are those obtained by `trigmv` and `trighmv`, respectively.

Experiment 2. In this experiment we compare `funmv` with `trigmv` and `trighmv` in terms of CPU time, the number of matrix–vector products, and 1-norm relative forward errors. We use two problem sets of real test matrices of dimensions between 900 and 1.3×10^6.

Set 1 is

- `orani678` (nonsymmetric), $n = 2529$, $b = [1, 1, \ldots, 1]^T$;

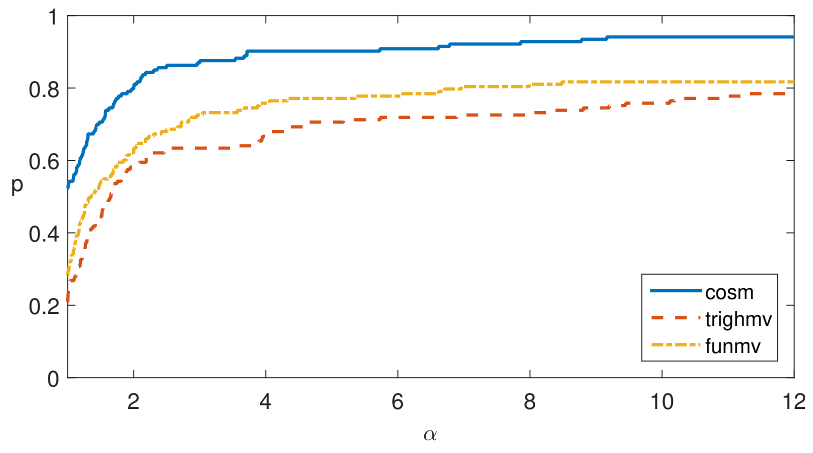

FIG. 6.4. *Double precision data of Figure* 6.3 *presented as a performance profile.*

- `bcspwr10` (symmetric), $n = 5300$, $b = [1, 0, \dots, 0, 1]^T$;
- `gr_30_30`, $n = 900$, $b = [1, 1, \dots, 1]^T$;
- `triw` denotes `-gallery('triw',2000,4)` of MATLAB, upper triangular with -1 in the main diagonal and -4 elsewhere, $n = 2000$, $b = [\cos 1, \cos 2, \dots, \cos n]^T$.

Some of the test matrices are described in [19, Example 4.2] and [2, Experiment 5]. The first three matrices belong to the Harwell–Boeing collection and are obtained from the University of Florida Sparse Matrix Collection [4]. Since these matrices are of medium size, we were able to use `cosm` and consider its approximation of $\cos(A)b$ as a reference solution in double precision.

Set 2 consists of large and sparse matrices. They are also taken from the University of Florida Sparse Matrix Collection [4] and described there as follows.

- `poisson3Db` (nonsymmetric), $n = 85623$, $\text{nnz}(A) = 2374949$, belongs to FEMLAB, a finite element method toolbox for MATLAB developed by COMSOL.
- `cage13` (nonsymmetric), $n = 445315$, $\text{nnz}(A) = 7479343$, arises form the cage model of DNA electrophoresis [26].
- `ecology1` (symmetric), $n = 10^6$, $\text{nnz}(A) = 4996000$, comes from landscape ecology problem, using electrical network theory to model animal movement and gene flow.
- `atmosmodd` (nonsymmetric), $n = 1270432$, $\text{nnz}(A) = 8814880$, arises in the numerical weather prediction and atmospheric modeling.

We take $b = [1, 0, \dots, 0, 1]^T$ with all of these matrices.

The results are shown in Table 6.1. Given A and b of Set 1 and a scalar t, Table 6.1(a) displays CPU time and the number of matrix–vector products, mv, for $\cos(tA)b$ and $\sin(tA)b$ simultaneously as computed by `funmv` (option 1) and by `trigmv`. The reported relative errors belong to $\cos(tA)b$. Similarly, Table 6.1(b) presents the results for $\cosh(tA)b$ and $\sinh(tA)b$ simultaneously as computed by `funmv` (option 2) and by `trighmv`. The presented relative errors belong to $\cosh(tA)b$. We show CPU time for `cosm` in both tables. In Table 6.1(c), we use A and b of Set 2 and a scalar t. The vectors $\cos(tA)b$ and $\sin(tA)b$ are simultaneously computed in single precision by `funmv` (option 1) and by `trigmv`. The relative errors reported here are calculated for the vectors $\sin(tA)b$ with reference solutions computed by `trigmv` in double precision.

Table 6.1
Experiment 2: *Behavior of* `funmv` *versus* `trigmv` *and* `trighmv`.

(a) The computation of $\cos(tA)b$ and $\sin(tA)b$ in double precision.

		`funmv` (option 1)			`trigmv`			`cosm(A)b`
	t	Time	mv	Error	Time	mv	Error	Time
`orani678`	100	2.1e-1	1111	1.0e-14	3.4e-1	2024	2.0e-15	2.3e2
`bcspwr10`	10	3.6e-2	379	3.6e-14	6.1e-2	618	3.6e-14	2.5e2
`gr_30_30`	2	4.7e-3	133	6.2e-14	7.4e-3	188	7.8e-14	2.4
`triw`	10	9.5e1	27005	7.1e-14	1.2e2	56560	1.4e-13	3.7e1

(b) The computation of $\cosh(tA)b$ and $\sinh(tA)b$ in double precision.

		`funmv` (option 2)			`trighmv`			`cosm(iA)b`
	t	Time	mv	Error	Time	mv	Error	Time
`orani678`	100	2.2e-1	1129	8.4e-15	3.1e-1	2026	4.8e-15	2.3e2
`bcspwr10`	10	3.9e-2	402	1.1e-15	6.0e-2	632	2.5e-15	2.5e2
`gr_30_30`	2	4.8e-3	129	1.6e-13	6.9e-3	166	8.6e-14	2.4
`triw`	10	9.7e1	28662	1.2e-12	1.2e2	56298	1.4e-13	4.3e1

(c) The computation of $\cos(tA)b$ and $\sin(tA)b$ in single precision.

		`funmv` (option 1)			`trigmv`		
	t	Time	mv	Error	Time	mv	Error
`poisson3Db`	50	2.5	327	1.0e-9	2.8	388	1.3e-9
`cage13`	100	9.9	449	1.5e-9	1.5e1	666	3.4e-9
`ecology1`	10	4.8e1	1519	4.8e-8	1.5e2	2986	1.3e-8
`atmosmodd`	0.01	1.0e2	3225	4.9e-8	2.8e2	6202	8.0e-7

We can see in view of this experiment that `funmv` (options 1 and 2) always needs fewer matrix–vector products, which for some cases represent about 50% of the cost of `trigmv` or `trighmv`. Moreover, `funmv` is always faster than `trigmv` and `trighmv` but the algorithms produce similar accuracy.

Experiment 3. In this experiment we use `funmv` (option 5) to compute the combination $y(t) = \cos(tA^{1/2})b + t\,\mathrm{sinc}(tA^{1/2})z$. Note that `trigmv` cannot be used for this problem because it requires an explicit computation of possibly dense and complex $A^{1/2}$ though A is real [17, sect. 6.2]. The computation of a matrix square root is a challenging problem itself and infeasible for large scale matrices. Another difficulty is that `trigmv` cannot immediately yield $x := \mathrm{sinc}(tA^{1/2})b$, yet x requires solving the system $A^{1/2}\,x = \sin(A^{1/2})b$.

Thus, we invoke our algorithm for the matrix $B = [b, z]$. The combination above can be viewed as an exact solution of the system (1.1) with $g \equiv 0$, $y(0) = b$, and $y'(0) = z$. We compare the approximation of $y(t)$ using our algorithm with that obtained from the formula

$$\exp\left(t\begin{bmatrix} 0 & I \\ -A & 0 \end{bmatrix}\right)\begin{bmatrix} b \\ z \end{bmatrix} = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}, \tag{6.1}$$

which is a particular case of the expression given in [17, Prob. 4.1]; see also [19, eq. (1.1)]. We use the `expmv` algorithm of Al-Mohy and Higham to evaluate the left-hand side of (6.1). The approximation of $y(t)$ is obtained by reading off the upper half of the resulting vector. We use the problems A and b as prescribed in Experiment 2. For Set 1 we invoke the algorithms in double precision and compute reference solutions using the MATLAB function `expm` applied to the left-hand side of (6.1). However, for Set

TABLE 6.2
Experiment 3: *Behavior of* `funmv` *versus* `expmv`.

(a) The computation of $\cos(tA^{1/2})b + t\,\mathrm{sinc}(tA^{1/2})z$ in double precision.

		`funmv` (option 5)			`expmv`			`expm`
	t	Time	mv	Error	Time	mv*	Error	Time
`orani678`	100	1.6e-1	920	3.2e-14	2.7e-1	1023	3.2e-14	1.1e2
`bcspwr10`	10	2.2e-2	190	4.5e-15	5.6e-2	308	4.4e-15	3.6e2
`gr_30_30`	2	3.6e-3	86	2.0e-15	5.6e-3	90	1.7e-15	1.2
`triw`	10	3.7	1694	3.3e-14	9.6	2072	3.9e-14	8.5e1

(b) The computation of $\cos(tA^{1/2})b + t\,\mathrm{sinc}(tA^{1/2})z$ in half precision.

		`funmv` (option 5)			`expmv`		
	t	Time	mv	Error	Time	mv*	Error
`poisson3Db`	50	1.9	268	1.6e-7	3.0	318	8.8e-6
`cage13`	100	7.8	356	8.6e-5	1.4e1	454	4.5e-5
`ecology1`	10	1.2e1	426	3.8e-8	1.9e1	524	7.1e-8
`atmosmodd`	0.01	1.3	28	1.6e-7	2.3e1	431	3.9e-6

2 we run the algorithms in half precision. For reference solutions, we invoke `expmv` in double precision. We take $z = [\sin 1, \sin 2, \dots, \sin n]^T$ for the matrices of both sets.

Like `funmv`, the function `expmv` benefits from the optimization of matrix–vector multiplication whenever available. However, these algorithms are not readily comprehensive enough to detect and exploit structures of input matrices. In spite of that, let's assume for this numerical experiment that `expmv` is able to exploit the structure of the block matrix in (6.1). Thus, the action of the block matrix on a block vector $[b_1, b_2]^T$, where $b_j \in \mathbb{C}^n$, requires only the computation of Ab_1. Therefore, we consider the number of matrix–vector products mv* counted by the code `expmv` as a lower bound for the computational cost of the algorithm.

Table 6.2 presents the results and reveals that the implementation of `funmv` is more efficient than `expmv` since mv $<$ mv* for every problem. In addition, `funmv` is faster than `expmv` as it needs less CPU time to compute the expression.

7. Concluding remarks.

The algorithm we developed here has direct applications to solving second order systems of ODE's and their trigonometric numerical schemes. A single invocation of Algorithm 5.1 for inputs h, A, and $B = [y_n, y_n', \widehat{g}(y_n)]$ returns the six vectors $\cos(hA^{1/2})y_n$, $\cos(hA^{1/2})y_n'$, $\cos(hA^{1/2})\widehat{g}(y_n)$, $\mathrm{sinc}(hA^{1/2})y_n$, $\mathrm{sinc}(hA^{1/2})y_n'$, and $\mathrm{sinc}(hA^{1/2})\widehat{g}(y_n)$ that make up the vectors y_{n+1} and y_{n+1}' in the scheme (1.5)–(1.6). The evaluation of this scheme draws our attention back to the end of section 3. Since the algorithm has to be executed repeatedly for a fixed matrix A and different B and perhaps different scalar h, it is recommended to precompute the matrix M (3.12) and provide it as an external input to reduce the cost of the whole computation.

Algorithm 5.1 has several features. First, it computes the action of the composition $f(tA^{1/2})B$ without explicitly computing $A^{1/2}$. Second, it returns results in a finite number of steps that can be predicted before executing the main phase of the algorithm. Third, the algorithm is easy to implement and works for any matrix, and the only external parameter that controls the computation is tol. Fourth, the algorithm spends most of its effort on multiplying A by vectors. Thus, it fully benefits from the sparsity of A and fast implementation of matrix multiplication. Fifth, we can use Algorithm 5.1 (option 2) to compute the action of the matrix exponential since $e^A B = \cosh(A)B + \sinh(A)B$. Finally, though we derive the values of θ_m in (3.5) for half, single, and double precisions, θ_m can be evaluated for any arbitrary precision. Thus, Algorithm 5.1 can be extended to be a multiprecision algorithm as in [8] since the function ρ_m (3.2) has an explicit expression that is easily handled by optimization software. All of these features make our algorithm attractive for black box use in a wide range of applications.

Our MATLAB codes are available at https://github.com/aalmohy/funmv.

Acknowledgments. I thank King Khalid University for supporting this research during my sabbatical leave. I also thank the University of Manchester for making their electronic resources available to me. Many thanks to the referees for their helpful comments and suggestions that helped to improve the presentation of this paper.

REFERENCES

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

[2] A. H. Al-Mohy and N. J. Higham, *Computing the action of the matrix exponential, with an application to exponential integrators*, SIAM J. Sci. Comput., 33 (2011), pp. 488–511, https://doi.org/10.1137/100788860.

[3] A. H. Al-Mohy, N. J. Higham, and S. D. Relton, *New algorithms for computing the matrix sine and cosine separately or simultaneously*, SIAM J. Sci. Comput., 37 (2015), pp. A456–A487, https://doi.org/10.1137/140973979.

[4] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), 1.

[5] N. J. Dingle and N. J. Higham, *Reducing the influence of tiny normwise relative errors on performance profiles*, ACM Trans. Math. Softw., 39 (2013), 24.

[6] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Programming, 91 (2002), pp. 201–213.

[7] E. Estrada, D. J. Higham, and N. Hatano, *Communicability and multipartite structures in complex networks at negative absolute temperatures*, Phys. Rev. E, 78 (2008), 026102.

[8] M. Fasi and N. J. Higham, *Multiprecision Algorithms for Computing the Matrix Logarithm*, MIMS EPrint 2017.16, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2017.

[9] L. Gauckler, J. Lu, J. L. Marzuola, F. Rousset, and K. Schratz, *Trigonometric Integrators for Quasilinear Wave Equations*, preprint, https://arxiv.org/abs/1702.02981, 2017.

[10] V. Grimm and M. Hochbruck, *Error analysis of exponential integrators for oscillatory second-order differential equations*, J. Phys. A: Math. Gen., 39 (2006), pp. 5495–5507.

[11] V. Grimm and M. Hochbruck, *Rational approximation to trigonometric operators*, BIT, 48 (2008), pp. 215–229.

[12] E. Hairer and C. Lubich, *Long-time energy conservation of numerical methods for oscillatory differential equations*, SIAM J. Numer. Anal., 38 (2000), pp. 414–441, https://doi.org/10.1137/S0036142999353594.

[13] E. Hairer, C. Lubich, and G. Wanner, *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer Ser. Comput. Math. 31. Springer Science & Business Media, Berlin, 2006.

[14] G. I. Hargreaves and N. J. Higham, *Efficient algorithms for the matrix cosine and sine*, Numer. Algorithms, 40 (2005), pp. 383–400.

[15] D. J. Higham and N. J. Higham, *MATLAB Guide*, 3rd ed., SIAM, Philadelphia, 2017.

[16] N. J. Higham, *The Matrix Function Toolbox*, http://www.maths.manchester.ac.uk/~higham/mftoolbox.

[17] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008, https://doi.org/10.1137/1.9780898717778.

[18] N. J. Higham and A. H. Al-Mohy, *Computing matrix functions*, Acta Numer., 19 (2010), pp. 159–208.

[19] N. J. Higham and P. Kandolf, *Computing the action of trigonometric and hyperbolic matrix functions*, SIAM J. Sci. Comput., 39 (2017), pp. A613–A627, https://doi.org/10.1137/16M1084225.

[20] N. J. Higham and M. I. Smith, *Computing the matrix cosine*, Numer. Algorithms, 34 (2003), pp. 13–26.

[21] 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.

[22] L. Jódar, E. Navarro, A. E. Posso, and M. C. Casabán, *Constructive solution of strongly coupled continuous hyperbolic mixed problems*, Appl. Numer. Math., 47 (2003), pp. 447–492.

[23] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*, Chapman & Hall/CRC, Boca Raton, FL, 2003.

[24] 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.

[25] 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.

[26] A. van Heukelum, G. T. Barkema, and R. H. Bisseling, *DNA electrophoresis studied with the cage model*, J. Comput. Phys., 180 (2002), pp. 313–326.

[27] X. Wu, K. Liu, and W. Shi, *Structure-Preserving Algorithms for Oscillatory Differential Equations* II, Springer, Heidelberg; Science Press Beijing, Beijing, 2015.