A New Algorithm for Computing the Actions of Trigonometric and Hyperbolic Matrix Functions

Al-Mohy, Awad H.

2017

MIMS EPrint: **2017.29**

ISSN 1749-9097

A NEW ALGORITHM FOR COMPUTING THE ACTIONS OF TRIGONOMETRIC AND HYPERBOLIC MATRIX FUNCTIONS*

AWAD H. AL-MOHY†

Abstract. A new algorithm is derived for computing the actions $f(tA)B$ and $f(tA^{1/2})B$, where f is cosine, sinc, sine, hyperbolic cosine, hyperbolic sinc, or hyperbolic sine function. A is an $n \times n$ matrix and B is $n \times n_0$ with $n_0 \ll n$. $A^{1/2}$ denotes any matrix square root of A and it is never required to be computed. 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 $f(s^{-1}tA)B$ by a truncated Taylor series, and finally uses the recurrences of the Chebyshev polynomials of the first and second kind to recover $f(tA)B$. The selection of the scaling parameter and the degree of Taylor polynomial are based on a forward error analysis and a sequence of the form $\|A^k\|^{1/k}$ in such a way the overall computational cost of the algorithm is optimized. 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

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 [8, 23]

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

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

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

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

*Version of August 28, 2017

†Department of Mathematics, King Khalid University, Abha, Saudi Arabia (ahalmohy@kku.edu.sa, http://www.maths.manchester.ac.uk/~almohy).

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

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

This is important to clear any ambiguity that a square root of A is needed. We will see below how the actions of $\cos(tA^{1/2})$ and $\mathrm{sinc}(tA^{1/2})$ 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

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

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

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 [7, sect. 2], [8, sect. 2], or [11, sect. XIII.2.2]. Many filter functions are proposed in literature and most of them involve several actions of $\mathrm{sinc}(hA^{1/2})$ to evaluate $\widehat{g}(y)$. For example Hairer and Lubich [10] chose $\psi = \mathrm{sinc}$ and $\phi = 1$ while Griman and Hochbruck proposed $\psi = \mathrm{sinc}^2$ and $\phi = \mathrm{sinc}$ [8].

The system (1.1) arises from semidiscretization of some second order PDE's by finite difference or finite elements methods [21]. The hyperbolic matrix functions: $\cosh A$, $\sinh A$, and $\mathrm{sinch}\,A$, where $\mathrm{sinch}\,A = \mathrm{sinc}(\mathrm{i}A)$, arise in the solution of coupled hyperbolic systems of PDE's [19]. They also have an application in communicability analysis in complex networks [5]. The matrix A is usually large and sparse, so finding methods to compute the action of these matrix functions on vectors are so crucial to reduce computational cost.

The computation of the action of the matrix exponential has received significant research attention; see [2] and the references therein. However it is not the case for trigonometric and hyperbolic matrix functions. A 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) below. Grimm and Hochbruck [9] proposed the use of a rational Krylov subspace method instead of the standard one for certain problems to compute $\cos(tA^{1/2})b$ and $\mathrm{sinc}(tA^{1/2})b$. Recently, Higham and Kandolf [16] derived an algorithm to compute 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 evaluation of $\cos(A)B$ and $\sin(A)B$ (or $\cosh(A)B$ and $\sinh(A)B$) requires the action of e^A on the matrix $[B, B]/2 \in \mathbb{C}^{n\times 2n_0}$.

The calculation of $\cos A$ and $\sin A$ for dense A of medium size is will-studied. Serbin and Blalock [22] 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 [17] uses the [8/8] Padé approximant with the aid of a forward error analysis to specify the scaling parameter s. Hargreaves and Higham [12] develop an algorithm with a variable choice of the degree of Padé approximants based on forward error bounds in such a way the computational cost is minimized. They also derive an algorithm that computes

$\cos A$ and $\sin A$ simultaneously. Recently, Al-Mohy et al. [3] derive 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 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 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 cannot compute $f(tA^{1/2})B$ without explicitly computing $A^{1/2}$, which is completely impractical. Moreover, their algorithm cannot immediately 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 are 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

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

Let $T_k(X, B) = \cos(kX)B$ and simply denote it by T_k, where $k \geq 0$. Thus we obtain the three term recurrence

$$T_k + T_{k-2} = 2\cos(X)T_{k-1} = 2\,T_1(X, T_{k-1}), \quad k \geq 2. \tag{2.2}$$

Observe that (2.2) is the recurrence that generates *Chebyshev polynomials of the first kind for* $T_0 = 1$ and $T_1 = x$ [20]. The heaviest computational work in the recurrence (2.2) lies in $T_1(X, T_{k-1})$ 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

$$T_1(s^{-1}A, T_{k-1}) = \cos(s^{-1}A)T_{k-1} \approx r(s^{-1}A)T_{k-1}.$$

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

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

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      β = 2k, γ = 2k − 1
4      B = AB
5      B = (AB) (s^2 βγ)^{-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 only need to delete line 4 of Code Fragment 2.1.

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

$$U_k - U_{k-2} = 2T_k, \quad k \geq 2, \quad U_0 = B, \quad U_1 = 2T_1 = 2\cos(X)B. \tag{2.3}$$

It is the recurrence that yields *the Chebyshev polynomials of the second kind* [20]. By induction on k, it easy to verify that

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

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

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

Since (2.4) holds for every X we conclude that

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

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

$$\frac{1}{2}U_{s-1} = \begin{cases} T_1 + T_3 + T_5 \cdots + T_{s-1}, & \text{if } s \text{ is even,} \\ \frac{1}{2}T_0 + T_2 + T_4 + \cdots + T_{s-1}, & \text{if } s \text{ is odd,} \end{cases} \tag{2.6}$$

which can be easily derived from (2.3).

Given the relations between trigonometric and hyperbolic functions, we can 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

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

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

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

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 [15, 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 T_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), U_{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 $V = U_{s-1}$ 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 an 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 is m, 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 don't effect the optimization for the value of m_*. Thus we consider the sequence

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

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 don't need to consider the case $p = 1$ when minimizing $C_m(A^\sigma)$ since $C_m(A^\sigma) \le m\lceil \alpha_1(A^\sigma)/\theta_m \rceil$. 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

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

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

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

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 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 [18]. 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 is 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 T_s and $\widetilde{r}_m(s^{-1}A^\sigma)U_{s-1}$ 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 counted. 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    $m_* = \mathrm{argmin}_{1\le m\le m_{\max}}\, m\lceil \|A\|_1^\sigma/\theta_m \rceil$
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      $m_* = \mathrm{argmin}_{1\le m\le m_{\max}}\, m\lceil d_2/\theta_m \rceil$
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 $s = \max\big(C_{m_*}(A^\sigma)/m_*, 1\big)$
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 $S \in \mathbb{R}^{(p_{\max}-1)\times m_{\max}}$ given by

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

and then for each t obtain $C_{m_*}(tA^\sigma)$ as the smallest nonzero element in the matrix $\lceil |t|S \rceil \mathrm{diag}(1, 2, \dots, m_{\max})$, 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.

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	1.8e0	4.2e0	6.9e0	9.7e0	1.3e1	1.5e1	1.8e1	2.1e1	2.4e1	2.7e1	3.0e1	3.3e1
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

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 that the smaller the s the better the stability of the algorithm in general, and the lower the computational cost. That 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 have found empirically that the shift $\mu = n^{-1}\mathrm{trace}(A)$ [14, Thm. 4.21] that minimizes the Frobenius norm of the matrix $\widetilde{A} = A - \mu I$ leads to smaller values of $\alpha_p(\widetilde{A})$ 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 $\mathrm{sinc}(A)B$, $\mathrm{sinch}(A)B$, 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 $|\mathrm{Re}(\mu)|$. Same problem arises for $\cos\mu$ and $\sin\mu$ if $|\mathrm{Im}(\mu)|$ 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 $\cos(s^{-1}\widetilde{A})T_{k-1}$ for each k in the recurrence (2.2) using the formula in (4.1), which requires $\sin(s^{-1}\widetilde{A})T_{k-1}$. The next code shows how $\sin(s^{-1}\widetilde{A})T_{k-1}$ 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 $\quad \beta = 2k$, $\gamma = 2k-1$, $q = 1/(2k+1)$
4 $\quad B = \widetilde{A}B$
5 $\quad B = (\widetilde{A}B)\left(s^2\beta\gamma\right)^{-1}$
6 $\quad V = V + (-1)^k B$
7 $\quad Z = Z + (-1)^k qB$
8 end
9 $C = \cos(\mu/s)V - s^{-1}\sin(\mu/s)\widetilde{A}Z$

The recovery of $\sin(s^{-1}A)U_{s-1}$ (recall (2.4)) can be obtained by a single execution of Code Fragment 4.1 for $V = Z = U_{s-1}$ after setting $\gamma = 2k+1$ and $q = 2k+1$ in line 3. Thus $\sin(s^{-1}A)U_{s-1} \approx s^{-1}\cos(\mu/s)\widetilde{A}V + \sin(\mu/s)Z$. 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 T_s and n_0 to recover $\sin(A)B$ from $\sin(s^{-1}\widetilde{A})U_{s-1}$ using (2.4) and the formula in (4.1). However, the scaling parameter s selected based on $\widetilde{A}$ is potentially smaller than that selected 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 details 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`($t, A, B,$ 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	$C \approx \cos(tA)B$	$S \approx \text{sinc}(tA)B$	$(1, 1, 0)$
4	$C \approx \cosh(tA)B$	$S \approx \text{sinch}(tA)B$	$(1, 0, 0)$
5	$C \approx \cos(tA^{1/2})B$	$S \approx \text{sinc}(tA^{1/2})B$	$(\frac{1}{2}, 1, 0)$
6	$C \approx \cosh(tA^{1/2})B$	$S \approx \text{sinch}(tA^{1/2})B$	$(\frac{1}{2}, 0, 0)$

1 if shift, $\mu = \text{trace}(A)/n$, $A = A - \mu I$, end
2 if $t\|A\|_1 = 0$
3   $m_* = 0$, $s = 1$ % The case $tA = 0$.
4 else
5   $[m_*, s, \Theta_\sigma] = \text{parameters}(t^{1/\sigma}A, \sigma, \text{tol})$ % Code Fragment 3.1
6 end
7 `undoin` $= 0$, `undout` $= 0$ % undo shifting inside or outside the loop.
8 if option 1 and $|\text{Im}(t\mu)| > 0$
9   $\phi_1 = \cos(t\mu/s)$, $\phi_2 = \sin(t\mu/s)$, `undoin` $= 1$
10 elseif option 1 and $t\mu \in \mathbb{R}\backslash\{0\}$
11   $\phi_1 = \cos(t\mu)$, $\phi_2 = \sin(t\mu)$, `undout` $= 1$
12 elseif option 2 and $|\text{Re}(t\mu)| > 0$
13   $\phi_1 = \cosh(t\mu/s)$, $\phi_2 = \sinh(t\mu/s)$, `undoin` $= 1$
14 elseif option 2 and $t\mu \in \mathbb{C}\backslash\mathbb{R}$
15   $\phi_1 = \cosh(t\mu)$, $\phi_2 = \sinh(t\mu)$, `undout` $= 1$
16 end
17 $T_0 = 0$
18 if $2|s$, $T_0 = B/2$, end
19 $U = T_0$, $T_1 = B$
20 for $i = 1\colon s+1$
21   if $i = s+1$
22       $U = 2U$, $T_1 = U$

```
23      end
24      $V = T_1$, $Z = T_1$, $B = T_1$
25      $c_1 = \|B\|_\infty$
26      for $k = 1\colon m_*$
27          $\beta = 2k$
28          if $i \le s$, $\gamma = \beta - 1$, $q = 1/(\beta + 1)$ else $\gamma = \beta + 1$, $q = \gamma$, end
29          if $\sigma = 1$, $B = AB$, end
30          $B = (AB)\big((t/s)^2/(\beta\gamma)\big)$
31          $c_2 = \|B\|_\infty$
32          $V = V + (-1)^{k_0 k} B$
33          if undoin, $Z = Z + ((-1)^{k_0 k} q)B$, end
34          if $c_1 + c_2 \le \mathrm{tol}\|V\|_\infty$, break, end
35          $c_1 = c_2$
36      end
37      if undoin
38        if $i \le s$
39          $V = V\phi_1 + A(Z((-1)^{k_0} t\phi_2/s))$
40        else
41          $V = A(V(t\phi_1/s)) + Z\phi_2$
42        end
43      end
44      if $i = 1$, $T_2 = V$, elseif $i \le s$, $T_2 = 2V - T_0$, end    % using (2.2).
45      if $i \le s - 1$ and $(2|s$ xor $2|i)$
46        $U = U + T_2$    % using (2.6).
47      end
48      $T_0 = T_1$, $T_1 = T_2$
49  end
50  $C = T_2$
51  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    $S = \phi_1 S + \phi_2 T_2$
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

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

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. When $\sigma = 1/2$, the algorithm saves about 50 percent of the computational cost comparing to the other options. Therefore it

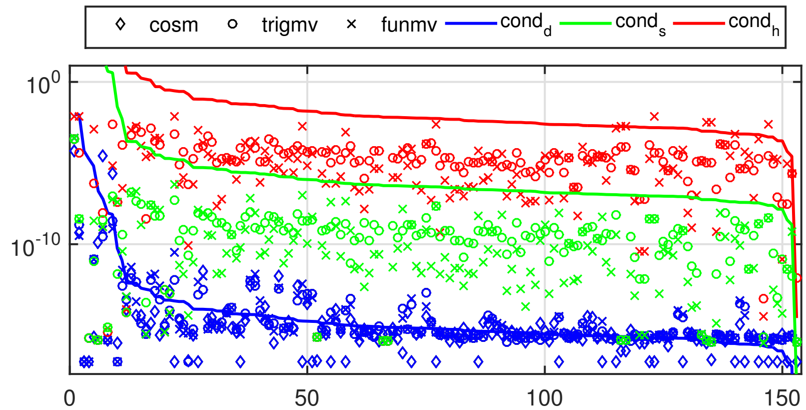

FIG. 6.1. *Experiment 1. Normwise relative errors in computing* $\cos(A)b$ *using different precisions.* cond_d, cond_s, *and* cond_h *represent* $\mathrm{cond}(\cos, A)$ *multiplied by* 2^{-53}, 2^{-24}, *and* 2^{-10}, *respectively.*

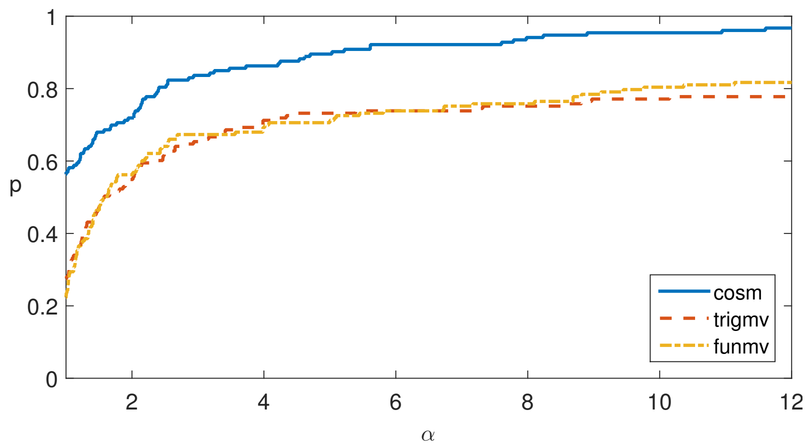

FIG. 6.2. *Double precision data of Figure 6.1 presented as a performance profile.*

is better not to provide $A^{1/2}$ even if it is easy to evaluate. As an example, take $A = \mathrm{diag}(1, 2, \cdots, 100)$, $b = [1, 1, \cdots, 1]^T$, and $t = 1$. Executing `funmv`(t, A, B) (option 5) requires 51 matrix–vector products whereas `funmv`$(t, A^{1/2}, B)$ (option 3) requires 102. Note that it is possible to obtain $\sin(tA)B$ and $\sinh(tA)B$ in options 3 and 4, respectively. However this is impossible in options 5 and 6 because of the absence of $A^{1/2}$. The present of shifting in options 1 and 2 makes it impossible to obtain $\mathrm{sinc}(tA)B$ and $\mathrm{sinch}(tA)B$ as we pointed out in the previous section.

6. Numerical experiments. In this section we give some numerical tests to illustrate the accuracy and efficiency of Algorithm 5.1. We use MATLAB® R2015a on a machine with Core i7. 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 [16, 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 in https://bitbucket.org/kandolfp/trigmv

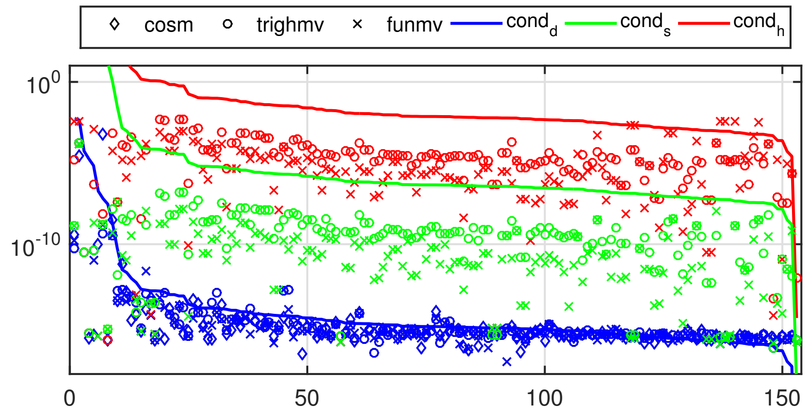

FIG. 6.3. *Experiment 1. Normwise relative errors in computing* $\cosh(A)b$ *using different precisions.* cond_d, cond_s, *and* cond_h *represent* $\mathrm{cond}(\cosh, A)$ *multiplied by* 2^{-53}, 2^{-24}, *and* 2^{-10}, *respectively.*

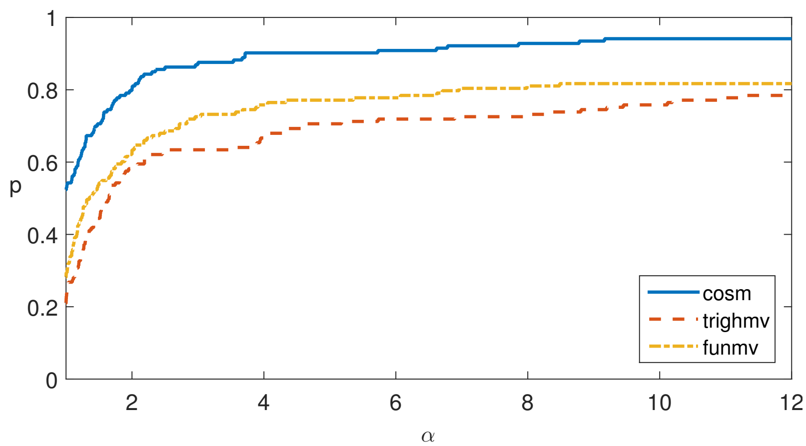

FIG. 6.4. *Double precision data of Figure 6.3 presented as a performance profile.*

3. `expmv`: MATLAB code for the algorithm of Al-Mohy and Higham [2, Alg. 3.2] that compute the action of the matrix exponential $e^A B$. The code is available in https://github.com/higham/expmv.
4. `cosm` and `sinm`: [3, Alg's 4.2 & 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 in https://github.com/sdrelton/cosm_sinm.

Experiment 1. In this experiment we test the stability of `funmv` (option 1) comparing with `trigmv` and `cosm`. We use the test matrices described in [1, sect. 6] and used also in [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` 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 $\text{cond}(\cos, A)$ multiplied by the associate tolerance tol sorted in a descending order. The condition number with respect to Frobenius norm is estimated using the code `funm_condest_fro` from the Matrix Function Toolbox [13].

Figure 6.2 displays a performance profile for 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` outperforms the other methods while `funmv` and `trigmv` have similar behavior.

We repeat the experiment for $\cosh(A)b$ using `funmv` (option 2), `trighmv`, and `cosm` with argument $\mathrm{i}A$. The results are reported in Figure 6.3 and Figure 6.4. Both methods behave in a stable manner but `funmv` outperforms `trighmv` in view of the performance profile.

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

Experiment 2. In this experiment we compute $\cos(A)b$ for large and sparse matrices. We compare `funmv` (option 1) with `trigmv` in terms of CPU time, matrix–vector products, and relative forward errors in 1-norm. We use `cosm` to compute the reference solution in double precision. The test matrices are prescribed in [16, Example 4.2] and [2, Experiment 5]. The first three matrices of Table 6.1 belong to the Harwell-Boeing collection and are obtained from the University of Florida Sparse Matrix Collection [4]. The matrix `triw` and `poisson` are from the MATLAB `gallery`. The matrices and problem details are

- `orani678` (nonsymmetric), $n = 2529$, $b = [1, 1, \cdots, 1]^T$;
- `bcspwr10` (symmetric), $n = 5300$, $b = [1, 0, \cdots, 0, 1]^T$;
- `gr_30_30`, $n = 900$, $b = [1, 1, \cdots, 1]^T$;
- `triw` denotes `-gallery('triw',2000,4)` (upper triangular with -1 in the main diagonal and -4 elsewhere), $n = 2000$, $b = [\cos 1, \cos 2, \cdots, \cos n]^T$;
- `poisson` denotes `-gallery('poisson',99)` (symmetric negative definite), $n = 9801$, $b = [\cos 1, \cos 2, \cdots, \cos n]^T$. This matrix arises from a finite difference discretization of the two–dimensional Laplacian in the unit square.

The results are shown in Table 6.1. The three blocks of the table display the computations with different tolerances that represent double, single, and half precisions. The symbol t_{ratio} denotes CPU time for method divided by CPU time for `funmv` and the symbol mv denotes the number of matrix–vector products required by each methods. Obviously `funmv` proves superiority. It does outperform `trigmv` in terms of CPU running time and computational cost. the number of matrix–vector products of `funmv` is about the half of that of `trigmv` for most cases. No wonder since `trigmv` requires the action of the matrix exponential on a matrix of two columns—namely $B = [b, b]/2$—to yield $\cos(tA)b$.

Experiment 3. In this experiment we use `funmv` (option 5) to compute the combination $y(t) = \cos(tA^{1/2})b + t\,\text{sinc}(tA^{1/2})z$. Note that `trigmv` is inapplicable for this problem because it requires an explicit computation of possibly dense $A^{1/2}$. 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 $x := \text{sinc}(tA^{1/2})b$, yet x requires solving the system $A^{1/2}\,x = \sin(A^{1/2})b$, which could be dense or ill–conditioned.

Thus we invoke our algorithm for the matrix $B = [b, z]$. The combination above

Table 6.1
Experiment 2: t_{ratio} denotes time for method divided by time for `funmv`.

(a) Double precision

	t	funmv t_{ratio}	funmv mv	funmv Error	trigmv t_{ratio}	trigmv mv	trigmv Error	cosm t_{ratio}
orani678	100	1	1111	6.0e-15	1.4	2024	4.5e-15	9.9e2
bcspwr10	10	1	379	3.8e-14	1.7	618	3.8e-14	2.5e3
gr_30_30	2	1	133	6.1e-14	1.3	188	7.8e-14	3.2e2
triw	10	1	27005	7.1e-14	1.2	56560	1.4e-13	2.2e-1
poisson	500	1	9757	4.0e-13	2.2	19036	2.2e-13	1.0e3

(b) Single precision

	t	funmv t_{ratio}	funmv mv	funmv Error	trigmv t_{ratio}	trigmv mv	trigmv Error	cosm t_{ratio}
orani678	100	1	719	2.1e-9	1.5	1224	4.1e-8	1.8e3
bcspwr10	10	1	265	3.0e-10	1.6	402	4.7e-10	3.7e3
gr_30_30	2	1	97	3.8e-9	1.2	136	5.5e-9	5.1e2
triw	10	1	13011	8.2e-13	1.1	26708	5.1e-9	4.4e-1
poisson	500	1	6415	1.3e-8	2.2	12436	2.5e-7	1.5e3

(c) Half precision

	t	funmv t_{ratio}	funmv mv	funmv Error	trigmv t_{ratio}	trigmv mv	trigmv Error	cosm t_{ratio}
orani678	100	1	551	1.6e-4	1.3	848	1.9e-3	2.3e3
bcspwr10	10	1	215	7.3e-6	1.5	324	1.3e-5	4.4e3
gr_30_30	2	1	93	3.1e-4	1.3	108	5.4e-6	4.9e2
triw	10	1	7381	1.1e-6	1.1	15028	5.4e-5	7.6e-1
poisson	500	1	5223	2.7e-4	2.1	9810	4.2e-4	1.9e3

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 [14, Prob. 4.1]; see also [16, Eq. (1.1)]. We use the Algorithm of Al-Mohy and Higham `expmv` 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. For a reference solution we use the MATLAB function `expm` to compute the left hand side of (6.1). We use the matrices and the vectors b prescribed in Experiment 2 except `poisson` due to memory limitation because of the use of `expm`. We take $z = [\sin 1, \sin 2, \cdots, \sin n]^T$ for all matrices. For fairer comparison we multiply by two the number of matrix–vector products mv counted by the code `expmv` because the dimension of the input matrices is $2n \times 2n$. Table 6.2 presents the results. Obviously our algorithm outperforms the alternative block version of the problem in terms of CPU time and computational cost with slightly better relative forward errors for single and half precisions. Using the MATLAB function `profile` to analyze the execution time for `funmv` and `expmv` in the experiment as a whole, the CPU time of `funmv` represents around 22 percent of the CPU time of both functions.

7. Concluding remarks. The algorithm we developed here has direct applications to solving second order systems of ODE's and their trigonometric numerical

Table 6.2
Experiment 3: t_{ratio} denotes time for method divided by time for `funmv`.

(a) Double precision

	t	funmv t_{ratio}	funmv mv	funmv Error	expmv t_{ratio}	expmv mv	expmv Error	expm t_{ratio}
orani678	100	1	920	3.2e-14	1.3	2046	3.2e-14	7.1e2
bcspwr10	10	1	190	4.5e-15	2.6	616	4.4e-15	1.6e4
gr_30_30	2	1	86	2.0e-15	1.5	180	1.7e-15	3.3e2
triw	10	1	1694	3.3e-14	3.8	4144	3.9e-14	3.3e1

(b) Single precision

	t	funmv t_{ratio}	funmv mv	funmv Error	expmv t_{ratio}	expmv mv	expmv Error	expm t_{ratio}
orani678	100	1	558	1.5e-9	1.5	1348	2.8e-8	1.3e3
bcspwr10	10	1	134	6.9e-11	3.0	496	2.3e-10	2.3e4
gr_30_30	2	1	58	3.2e-11	1.4	96	1.9e-10	4.6e2
triw	10	1	930	3.9e-11	3.6	2216	4.7e-9	5.8e1

(c) Half precision

	t	funmv t_{ratio}	funmv mv	funmv Error	expmv t_{ratio}	expmv mv	expmv Error	expm t_{ratio}
orani678	100	1	400	1.2e-4	1.6	992	1.8e-3	1.8e3
bcspwr10	10	1	80	2.4e-6	4.4	444	3.0e-5	3.6e4
gr_30_30	2	1	46	2.1e-7	1.1	60	1.8e-5	5.2e2
triw	10	1	650	6.9e-6	3.1	1370	2.5e-4	8.0e1

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) and (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 S_{pm} (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 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 control the computation is tol. Fourth, the algorithm spends most of its work 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. Algorithm 5.1 can be extended to be a multiprecision algorithm as in [6] since the function ρ_m (3.2) has an explicit expression that is easy to be handled by optimization software.

All these features make our algorithm attractive for black box use in a wide range of applications.

REFERENCES

[1] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.

[2] Awad H. Al-Mohy and Nicholas J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.*, 33(2):488–511, 2011.

[3] Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. New algorithms for computing the matrix sine and cosine separately or simultaneously. *SIAM J. Sci. Comput.*, 37(1): A456–A487, 2015.

[4] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Software*, 38(1):1–25, 2011.

[5] Ernesto Estrada, Desmond J. Higham, and Naomichi Hatano. Communicability and multipartite structures in complex networks at negative absolute temperatures. *Phys. Rev. E*, 78: 026102, 2008.

[6] Massimiliano Fasi and Nicholas J. Higham. Multiprecision algorithms for computing the matrix logarithm. MIMS EPrint 2017.16, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, May 2017. 19 pp.

[7] Ludwig Gauckler, Jianfeng Lu, Jeremy L. Marzuola, Frédéric Rousset, and Katharina Schratz. Trigonometric integrators for quasilinear wave equations. *arXiv preprint arXiv:1702.02981*, 2017.

[8] Volker Grimm and Marlis Hochbruck. Error analysis of exponential integrators for oscillatory second-order differential equations. *J. Phys. A: Math. Gen.*, 39(19):5495–5507, 2006.

[9] Volker Grimm and Marlis Hochbruck. Rational approximation to trigonometric operators. *BIT*, 48(2):215–229, 2008.

[10] Ernst Hairer and Christian Lubich. Long-time energy conservation of numerical methods for oscillatory differential equations. *SIAM J. Numer. Anal.*, 38(2):414–441, 2000.

[11] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31. Springer Science & Business Media, 2006. ISBN 978-3-540-30666-5.

[12] Gareth I. Hargreaves and Nicholas J. Higham. Efficient algorithms for the matrix cosine and sine. *Numer. Algorithms*, 40(4):383–400, 2005.

[13] Nicholas J. Higham. The Matrix Function Toolbox. http://www.maths.manchester.ac.uk/~higham/mftoolbox.

[14] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. xx+425 pp. ISBN 978-0-898716-46-7.

[15] Nicholas J. Higham and Awad H. Al-Mohy. Computing matrix functions. *Acta Numerica*, 19: 159–208, 2010.

[16] Nicholas J. Higham and Peter Kandolf. Computing the action of trigonometric and hyperbolic matrix functions. *SIAM J. Sci. Comput.*, 39(2):A613–A627, 2017.

[17] Nicholas J. Higham and Matthew I. Smith. Computing the matrix cosine. *Numer. Algorithms*, 34:13–26, 2003.

[18] Nicholas J. Higham and Françoise Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21(4):1185–1201, 2000.

[19] L. Jódar, E. Navarro, A. E. Posso, and M. C. Casabán. Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Appl. Numer. Math.*, 47(3-4):447–492, 2003.

[20] J. C. Mason and D. C. Handscomb. *Chebyshev Polynomials*. Chapman & Hall/CRC, Boca Raton, FL, 2003. xiv+341 pp. ISBN 0-8493-0355-9.

[21] Steven M. Serbin. Rational approximations of trigonometric matrices with application to second-order systems of differential equations. *Appl. Math. Comput.*, 5(1):75–92, 1979.

[22] Steven M. Serbin and Sybil A. Blalock. An algorithm for computing the matrix cosine. *SIAM J. Sci. Statist. Comput.*, 1(2):198–204, 1980.

[23] Xinyuan Wu, Kai Liu, and Wei Shi. *Structure-Preserving Algorithms for Oscillatory Differential Equations II*. Springer, Heidelberg; Science Press Beijing, Beijing, 2015. xv+298 pp. ISBN 978-3-662-48156-1.