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

Al-Mohy, Awad H. and Higham, Nicholas J. and Relton, Samuel D.

2014

MIMS EPrint: **2014.31**

ISSN 1749-9097

NEW ALGORITHMS FOR COMPUTING THE MATRIX SINE AND COSINE SEPARATELY OR SIMULTANEOUSLY[*]

AWAD H. AL-MOHY[†], NICHOLAS J. HIGHAM[‡], AND SAMUEL D. RELTON[‡]

Abstract. Several existing algorithms for computing the matrix cosine employ polynomial or rational approximations combined with scaling and use of a double angle formula. Their derivations are based on forward error bounds. We derive new algorithms for computing the matrix cosine, the matrix sine, and both simultaneously, that are backward stable in exact arithmetic and behave in a forward stable manner in floating point arithmetic. Our new algorithms employ both Padé approximants of $\sin x$ and new rational approximants to $\cos x$ and $\sin x$ obtained from Padé approximants to e^x. The amount of scaling and the degree of the approximants are chosen to minimize the computational cost subject to backward stability in exact arithmetic. Numerical experiments show that the new algorithms have backward and forward errors that rival or surpass those of existing algorithms and are particularly favorable for triangular matrices.

Key words. matrix sine, matrix cosine, matrix exponential, matrix function, backward error, forward error, rational approximation, Padé approximation, MATLAB, double angle formula, triple angle formula

AMS subject classifications. 65F30, 65F60

1. Introduction.

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

$$\sin A = A - \frac{A^3}{3!} + \frac{A^5}{5!} - \frac{A^7}{7!} + \cdots, \tag{1.1}$$

$$\cos A = I - \frac{A^2}{2!} + \frac{A^4}{4!} - \frac{A^6}{6!} + \cdots. \tag{1.2}$$

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

$$y''(t) + Ay(t) = g(t), \qquad y(0) = y_0, \quad y'(0) = y_0', \tag{1.3}$$

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

$$y(t) = \cos(\sqrt{A}t)y_0 + (\sqrt{A})^{-1}\sin(\sqrt{A}t)y_0' + \int_0^t (\sqrt{A})^{-1}\sin\big(\sqrt{A}(t-s)\big)g(s)\,ds, \tag{1.4}$$

where $\sqrt{A}$ denotes any square root of A [13, p. 124], [32]; see also [21, Prob. 4.1] for the case $g(t) = 0$. In practical computation a Krylov subspace method can be used to produce an approximation to (1.4) in which the same expression occurs with the potentially large matrix A replaced by a much smaller matrix [14].

[*] Version of January 9, 2015. This work was supported by European Research Council Advanced Grant MATFUN (267526) and Engineering and Physical Sciences Research Council grant EP/I03112X/1.

[†] Department of Mathematics, King Khalid University, PO Box 9004, Abha, Saudi Arabia (aal-mohy@hotmail.com, http://www.ma.man.ac.uk/~almohy).

[‡] School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk, http://www.maths.manchester.ac.uk/~higham samuel.relton@manchester.ac.uk, http://www.maths.manchester.ac.uk/~srelton).

More generally, (1.3) arises with a right-hand side of the form $g(t, y(t), y'(t))$ in a wide variety of applications and these problems can have highly oscillatory solutions [36] or be stiff with A having large norm [29]. Further generalizations of (1.3) whose solutions involve the matrix sine and cosine have a right-hand side $f(t, y(t)) + Bu(t)$, where $u(t)$ is a control vector and B a matrix [34], possibly with a delayed linear term $Ay(t - \tau)$ for a constant delay $\tau > 0$ [9].

Serbin and Blalock [33] proposed the following algorithm for the matrix cosine, which has served as a basis for several subsequent algorithms. It employs the double angle formula $\cos(2X) = 2\cos^2 X - I$ [21, Thm. 12.1].

Algorithm 1.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes an approximation Y to* $\cos A$.

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

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

Serbin and Blalock do not propose any specific algorithmic parameters. Higham and Smith [25] develop an algorithm based on the [8/8] Padé approximant with the scaling parameter s chosen with the aid of a forward error bound. Hargreaves and Higham [15] derive an algorithm with a variable choice of Padé degree (up to degree 20), again based on forward error bounds. They also give an algorithm that computes $\cos A$ and $\sin A$ simultaneously at a lower computational cost than computing the two functions separately. However they do not give an algorithm to compute $\sin A$ alone because the double angle formula $\sin(2X) = 2\sin X \cos X$ for the sine involves the cosine. Indeed, as far as we are aware, no algorithm of the general form in Algorithm 1.1 has been proposed for computing $\sin A$ directly.

Recently Sastre et al. [31] have derived a new algorithm for the cosine that combines Taylor series approximations of degree up to 40 with sharper forward error bounds derived using ideas similar to those in [1, sect. 4].

In this work we develop three new algorithms for the sine and cosine that are based on backward error analysis and are backward stable in exact arithmetic. Our backward error analysis has two advantages over the forward error analyses used in the derivation of existing algorithms. First, the backward error analysis applies to the overall algorithm, whereas the forward error analyses bound the forward error of the function of the scaled matrix only, and the best overall forward error bound contains a term exponential in the number of double-angle steps [21, Thm. 12.5]. Second, the current forward error-based algorithms actually bound the absolute error of the function of the scaled matrix rather than the relative error, which is a heuristic choice based on numerical experiments. With backward error analysis there is no need for such considerations, as relative backward errors are unequivocally appropriate.

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

The algorithm for the matrix cosine follows the outline of Algorithm 1.1 but uses Padé approximants of the exponential rather than the cosine. The algorithm for the matrix sine scales by powers of 3 rather than 2, employs Padé approximants to the sine and the exponential, and uses the triple angle formula to undo the effects of the scaling. The algorithm for computing the cosine and the sine simultaneously makes use of a new choice of double angle formula that improves stability.

The outline of the paper is as follows. In section 2 we perform backward error analysis of the Padé approximants for the sine and cosine and show how certain limitations can be overcome using alternative rational approximations based upon the matrix exponential. In section 3 we give explicit formulas for the cosine and sine of 2×2 (real quasi-)triangular matrices, which are used in the algorithms for better accuracy. In sections 4–6 we design cost effective methods for computing the matrix cosine and sine (separately and simultaneously) using the double and triple angle formulas. We then compare our algorithms numerically against existing alternatives in section 7 and give some concluding remarks in section 8.

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

2.1. Padé approximants of matrix sine. Let $r_m(x) = p_m(x)/q_m(x)$ denote the $[m/m]$ Padé approximant to the sine function. Since the sine function is odd the Padé table splits into 2×2 blocks containing identical entries having odd numerator p_m and even denominator q_m [5, p. 65]. From this we can show that for $k \geq 2$ the number of matrix multiplications required to form r_{2k} is equal to that for forming r_{2k+1} and hence we need only consider diagonal Padé approximants of odd order after r_1 and r_2. For a similar discussion on the Padé table of the cosine see [28] and [35, p. 245].

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

$$h_{2m+1}(x) := \arcsin r_m(x) - x, \tag{2.1}$$

where $\arcsin x$ denotes the principal arc sine, that is, the one for which $\arcsin x \in [-\pi/2, \pi/2]$ for x on the interval $[-1, 1]$. For $x \in \mathbb{C}$ with $|x| \leq 1$ we have $\sin \arcsin x = x$. It follows that for $X \in \mathbb{C}^{n\times n}$ with $\rho(r_m(X)) \leq 1$, where ρ denotes the spectral radius, $\arcsin r_m(X)$ is defined and, from (2.1),

$$r_m(X) = \sin(X + h_{2m+1}(X)) =: \sin(X + \mathit{\Delta} X). \tag{2.2}$$

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

Figure 2.1 shows the order stars for the Padé approximants r_m, that is, the regions of the complex plane where $|r_m(x)| \leq 1$, for a range of m [6], [27]. For a given value of m, if all the eigenvalues of X lie within this region then our backward error analysis holds. Since the regions are not circular it is difficult to check this criterion in practice, but we found numerically that for $m = 1\colon 13$ the largest circular disk centered at the origin that lies within all the regions has radius approximately $\operatorname{arcsinh} 1 \approx 0.881$. Therefore it is sufficient to check that $\rho(X) \leq \operatorname{arcsinh} 1$ for our analysis to hold.

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

$$h_{2m+1}(X) = X \sum_{k=0}^{\infty} c_{m,k} X^{2(m+k)}, \tag{2.3}$$

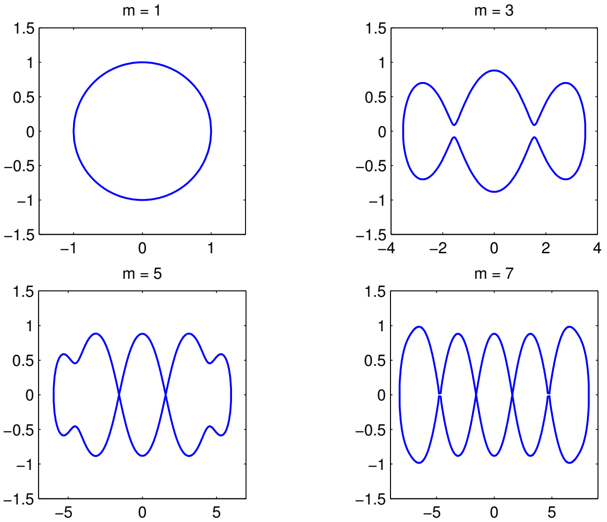

FIG. 2.1. *The boundary of the region within which $|r_m(x)| \le 1$, where $r_m(x)$ is the $[m/m]$ Padé approximant to the sine function, for $m = 1, 3, 5, 7$.*

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

$$\frac{\|\varDelta X\|}{\|X\|} \le \sum_{k=0}^{\infty} |c_{m,k}| \alpha_p(X)^{2(m+k)}, \tag{2.4}$$

where

$$\alpha_p(X) = \max\bigl(\|X^{2p}\|^{1/(2p)}, \|X^{2p+2}\|^{1/(2p+2)}\bigr) \tag{2.5}$$

and the integer $p \ge 1$ is such that $m \ge p(p-1)$. Note that we have $\rho(X) \le \alpha_p(X) \le \|X\|$ and $\alpha_p(X)$ can be much smaller than $\|X\|$ for nonnormal X, so the gains from working with (2.4) instead of the corresponding bound expressed solely in terms of $\|X\|$ can be significant. It is easy to show that $\alpha_3(X) \le \alpha_2(X) \le \alpha_1(X)$ and $\alpha_4(X) \le \alpha_2(X)$, but the relationship between $\alpha_3(X)$, $\alpha_4(X)$, and $\alpha_5(X)$ depends upon X, so we need to compute or estimate all of the latter three quantities and use the smallest, subject to the constraint $m \ge p(p-1)$.

To ensure maximum accuracy we would like to bound the backward error (2.4) by the unit roundoff $u = 2^{-53} \approx 1.1 \times 10^{-16}$ in IEEE double precision arithmetic. If we define

$$\beta_m = \max\left\{\beta : \sum_{k=0}^{\infty} |c_{m,k}| \beta^{2(m+k)} \le u\right\}, \tag{2.6}$$

then $\alpha_p(X) \le \beta_m \le \operatorname{arcsinh} 1$ implies that $\|\varDelta X\|/\|X\| \le u$ for $m \le 13$. In the second row of Table 5.1 we show the values β_m calculated using variable precision arithmetic in the Symbolic Math Toolbox for several values of m. We have $\beta_m > \operatorname{arcsinh} 1 \approx 0.881$ for $m \ge 9$, but our backward error bounds require that $\rho(X) \le \operatorname{arcsinh} 1$. This means that for $\beta_7 < \alpha_p(X) \le \operatorname{arcsinh} 1$ we can use $m = 9$, but for $\operatorname{arcsinh} 1 <$

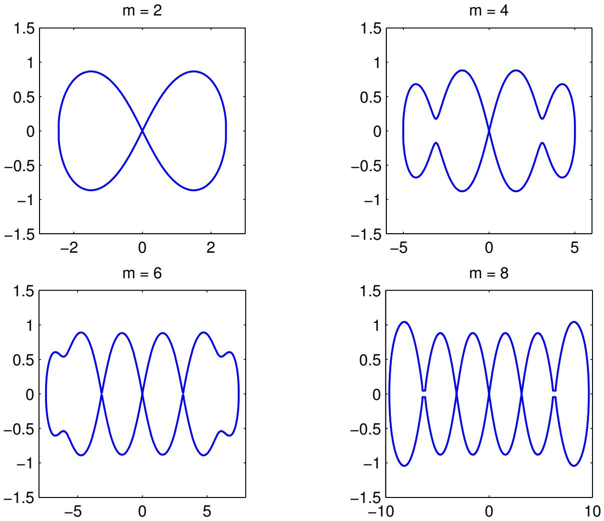

FIG. 2.2. *Boundary of the region within which $|r_m(x)| \le 1$, where $r_m(x)$ is the $[m/m]$ Padé approximant to the cosine function.*

$\alpha_p(X) \le \beta_9$ our backward error results are not applicable, so we artificially reduce β_9 to 0.881. From this point on we will not consider $m > 9$.

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

Figure 2.2 shows the order stars of r_m for a range of m. Requiring the eigenvalues of X to lie inside the order stars imposes tight restrictions on them. Even more prohibitively, the eigenvalues must also lie in the strip of the right half-plane with real part between 0 and π, so that the principal branch of arccos gives $\arccos \cos x = x$.

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

2.3. Exploiting Padé approximants of the exponential. Let the $[m/m]$ Padé approximant to e^x be denoted by $r_m(x) = p_m(x)/q_m(x)$. Al-Mohy and Higham [1, sect. 3] show that if $\rho(e^{-X} r_m(X) - I) < 1$ and $\rho(X) < \min\{\, |t| : q_m(t) = 0 \,\}$ then

$$r_m(X) = e^{X + h_{2m+1}(X)}, \tag{2.7}$$

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

$$r_m(-X) = e^{-(X + h_{2m+1}(X))}. \tag{2.8}$$

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

$$h_{2m+1}(iX) = i \sum_{k=m}^{\infty} (-1)^k c_{m,k} X^{2k+1} =: i\phi(X). \tag{2.9}$$

Now [21, eq. (12.2)]

$$\cos X = \frac{e^{iX} + e^{-iX}}{2}, \quad \sin X = \frac{e^{iX} - e^{-iX}}{2i}, \tag{2.10}$$

which suggests using as approximations the rational functions with real coefficients

$$c_m(x) = \frac{r_m(ix) + r_m(-ix)}{2}, \qquad s_m(x) = \frac{r_m(ix) - r_m(-ix)}{2i}. \tag{2.11}$$

From (2.7) and (2.9) we have

$$c_m(X) = \frac{r_m(iX) + r_m(-iX)}{2} = \frac{e^{i(X+\phi(X))} + e^{-i(X+\phi(X))}}{2} = \cos(X + \phi(X)), \tag{2.12}$$

$$s_m(X) = \frac{r_m(iX) - r_m(-iX)}{2i} = \frac{e^{i(X+\phi(X))} - e^{-i(X+\phi(X))}}{2i} = \sin(X + \phi(X)), \tag{2.13}$$

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

$$\frac{\|\phi(X)\|}{\|X\|} \le \frac{\widetilde{\phi}(\|X\|)}{\|X\|} = \frac{\widetilde{h}_{2m+1}(\|X\|)}{\|X\|}, \tag{2.14}$$

since $\widetilde{\phi}(x) = \widetilde{h}_{2m+1}(x)$. From [1, sect. 3] it follows that $\|\phi(X)\|/\|X\| \le u$ if $\alpha_p(X) \le \theta_m$, where the θ_m are given in Table 4.1 (reproduced from [20, Table 2.1]) and p is chosen to minimize $\alpha_p(X)$ in (2.5) subject to $m \ge p(p-1)$.

For the cosine we will use c_m but for the sine we will use a mixture of s_m and Padé approximants to $\sin x$. We now need to devise a strategy for choosing the parameters s (the amount of scaling) and m (the degree of the Padé approximant). We have $\theta_m \ge \beta_m$ for $m \le 21$, as can partially be seen from Tables 4.1 and 5.1, which means that less scaling is required if we approximate $\sin X$ by s_m than by the Padé approximant r_m. On the other hand, the numerator and denominator polynomials of c_m and s_m are of higher degree than those of the Padé approximants of the cosine and sine, so c_m and s_m will be more expensive to evaluate. In all cases considered here, s_m is a $[2m-1/2m]$ order rational approximant whereas c_m is of order $[2m/2m]$. For example

$$s_3(x) = \frac{x + \dfrac{7x^3}{60} - \dfrac{x^5}{600}}{1 + \dfrac{x^2}{20} + \dfrac{x^4}{600} + \dfrac{x^6}{14400}}, \qquad c_3(x) = \frac{1 - \dfrac{9x^2}{20} + \dfrac{11x^4}{600} - \dfrac{x^6}{14400}}{1 + \dfrac{x^2}{20} + \dfrac{x^4}{600} + \dfrac{x^6}{14400}}.$$

In sections 4–6 we explain how to balance the degree of the rational approximant with the amount of scaling in order to achieve the desired backward error at minimal cost.

2.4. Backward error propagation in multiple angle formulas. We have shown how to achieve a small backward error for the rational approximation of the sine or cosine of the scaled matrix. We now show that this backward error propagates linearly through the multiple angle formula phase of the algorithm (lines 3–5 of Algorithm 1.1), resulting in a small overall backward error. We state the result for the cosine; an analogous result holds for the sine.

Lemma 2.1. *Let $A \in \mathbb{C}^{n\times n}$ and $X = \eta^{-s}A$ for a positive integer η and nonnegative integer s, and suppose that $r(X) = \cos(X + \Delta X)$ for a rational function r. Then the approximation Y from the "scaling and multiple angle" method* *satisfies* $Y = \cos(A + \Delta A)$, *where* $\Delta A = \eta^s \Delta X$ *in exact arithmetic, and hence* $\|\Delta A\|/\|A\| = \|\Delta X\|/\|X\|$.

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

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

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

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

Lemma 2.1 shows that there is no growth in the relative backward error during the multiple angle phase in exact arithmetic. Hence by choosing the parameters s and m so that $\|\Delta X\|/\|X\| \le u$ we achieve an overall backward error bounded by u. This contrasts with the algorithms for the matrix cosine in [15], [21, Alg. 12.6], [31], which choose r to make $\|r(X) - \cos X\|$ small, since the overall error $\|r(A) - \cos A\|$ bears no simple relation to $\|r(X) - \cos X\|$.

3. Recomputing the cosine and sine of 2×2 matrices.

If we begin with an initial (real) Schur decomposition of our input matrix A, so that we are working with upper (quasi-)triangular matrices, then we can obtain higher overall accuracy by explicitly computing (instead of approximating) the diagonal blocks (including all of the first superdiagonal) explicitly throughout the algorithm. This idea has been used to good effect in recent algorithms for the matrix exponential [1], the logarithm [2], [3], and matrix powers [22], [23].

For 2×2 triangular matrices $T = \begin{bmatrix} \lambda_1 & t \\ 0 & \lambda_2 \end{bmatrix}$ it is known that [21, eq. (4.16)]

$$f(T) = \begin{bmatrix} f(\lambda_1) & tf[\lambda_1, \lambda_2] \\ 0 & f(\lambda_2) \end{bmatrix}, \tag{3.1}$$

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

$$f[\lambda_1, \lambda_2] = \begin{cases} (f(\lambda_1) - f(\lambda_2))/(\lambda_1 - \lambda_2), & \text{if } \lambda_1 \ne \lambda_2, \\ f'(\lambda_1), & \text{if } \lambda_1 = \lambda_2. \end{cases} \tag{3.2}$$

For the cosine and sine we can avoid subtractive cancellation in the numerator of the divided difference when $\lambda_1 \approx \lambda_2$ by computing

$$\cos[\lambda_1, \lambda_2] = -\left[\sin\left(\frac{\lambda_1 + \lambda_2}{2}\right) \sin\left(\frac{\lambda_1 - \lambda_2}{2}\right)\right] \Big/ \left(\frac{\lambda_1 - \lambda_2}{2}\right), \tag{3.3}$$

$$\sin[\lambda_1, \lambda_2] = \left[\cos\left(\frac{\lambda_1 + \lambda_2}{2}\right)\right] \sin\left(\frac{\lambda_1 - \lambda_2}{2}\right) \Big/ \left(\frac{\lambda_1 - \lambda_2}{2}\right). \tag{3.4}$$

These formulae are evaluated to high relative accuracy in floating point arithmetic when $\lambda_1 \approx \lambda_2$ because the function $\sin x/x$ is well conditioned for small $|x|$; moreover, for $\lambda_1 \approx \lambda_2$ the difference $\lambda_1 - \lambda_2$ is evaluated exactly [19, sec. 2.5].

A 2×2 block of a real upper quasi-triangular matrix computed by the LAPACK Schur decomposition code `dgees` [4] has the form

$$B = \begin{bmatrix} a & b \\ c & a \end{bmatrix}, \quad bc < 0. \tag{3.5}$$

We can use the polynomial definition of a matrix function [21, Def. 1.4] to show that

$$\cos B = \begin{bmatrix} \cos a \cosh \theta & -\theta^{-1} b \sin a \sinh \theta \\ -\theta^{-1} c \sin a \sinh \theta & \cos a \cosh \theta \end{bmatrix}, \tag{3.6}$$

$$\sin B = \begin{bmatrix} \sin a \cosh \theta & b \cos a \sinh \theta \\ \theta^{-1} c \cos a \sinh \theta & \sin a \cosh \theta \end{bmatrix}, \tag{3.7}$$

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

4. Algorithm for the matrix cosine. We now build an algorithm for the matrix cosine based upon the rational approximation c_m of (2.11) along with the double angle formula.

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

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

First we discuss which values of m need to be considered for the approximant $c_m(X)$. We will consider $m = 1\colon m_{\max}$, where $m_{\max}$ is defined as the largest m for which the denominator of $c_m(X)$ has condition number, in the appropriate norm, smaller than 10 for any X in the region where $\alpha_p(X) \le \theta_m$. The appropriate norm is as follows: for any $\epsilon > 0$ there exists a norm $\|\cdot\|_\epsilon$ satisfying $\|X\|_\epsilon \le \rho(X) + \epsilon \le \alpha_p(X) + \epsilon$, where we will choose $\epsilon = u$ to be the unit roundoff. This norm is necessary since we intend to choose the value of m using $\alpha_p(X)$, which is potentially much smaller than $\|X\|$, and therefore we have no a priori knowledge of how large $\|X\|$ will be. In this norm the corresponding condition number of the denominator $q_m(X)$ of $c_m(X)$ can be bounded above (see [1, p. 982]) as

$$\|q_m(X)\|_\epsilon \|q_m(X)^{-1}\|_\epsilon \le \widetilde{q}_m(\alpha_p(X) + \epsilon) \sum_{k=0}^{\infty} |a_k| (\alpha_p(X) + \epsilon)^k \tag{4.1}$$

$$\le \widetilde{q}_m(\theta_m + \epsilon) \sum_{k=0}^{\infty} |a_k| (\theta_m + \epsilon)^k =: \kappa_m, \tag{4.2}$$

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

Using 250 digit arithmetic and calculating the first 350 terms of the Taylor series in the bound (4.2), we found that for the matrix cosine this means considering $m_{\max} = 21$, where θ_{21} is reduced from 13.95, which has corresponding condition number 10.75, to 13 with condition number 7.86 for additional stability. Note that this part of our

Table 4.1

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

m	1	2	3	4	6	8	10	12	15	18	21
$\pi(c_m(X))$	1	2	3	4	5	6	7	8	9	10	11
θ_m	3.6e-8	5.3e-4	1.5e-2	8.5e-2	5.4e-1	1.47	2.8	4.46	7.34	10.54	13.95 (13)
p	1	2	2	2	3	3	3	3, 4	3, 4	3, 4	3, 4, 5
κ_m	1	1	1	1	1.01	1.07	1.2	1.54	2.53	4.89	7.86

analysis also applies to the matrix sine since c_m and s_m share the same denominator polynomial.

This condition ensures that solving the multiple right-hand side system to form the rational approximant does not introduce significant errors into the computation—at least as long as $\|\cdot\|_\epsilon$ is not too badly scaled. The upper bounds κ_m on the condition number for each m of interest are shown in Table 4.1.

Now that we have a range of m to consider, we must find those values for which c_m can be formed in the smallest number of matrix multiplications. There are many ways to evaluate the rational approximants: in the appendix we show that applying the Paterson–Stockmeyer scheme [21, pp. 73–74], [30] is more efficient than explicit computation of the necessary powers. Evaluating the numerator and denominator polynomials using the Paterson–Stockmeyer scheme as described in the appendix we see that, for example, $c_7(X)$ and $c_8(X)$ can both be formed using a minimum of 6 matrix multiplications. Since $c_8(X)$ can be applied to X with a larger value of $\alpha_p(X)$ it renders $c_7(X)$ redundant.

Table 4.1 contains the relevant values of m and θ_m along with the number of matrix multiplications required to form $c_m(X)$. It also shows the values of p that we need to consider in order to minimize $\alpha_p(X)$ in (2.5) for each m.

Finally, we consider the choice of the parameters s (where $X = 2^{-s}A$) and m. It may sometimes be cheaper to perform a stronger scaling on A and use a lower order approximant. In particular, each invocation of the double angle formula costs one matrix multiplication and therefore it is worth increasing s by q if more than q multiplications can be saved when evaluating the rational approximant. (Note that this logic applies equally well to full matrices and triangular matrices.) From Table 4.1 we see that there are a number of places where such an increase in s is desirable. For example when $\theta_{10} < \alpha_3(X) \le 2\theta_8$ we can perform one extra scaling and use $c_8(X)$ instead of $c_{12}(X)$, saving one multiplication.

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

X^ℓ for any integer $\ell > 0$, without calculating the matrix power explicitly, using the block 1-norm estimator of Higham and Tisseur [26]. A further optimization is that after eliminating $m = 1$ as a possibility, for example, all higher order approximants $c_m(X)$ require X^4 so we can compute and store X^4 and hence use $\|X^4\|_1^{1/4}$ rather than an estimate of it in the logical tests. A similar optimization can be performed after eliminating $m = 2$ and $m = 6$. Taking all these issues into account leads to the following algorithm to determine the parameters s and m. The many logical tests are the price we pay for making the most efficient choice of parameters.

Algorithm 4.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the parameters s and m such that the algorithm for approximating $\cos A$ based on $c_m(2^{-s}A)$ and the double angle formula produces (in exact arithmetic) a backward error bounded by u. The algorithm uses the parameters θ_m given in Table 4.1.*

```
$s = 0$
Compute and store $A^2$.
$\alpha_1(A) = \|A^2\|_1^{1/2}$
if $\alpha_1(A) \le \theta_1$, $m = 1$, quit, end
Compute and store $A^4$.
$d_4 = \|A^4\|_1^{1/4}$
Estimate $d_6 = \|A^6\|_1^{1/6}$.
$\alpha_2(A) = \max(d_4, d_6)$
if $\alpha_2(A) \le \theta_2$, $m = 2$, quit, end
Compute and store $A^6$.
$d_6 = \|A^6\|_1^{1/6}$   % Compute exact value and update $\alpha_2$.
$\alpha_2(A) = \max(d_4, d_6)$
if $\alpha_2(A) \le \theta_3$, $m = 3$, quit, end
if $\alpha_2(A) \le \theta_4$, $m = 4$, quit, end
Estimate $d_8 = \|A^8\|_1^{1/8}$.
$\alpha_3(A) = \max(d_6, d_8)$
if $\alpha_3(A) \le \theta_6$, $m = 6$, quit, end
Compute and store $A^8$.
$d_8 = \|A^8\|_1^{1/8}$   % Compute exact value and update $\alpha_3$.
$\alpha_3(A) = \max(d_6, d_8)$
if $\alpha_3(A) \le \theta_8$, $m = 8$, quit, end
if $\alpha_3(A) \le \theta_{10}$, $m = 10$, quit, end
if $\alpha_3(A) \le 2\theta_8$, $s = 1$, $m = 8$, quit, end
Estimate $d_{10} = \|A^{10}\|_1^{1/10}$.
$\alpha_4(A) = \max(d_8, d_{10})$
$\alpha_{3/4} = \min(\alpha_3(A), \alpha_4(A))$
if $\alpha_{3/4}(A) \le \theta_{12}$, $m = 12$, quit, end
if $\alpha_3(A) \le 2\theta_{10}$, $s = 1$, $m = 10$, quit, end
if $\alpha_3(A) \le 4\theta_8$, $s = 2$, $m = 8$, quit, end
% Note that $s = 0$ at this point
if $\alpha_{3/4}(A) \le \theta_{15}$, $m = 15$, quit, end
if $\alpha_{3/4}(A) \le 2\theta_{12}$, $s = s + 1$, $m = 12$, quit, end
if $\alpha_3(A) \le 4\theta_{10}$, $s = s + 2$, $m = 10$, quit, end
if $\alpha_3(A) \le 8\theta_8$, $s = s + 3$, $m = 8$, quit, end
if $\alpha_{3/4}(A) \le \theta_{18}$, $m = 18$, quit, end
if $\alpha_{3/4}(A) \le 2\theta_{15}$, $s = s + 1$, $m = 15$, quit, end
```

```
37  if $\alpha_{3/4}(A) \le 4\theta_{12}$, $s = s + 2$, $m = 12$, quit, end
38  if $\alpha_3(A) \le 8\theta_{10}$, $s = s + 3$, $m = 10$, quit, end
39  Estimate $d_{12} = \|A^{12}\|_1^{1/12}$.
40  $\alpha_5(A) = \max(d_{10}, d_{12})$
41  $\alpha_{3/4/5}(A) = \min(\alpha_3, \alpha_4, \alpha_5)$
42  if $\alpha_{3/4/5}(A) \le \theta_{21}$, $m = 21$, quit, end
43  % $A$ needs to be scaled. After scaling, $6.5 \approx \theta_{21}/2 < \alpha_{3/4/5}(A) \le \theta_{21}$.
44  $s = \lceil \log_2(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$
45  $\alpha_3(A) = \alpha_3(A)/2^s$
46  $\alpha_{3/4}(A) = \alpha_{3/4}(A)/2^s$
47  $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/2^s$
48  Execute lines 31–38.
49  $m = 21$
```

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

ALGORITHM 4.2. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm computes $C = \cos A$. The algorithm is designed for use with IEEE double precision arithmetic.*

```
1   Compute the (real if $A \in \mathbb{R}^{n\times n}$) Schur decomposition $A = QTQ^*$.
2   Obtain $s$ and $m$ from Algorithm 4.1 applied to $T$.
3   $T \leftarrow 2^{-s}T$ and $T^k \leftarrow 2^{-sk}T^k$ for any powers $T^k$ stored during line 2.
4   Compute $C = c_m(T)$: use the Paterson–Stockmeyer scheme to compute the
    numerator $p_m(X)$ and denominator $q_m(X)$ and then solve $q_m(T)C = p_m(T)$.
5   Recompute the diagonal blocks of $C$ using (3.1), (3.3), (3.6)
6   for $j = 1\colon s$
7       $C \leftarrow 2C^2 - I$
8       Recompute the diagonal blocks of $C = \cos(2^jT)$ using (3.1) (3.3), (3.6).
9   end
10  $C \leftarrow QCQ^*$
```

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

5. Algorithm for the matrix sine.

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

Table 5.1 shows the number of matrix multiplications required to form the Padé approximants $r_m(X)$, denoted $\pi(r_m(X))$. The values of $\pi(s_m(X))$ are $\pi(s_1(X)) = 1$ and for $m \ge 2$ we have $\pi(s_m(X)) = \pi(c_m(X)) + 1$, where $\pi(c_m(X))$ is given in Table 4.1. The table also contains the values of p that we need to consider and the values β_m such that $\alpha_p(X) \le \beta_m$ implies the backward error of the result is bounded by u in exact arithmetic. The last row of the table shows an upper bound κ_m on the condition number of the denominator polynomial of $r_m(X)$ for $\alpha_p(X) \le \beta_m$, for the

Table 5.1

The number of matrix multiplications $\pi(\cdot)$ required to evaluate $r_m(x)$, values of β_m in (2.6), values of p to be considered, and an upper bound κ_m for the condition number of the denominator polynomial of r_m for $\alpha_p(X) \le \beta_m$. Values of m for which $\pi(r_m(X)) = \pi(r_{m+1}(X))$ are not shown as they provide no benefit. The values for β_9 has been artificially reduced from 1.14 *to* 8.81e-1 *as required by our backward error analysis in section 2.1; this changes the value of κ_9.*

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

$\|\cdot\|_\epsilon$ norm defined in the previous section. For $s_m(X)$ the values for θ_m, p, and κ_m exactly match those already given for the cosine in Table 4.1: the values of θ_m match due to the relationship to the matrix exponential (see section 2.3), the p match due to the degree of the approximants, and the κ_m match as c_m and s_m have identical denominator polynomials.

For the Padé approximants r_m, the maximum order of approximation considered is $m = 9$ in order to ensure the validity of the bounds, as explained in section 2.1. On the other hand, as for the matrix cosine, we consider up to $m = 21$ for the alternative rational approximants $s_m(X)$. The use of $s_m(X)$ allows larger p within the values $\alpha_p(X)$ used in our backward error bounds and can therefore reduce the amount of scaling required. For example, suppose A is nilpotent of degree 10 with $\alpha_3(A) \gg \beta_9$. Using only Padé approximants we would require a large amount of scaling but since A is nilpotent we know that $\alpha_5(A) = 0$, allowing the use of $s_{21}(A)$ with no scaling required.

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

This discussion leads to the following parameter selection algorithm.

Algorithm 5.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the scaling parameter s and rational approximant φ, equal to r_m or s_m, such that the algorithm for approximating* $\sin A$ *based on $\varphi(3^{-s}A)$ and the triple angle formula produces (in exact arithmetic) a backward error bounded by u. The algorithm uses the parameters θ_m and β_m given in Tables 4.1 and 5.1, respectively.*

```
$s = 0$
Estimate $d_2 = \|A^2\|_1^{1/2}$.
$\alpha_1(A) = d_2$
if $\alpha_1(A) \le \beta_1$, $\varphi(x) = r_1(x)$, quit, end
Compute and store $A^2$.
$d_2 = \|A^2\|_1^{1/2}$   % Compute exact value and update $\alpha_1$.
$\alpha_1(A) = d_2$
if $\alpha_1(A) \le \theta_1$, $\varphi(x) = s_1(x)$, quit, end
Estimate $d_4 = \|A^4\|_1^{1/4}$ and $d_6 = \|A^6\|_1^{1/6}$.
```

```
10  $\alpha_2(A) = \max(d_4, d_6)$
11  if $\alpha_2(A) \le \beta_3$, $\varphi(x) = r_3(x)$, quit, end
12  Compute and store $A^4$.
13  $d_4 = \|A^4\|_1^{1/4}$   % Compute exact value and update $\alpha_2$.
14  $\alpha_2(A) = \max(d_4, d_6)$
15  if $\alpha_2(A) \le \beta_5$, $\varphi(x) = r_5(x)$, quit, end
16  Compute and store $A^6$.
17  $d_6 = \|A^6\|_1^{1/6}$
18  Estimate $d_8 = \|A^8\|_1^{1/8}$.
19  $\alpha_3(A) = \max(d_6, d_8)$
20  if $\alpha_3(A) \le \beta_7$, $\varphi(x) = r_7(x)$, quit, end
21  if $\alpha_3(A) \le \beta_9$, $\varphi(x) = r_9(x)$, quit, end
22  if $\alpha_3(A) \le 3\beta_7$, $s = 1$, $\varphi(x) = r_7(x)$, quit, end
23  if $\alpha_3(A) \le 3\beta_9$, $s = 1$, $\varphi(x) = r_9(x)$, quit, end
24  if $\alpha_3(A) \le \theta_{10}$, $\varphi(x) = s_{10}(x)$, quit, end
25  if $\alpha_3(A) \le 9\beta_7$, $s = 2$, $\varphi(x) = r_7(x)$, quit, end
26  Compute and store $A^8$.
27  $d_8 = \|A^8\|_1^{1/8}$   % Compute exact value and update $\alpha_3$.
28  $\alpha_3(A) = \max(d_6, d_8)$
29  Compute and store $A^{10}$.
30  $d_{10} = \|A^{10}\|_1^{1/10}$
31  $\alpha_4(A) = \max(d_8, d_{10})$
32  $\alpha_{3/4} = \min(\alpha_3(A), \alpha_4(A))$
33  % Note that $s = 0$ at this point.
34  if $\alpha_{3/4}(A) \le \theta_{12}$, $\varphi(x) = s_{12}(x)$, quit, end
35  if $\alpha_3(A) \le 9\beta_9$, $s = s + 2$, $\varphi(x) = r_9(x)$, quit, end
36  if $\alpha_{3/4}(A) \le \theta_{15}$, $\varphi(x) = s_{15}(x)$, quit, end
37  if $\alpha_3(A) \le 3\theta_{10}$, $s = s + 1$, $\varphi(x) = s_{10}(x)$, quit, end
38  if $\alpha_{3/4}(A) \le \theta_{18}$, $\varphi(x) = s_{18}(x)$, quit, end
39  if $\alpha_{3/4}(A) \le 3\theta_{12}$, $s = s + 1$, $\varphi(x) = s_{12}(x)$, quit, end
40  Estimate $d_{12} = \|A^{12}\|_1^{1/12}$.
41  $\alpha_5(A) = \max(d_{10}, d_{12})$
42  $\alpha_{3/4/5} = \min(\alpha_3, \alpha_4, \alpha_5)$
43  if $\alpha_{3/4/5}(A) \le \theta_{21}$, $\varphi(x) = s_{21}(x)$, quit, end
44  % $A$ needs to be scaled. After scaling, $4.3 \approx \theta_{21}/3 < \alpha_{3/4/5}(A) \le \theta_{21}$.
45  $s = \lceil \log_3(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$
46  $\alpha_3(A) = \alpha_3(A)/3^s$
47  $\alpha_{3/4}(A) = \alpha_{3/4}(A)/3^s$
48  $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/3^s$
49  if $\alpha_3(A) \le 9\beta_7$, $s = s + 2$, $\varphi(x) = r_7(x)$, quit, end
50  Execute lines 34–39.
51  $\varphi(x) = s_{21}(x)$
```

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

Algorithm 5.2. *Given* $A \in \mathbb{C}^{n\times n}$ *this algorithm computes* $S = \sin A$. *The algorithm is designed for use with IEEE double precision arithmetic.*

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

Table 6.1
Number of matrix multiplications π_m required to evaluate both c_m and s_m.

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

```
2  Obtain $s$ and $\varphi(x)$ from Algorithm 5.1 applied to $T$.
3  $T \leftarrow 3^{-s}T$ and $T^k \leftarrow 3^{-sk}T^k$ for any powers of $T$ stored during line 2.
4  Compute $S = \varphi(T)$: use the Paterson–Stockmeyer scheme to compute the
   numerator $p(X)$ and denominator $q(X)$ and then solve $q(T)S = p(T)$.
5  Recompute the diagonal blocks of $S$ using (3.1), (3.4), (3.7)
6  for $j = 1\colon s$
7      $S \leftarrow S(3I - 4S^2)$
8      Recompute the diagonal blocks of $S = \sin(3^j T)$ using (3.1), (3.4), (3.7).
9  end
10 $S \leftarrow QSQ^*$
```

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

6. Algorithm for simultaneous computation of the matrix cosine and sine.

We now design an algorithm to compute the matrix cosine and sine simultaneously, a requirement that arises in the evaluation of (1.4), for example. We use the rational approximants $c_m(x)$ and $s_m(x)$ introduced in section 2.3, since they have the same denominator polynomial: this saves a significant amount of computation since we need only compute one denominator for both approximants and furthermore we can reuse an LU factorization of the denominator when computing $c_m(X)$ and $s_m(X)$.

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

$$\cos(2X) = 2\cos^2 X - I, \qquad \cos(2X) = I - 2\sin^2 X. \tag{6.1}$$

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

In Table 6.1 we show the number of matrix multiplications π_m required to form both $c_m(x)$ and $s_m(x)$ using the Paterson–Stockmeyer scheme (Appendix A). Since each invocation of the double angle formulas for the cosine and sine costs two matrix multiplications in total, it is worth performing q extra scalings if more than $2q$ matrix multiplications can be saved in the formation of the rational approximant. This results in the following parameter selection algorithm.

Algorithm 6.1. *Given $A \in \mathbb{C}^{n\times n}$ this algorithm determines the parameters s and m such that the algorithm for approximating $\cos A$ and $\sin A$ based on $c_m(2^{-s}A)$ and $s_m(2^{-s}A)$ and the double angle formulas produces (in exact arithmetic) backward errors bounded by u for both functions. The algorithm uses the parameters θ_m given in Table 4.1.*

```
1  $s = 0$
2  Compute and store $A^2$.
3  $d_2 = \|A^2\|_1^{1/2}$
```

```
4   $\alpha_1(A) = d_2$
5   if $\alpha_1(A) \le \theta_1$, $m = 1$, quit, end
6   Compute and store $A^4$.
7   $d_4 = \|A^4\|_1^{1/4}$
8   Estimate $d_6 = \|A^6\|_1^{1/6}$.
9   $\alpha_2(A) = \max(d_4, d_6)$
10  if $\alpha_2(A) \le \theta_2$, $m = 2$, quit, end
11  Compute and store $A^6$.
12  $d_6 = \|A^6\|_1^{1/6}$   % Compute exact value and update $\alpha_2$.
13  $\alpha_2(A) = \max(d_4, d_6)$
14  if $\alpha_2(A) \le \theta_3$, $m = 3$, quit, end
15  if $\alpha_2(A) \le \theta_4$, $m = 4$, quit, end
16  if $\alpha_2(A) \le \theta_5$, $m = 5$, quit, end
17  Estimate $d_8 = \|A^8\|_1^{1/8}$.
18  $\alpha_3(A) = \max(d_6, d_8)$
19  if $\alpha_3(A) \le \theta_6$, $m = 6$, quit, end
20  Compute and store $A^8$.
21  $d_8 = \|A^8\|_1^{1/8}$   % Compute exact value and update $\alpha_3$.
22  $\alpha_3(A) = \max(d_6, d_8)$
23  if $\alpha_3(A) \le \theta_8$, $m = 8$, quit, end
24  Compute and store $A^{10}$.
25  if $\alpha_3(A) \le \theta_{10}$, $m = 10$, quit, end
26  Compute and store $A^{12}$.
27  $d_{10} = \|A^{10}\|_1^{1/10}$
28  $\alpha_4(A) = \max(d_8, d_{10})$
29  $\alpha_{3/4}(A) = \min(\alpha_3, \alpha_4)$
30  if $\alpha_{3/4}(A) \le \theta_{12}$, $m = 12$, quit, end
31  if $\alpha_{3/4}(A) \le \theta_{14}$, $m = 14$, quit, end
32  % Note that $s = 0$ at this point.
33  if $\alpha_{3/4}(A) \le \theta_{16}$, $m = 16$, quit, end
34  if $\alpha_{3/4}(A) \le 2\theta_{12}$, $s = s + 1$, $m = 12$, quit, end
35  if $\alpha_{3/4}(A) \le \theta_{18}$, $m = 18$, quit, end
36  if $\alpha_{3/4}(A) \le 2\theta_{14}$, $s = s + 1$, $m = 14$, quit, end
37  $d_{12} = \|A^{12}\|_1^{1/12}$
38  $\alpha_5(A) = \max(d_{10}, d_{12})$
39  $\alpha_{3/4/5}(A) = \min(\alpha_3, \alpha_4, \alpha_5)$
40  if $\alpha_{3/4/5}(A) \le \theta_{21}$, $m = 21$, quit, end
41  % $A$ needs to be scaled. After scaling, $6.5 \approx \theta_{21}/2 \le \alpha_{3/4/5}(A) \le \theta_{21}$.
42  $s = \lceil \log_2(\alpha_{3/4/5}(A)/\theta_{21}) \rceil$
43  $\alpha_{3/4}(A) = \alpha_{3/4}(A)/2^s$
44  $\alpha_{3/4/5}(A) = \alpha_{3/4/5}(A)/2^s$
45  Execute lines 33–36.
46  $m = 21$
```

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

Algorithm 6.2. *Given $A \in \mathbb{C}^{n\times n}$ the following algorithm computes both $C = \cos A$ and $S = \sin A$. The common denominator of $c_m(x)$ and $s_m(x)$ is denoted*

by $\omega(x)$ so that $c_m(x) = \widehat{c}_m(x)/\omega(x)$ and $s_m(x) = \widehat{s}_m(x)/\omega(x)$. This algorithm is designed for use with IEEE double precision arithmetic.

```
1  Compute the (real if $A \in \mathbb{R}^{n\times n}$) Schur decomposition $A = QTQ^*$.
2  Obtain $s$ and $m$ from Algorithm 6.1 applied to $T$.
3  $T \leftarrow 2^{-s}T$ and $T^k \leftarrow 2^{-sk}T^k$ for any powers of $T$ stored during line 2.
4  Compute the shared denominator $\omega_m(X)$ and the numerators $\widehat{c}_m(X)$ and
   $\widehat{s}_m(X)$ using the Paterson–Stockmeyer method (Appendix A).
5  Compute an $LU$ factorization with partial pivoting $LU = \omega_m(X)$.
6  Compute $S = U^{-1}L^{-1}\widehat{s}_m(X)$ and $C = U^{-1}L^{-1}\widehat{c}_m(X)$ by substitution
   using the $LU$ factorization.
7  Recompute the block diagonals of $S$ and $C$ using (3.1)–(3.7).
8  for $j = 1\colon s$
9      $S_{\mathrm{old}} = S$
10     $S = 2SC$
11     Recompute the block diagonals of $S = \sin(2^jT)$ using (3.1), (3.4), (3.7).
12     $C = I - 2S_{\mathrm{old}}^2$
13     Recompute the block diagonals of $C = \cos(2^jT)$ using (3.1), (3.3), (3.6).
14 end
15 $S \leftarrow QSQ^*$
16 $C \leftarrow QCQ^*$
```

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

For comparison, the cost of calling Algorithms 4.2 and 5.2 separately—but using only one Schur decomposition—is $(31 + (\pi_c + \pi_s + s_c + s_s + 2)/3)n^3$ flops, or $(16/3 + 2(\pi_c + \pi_s + s_c + s_s))n^3$ flops if the transformation-free version is used, where the subscripts c and s denote the values obtained from the cosine and sine algorithms, respectively. To illustrate the benefit to the overall cost gained by computing the matrix cosine and sine simultaneously consider the matrix `A = gallery('frank',1000)`: we obtain the values $\pi_c = 9$, $\pi_s = 5$, $s_c = 14$, $s_s = 11$ when using Algorithms 4.2 and 5.2, and $\pi = 14$, $s = 14$ when using Algorithm 6.2. Therefore the cost of computing the matrix cosine and sine separately is $(220/3)n^3$ and $(316/3)n^3$ flops (with and without a Schur decomposition, respectively) as opposed to $(137/3)n^3$ and $(266/3)n^3$ flops when computing both functions simultaneously. For the Schur decomposition algorithm this is a computational saving of around 38 percent.

7. Numerical experiments.

All our experiments are performed in MATLAB 2013b. The test matrices are mainly 15×15 matrices adapted from the Matrix Computation Toolbox [17], the MATLAB `gallery` function, and the matrix function literature. We select matrices for which none of the algorithms overflow and multiply them by a uniform random scalar between 1 and 60, to ensure that some scaling will occur in the algorithms. Each matrix is then used once in this form and a second time multiplied by the imaginary unit i, in order to test the algorithms for complex matrices. In total 76 matrices are used. Similar test matrices were used in recent work on the matrix cosine [15], [21, chap.12], [31].

Our numerical experiments compare the normwise forward and backward errors of the competing methods. If Y denotes the computed value of $f(A)$ then the forward error is

$$\frac{\|Y - f(A)\|_1}{\|f(A)\|_1} \tag{7.1}$$

and the backward error is

$$\psi(Y) = \min\{\, \|E\|_1/\|A\|_1 : Y = f(A+E)\,\}. \tag{7.2}$$

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

$$Y = f(A+E) \approx f(A) + L_f(A,E), \tag{7.3}$$

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

$$K_f(A)\,\mathrm{vec}(E) \approx \mathrm{vec}(Y - f(A)), \tag{7.4}$$

where $K_f(A) \in \mathbb{C}^{n^2 \times n^2}$ is the Kronecker matrix and $\mathrm{vec}(E)$ stacks the columns of E vertically from first to last [21, chap. 3], [24]. We approximate the backward error $\psi(Y)$ by the minimum 2-norm solution of (7.4), where the Kronecker matrix (obtained using [21, Alg. 3.17]) and $f(A)$ are computed, and the system solved, in 250 digit arithmetic. A similar idea is used effectively by Deadman and Higham in [8, sect. 5] to compute linearized backward errors of functional identities.

Since we showed in Lemma 2.1 that all our algorithms are backward stable in exact arithmetic we expect the relative error (7.1) to be bounded by a modest multiple of $\mathrm{cond}(f,A)u$, where the condition number $\mathrm{cond}(f,A)$ is defined in [21, chap. 3].

Accurate values of $\cos A$ and $\sin A$ for use in (7.1) and (7.4) are generated in 250 digit arithmetic using the Symbolic Toolbox by adding a random perturbation of norm 10^{-125} to A then diagonalizing it and taking the cosine (or sine) of the eigenvalues; the perturbation ensures the eigenvalues are distinct so that the diagonalization is always possible. This idea is based upon [7] and has been used successfully in [3], [23], and [24]. The condition number of f is estimated using the code `funm_condest1` from the Matrix Function Toolbox [18], [21, Alg. 3.20].

We test our new algorithms against the existing alternatives:

- `cosm` for $\cos A$ from [18], which implements an algorithm of Hargreaves and Higham [15, Alg. 3.1], [21, Alg. 12.6].
- `cosmsinm` for $\cos A$ and $\sin A$ from [18], which implements an algorithm of Hargreaves and Higham [15, Alg. 5.1], [21, Alg. 12.8].
- `costay`[1] for $\cos A$ from Sastre et al. [31].

The first two algorithms use variable degree Padé approximants, while `costay` uses a variable degree truncated Taylor series. None of these algorithms performs a Schur decomposition of the input matrix or recomputes the diagonal blocks (as described in section 3), and all are based on forward error analysis. We will perform comparisons both with and without the initial Schur decomposition within our algorithms.

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

- `cosm_new`: Algorithm 4.2,

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

- `sinm_new`: Algorithm 5.2,
- `cosmsinm_new`: Algorithm 6.2.

In each case we plot our results in a 4×2 grid. Each plot in the $(1, 1)$ position shows the forward errors (7.1), with the new algorithm in transformation-free form, along with an estimate of $\text{cond}(f, A)u$. The $(1, 2)$ plot presents the same relative errors as a performance profile [11], [16, sect. 22.4], to which we apply the strategy from [10] to avoid tiny relative errors skewing the results. Briefly, for a given curve on the performance profile and a given α the corresponding p value is the proportion of problems for which the error for that method was within a factor α of the smallest error over all methods. The $(2, 1)$ plot shows the backward errors (7.2) and the $(2, 2)$ plot is a performance profile for the backward error results. The remaining four plots show the same results when we allow our new algorithms to compute an initial Schur decomposition, allowing recomputation of the diagonal blocks (section 3). The coloured triangles in each plot show any cases where the algorithm returned an error greater than one.

Our next experiment tests each algorithm on matrices that are already triangular: we precompute the Schur decomposition of each test matrix before sending the triangular factor to the algorithms. This shows how well the algorithms exploit triangularity. In some applications the original matrices are triangular; see, for example, [37] in the case of the matrix exponential.

The final experiment compares the accuracy of `cosm_new` and `costay` on a matrix resulting from the semidiscretization of a nonlinear wave problem [12, Problem 4]. We compare the two algorithms for a range of discretization points, where the matrix becomes more nonnormal (and hence the problem more difficult) as the resolution increases.

7.1. Matrix cosine. Figure 7.1 shows the results for the matrix cosine. For the transformation-free case of `cosm_new` we see from the $(1, 2)$ plot that `costay` was the most accurate algorithm for about 60 percent of the matrices, as shown by the $\alpha = 1$ ordinate, but was less reliable than `cosm_new`, as shown by the relative position of the curves for $\alpha \geq 2$. Indeed visible in the $(1, 1)$ plot is one relative error returned by `costay` (represented by the green triangle), which was $\gg$1, as opposed to 9e-8 and 3e-2 for `cosm_new` and `cosm`, respectively. The condition number of this problem was 4.8e9, so we would expect a forward error of order 10^{-7} from a forward stable algorithm. From the $(2, 1)$ and $(2, 2)$ plots we see that none of the algorithms is always backward stable and that `cosm_new` shows a small advantage over `costay`. The largest backward errors were $\gg$1, 2.1e1, and 1.4e-2 for `costay`, `cosm`, and `cosm_new`, respectively.

Also visible in the $(1, 1)$ plot is one case where `cosm` returned a relative error > 1 (denoted by the black triangle) whilst `cosm_new` returned an answer with essentially full accuracy.

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

The $(1, 1)$ and $(1, 2)$ plots of Figure 7.4 show the cost of each algorithm in multiples of n^3 flops using the transformation-free and Schur versions, respectively. We see that the transformation-free version of `cosm_new` is marginally more expensive than `costay` and `cosm` in most cases but is significantly cheaper than the latter on occasion. The Schur version has a relatively stable cost of around $32n^3$ flops, due to the fixed overhead of the Schur decomposition, which could be advantageous for highly oscillatory differential equations [36], where the matrices have large eigenvalues and hence a heavy scaling may be needed, requiring a large number of (now triangular) matrix multiplications. The precise criterion under which the Schur algorithm is cheaper than the transformation-free version was explained at the end of section 4.

7.2. Matrix sine. For our second experiment, since there are no other algorithms dedicated to computing the matrix sine, we compare `sinm_new` against the use of `costay` to compute $\sin A = \cos(A - \pi I/2)$. Our comparison of these two algorithms is shown in Figure 7.2. For the transformation-free algorithm we see that `sinm_new` has significantly better forward error and backward error performance than `costay`. The use of the Schur decomposition improves the forward stability of `sinm_new`: the forward errors in the $(3,1)$ plot are always within a factor 15 of the condition number times u as opposed to 186 for the $(1,1)$ plot. As in the previous experiment we see that `costay` is not reliable as it returns a relative error and a backward error $\gg 1$ in one test case.

The cost of each algorithm is given in the $(2,1)$ and $(2,2)$ plots of Figure 7.4 for the transformation-free and Schur versions, respectively. In each case `sinm_new` is generally more expensive than `costay`, and using the Schur decomposition gives a fairly stable cost of around $32n^3$ flops.

7.3. Matrix cosine and sine. Our third experiment compares `cosmsinm_new` to `cosmsinm`. For each test matrix we show the largest of the two errors in evaluating the matrix cosine and sine: if a particular test matrix results in backward or forward errors e_c and e_s for the cosine and sine respectively we plot $\max(e_c, e_s)$. In the plots the quantity $\mathrm{cond}(f, A)u$ denotes the larger of $\mathrm{cond}(\cos, A)u$ and $\mathrm{cond}(\sin, A)u$. It is clear from Figure 7.3 that the new algorithm has significantly better forward and backward error performance than `cosmsinm`. Plots of the individual errors for the sine and cosine have similar forms. The largest relative errors returned by `cosmsinm` were 2.9e2 and 1e3 for the matrix cosine and sine, respectively (both achieved for the first test matrix in the plot), while the corresponding errors for `cosmsinm_new` were 1.5e-7 and 5e-7.

Plots showing the cost of each algorithm, both avoiding and utilizing an initial Schur decomposition, are given in the $(3,1)$ and $(3,2)$ positions of Figure 7.4, respectively. We see that the transformation-free version of `cosmsinm_new` is slightly more expensive than `cosmsinm`, but allowing an initial Schur decomposition makes our new algorithm cheaper in some of the test cases.

7.4. Triangular matrices. Figures 7.5–7.7 show the results of applying the algorithms to matrices that are already (quasi)-triangular, obtained by taking the (real) Schur form of each matrix before passing it to the competing algorithms. The new algorithms are greatly superior to the existing ones in terms of both forward and backward error, often achieving values of order u. By comparison with Figures 7.1–7.3 it is clear that the Schur decomposition is a significant source of error in our algorithms.

The cost of the competing algorithms for triangular matrices is shown in Figure 7.8. In the majority of cases our new algorithms are more efficient than the alternatives.

7.5. Wave discretization problem. Our final experiment compares the forward errors of `cosm_new` and `costay` when computing the cosine of a matrix arising from the semidiscretization of the wave equation [12, Problem 4]

$$\frac{\partial^2 u}{\partial t^2} - a(x)\frac{\partial^2 u}{\partial x^2} + \alpha u = f(t, x, u), \tag{7.5}$$

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

$$f(t,x,u) = u^5 - a(x)^2u^3 + \frac{a(x)^5}{4}\sin^2(20t)\cos(10t), \quad a(x) = 4x(1-x). \tag{7.6}$$

With Dirichlet boundary conditions the solution is $u(x,t) = a(x)\cos(10t)$. If we perform a semidiscretization in the spatial variable with mesh size $1/n$ we obtain a system of the form (1.3) with parametrized matrix

$$A = n^2 \begin{bmatrix} 2a(x_1) & -a(x_1) & & & \\ -a(x_2) & 2a(x_2) & -a(x_2) & & \\ & \ddots & \ddots & \ddots & \\ & & -a(x_{n-2}) & 2a(x_{n-2}) & -a(x_{n-2}) \\ & & & -a(x_{n-1}) & 2a(x_{n-1}) \end{bmatrix} + \alpha I. \tag{7.7}$$

This matrix becomes increasingly nonnormal as n grows.

In Figure 7.9 we show the forward errors of the two algorithms when computing $\cos A$ for a range of n as α varies between 0 and 10. The results shown are for the transformation-free version of `cosm_new`; similar results were obtained using the Schur decomposition. For $n = 10$, both algorithms behave in a forward stable manner, with `costay` generally more accurate. As n increases `cosm_new` becomes significantly more accurate than `costay`, the latter showing signs of forward instability and having errors varying much less smoothly with α.

8. Concluding remarks. The goal of this work was to derive algorithms for computing the matrix sine and cosine—both separately and together—that are backward stable in exact arithmetic, thereby providing a more rigorous foundation than for previous algorithms, all of which are based on bounding absolute or forward errors of the function of a scaled matrix. Algorithms with this form of backward stability are already available for the matrix exponential and matrix logarithm. A key finding is that Padé approximants of the matrix cosine do not lend themselves to deriving backward stable algorithms, while those for the matrix sine put strong constraints on the spectral radius of the matrix. We therefore introduced new rational approximants obtained from Padé approximants of the exponential, which yield backward stable approximants of the sine and cosine with no a priori limit on the spectral radius. We also gave the first multiple angle formula-based algorithm for the matrix sine, which uses the triple angle formula in order to avoid the cosines that would be needed by the double angle formula. Experiments show that the new algorithms behave in a forward stable manner in floating point arithmetic, have better backward stability properties than their competitors, and are especially effective for triangular matrices.

A remaining open question is why the second double angle formula in (6.1) performs better in floating point arithmetic than the first in Algorithm 6.2 for simultaneous computation of the sine and cosine.

Appendix. Evaluating rational approximants with the Paterson–Stockmeyer method.

It was mentioned in sections 4–6 that we can efficiently compute the rational approximants c_m and s_m using the Paterson–Stockmeyer scheme [30]. Given a matrix polynomial $p_m(X) = \sum_{k=0}^{m} a_k X^k$ we can write it in the form

$$p_m(X) = \sum_{k=0}^{\ell} g_k(X)(X^\tau)^k, \tag{A.1}$$

Table A.1

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

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

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

$$g_k(X) = \begin{cases} a_{\tau k+\tau-1}X^{\tau-1} + \cdots + a_{\tau k}I, & k = 0\colon \ell - 1, \\ a_m X^{m-\tau\ell} + \cdots + a_{\tau\ell}I, & k = \ell. \end{cases} \tag{A.2}$$

From this it can be shown that the cost (in matrix multiplications) of evaluating the polynomial, using Horner's method for (A.1), is

$$\tau + \ell - 1 - \phi(\tau, m), \tag{A.3}$$

where $\phi(\tau, m) = 1$ if τ divides m and is equal to 0 otherwise [21, pp. 73–74]. We can find the optimal τ by minimizing the cost function (A.3) over $\tau = 1\colon m$.

To apply this to the matrix cosine, we see that the rational approximations obtained in section 2.2 are of the form

$$c_m(x) = \frac{\sum_{k=0}^{m} a_k x^{2k}}{\sum_{k=0}^{m} b_k x^{2k}}, \tag{A.4}$$

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

$$2\left\lfloor \frac{m}{\tau} \right\rfloor + \tau - 2\phi(\tau, m). \tag{A.5}$$

The values $\pi(c_m(X))$ in Table 4.1 were generated by minimizing (A.5) over $\tau = 1 : m$ for each m.

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

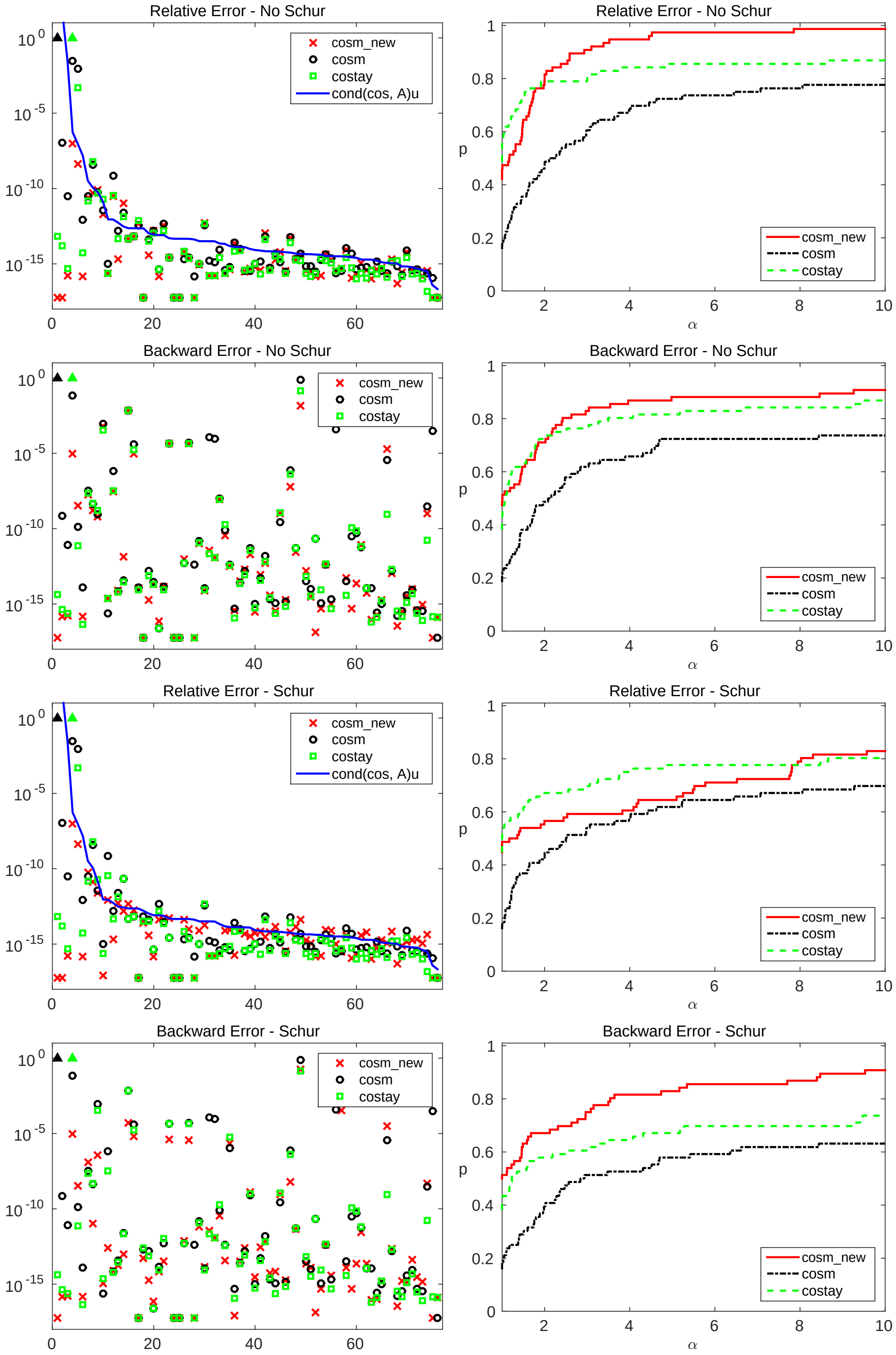

FIG. 7.1. *The forward and backward errors of competing algorithms for the matrix cosine, for the* 76 *test matrices. The first four plots are for the transformation-free version of Algorithm* 4.2*, whereas for the remaining four plots an initial Schur decomposition is used. The results are ordered by decreasing condition number. Coloured triangles indicate errors* > 1.

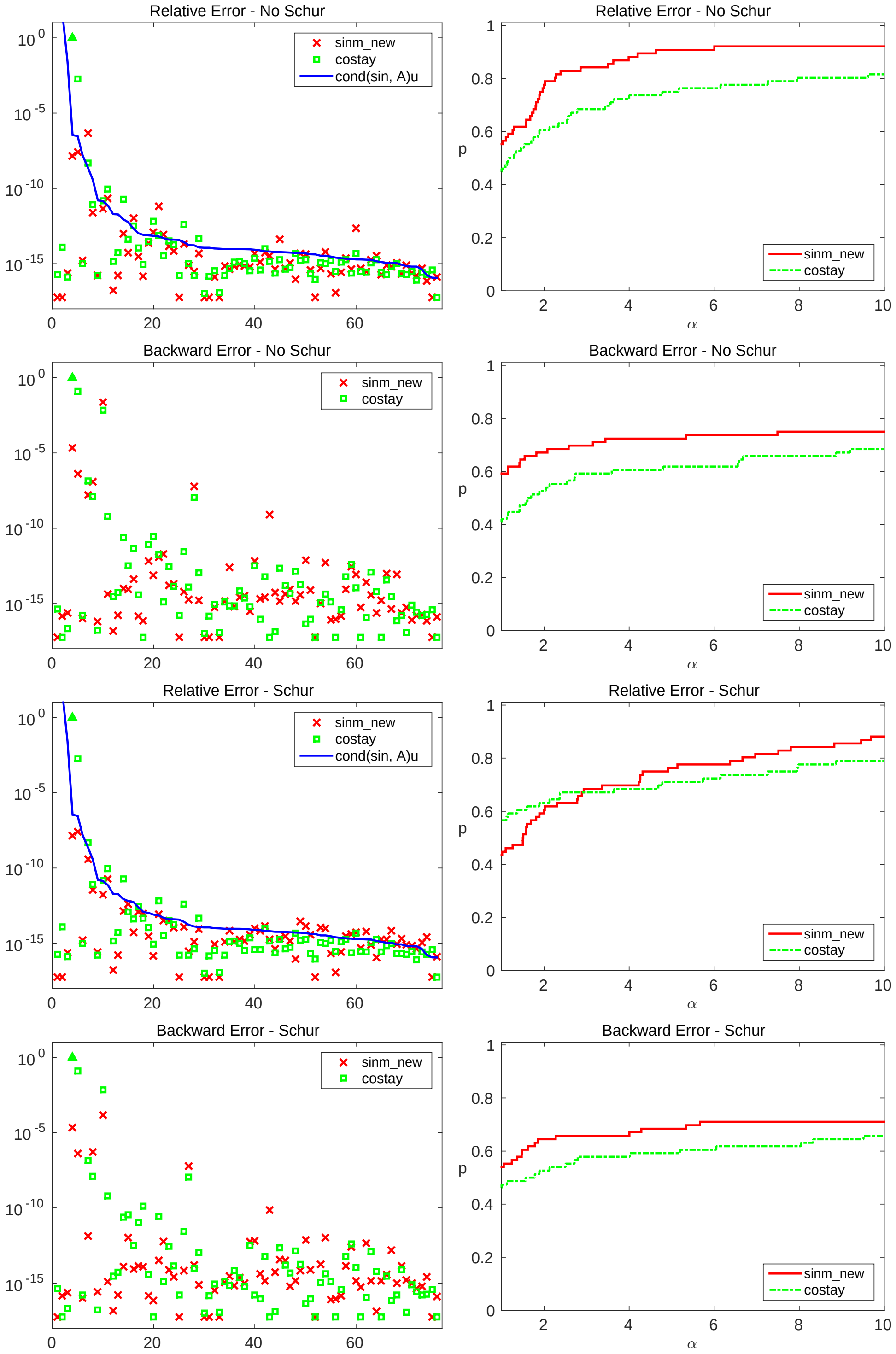

FIG. 7.2. *The forward and backward errors of competing algorithms for the matrix sine, for the* 76 *test matrices. The first four plots are for the transformation-free version of Algorithm* 5.2*, whereas for the remaining four plots an initial Schur decomposition is used. The results are ordered by decreasing condition number. Coloured triangles indicate errors* > 1.

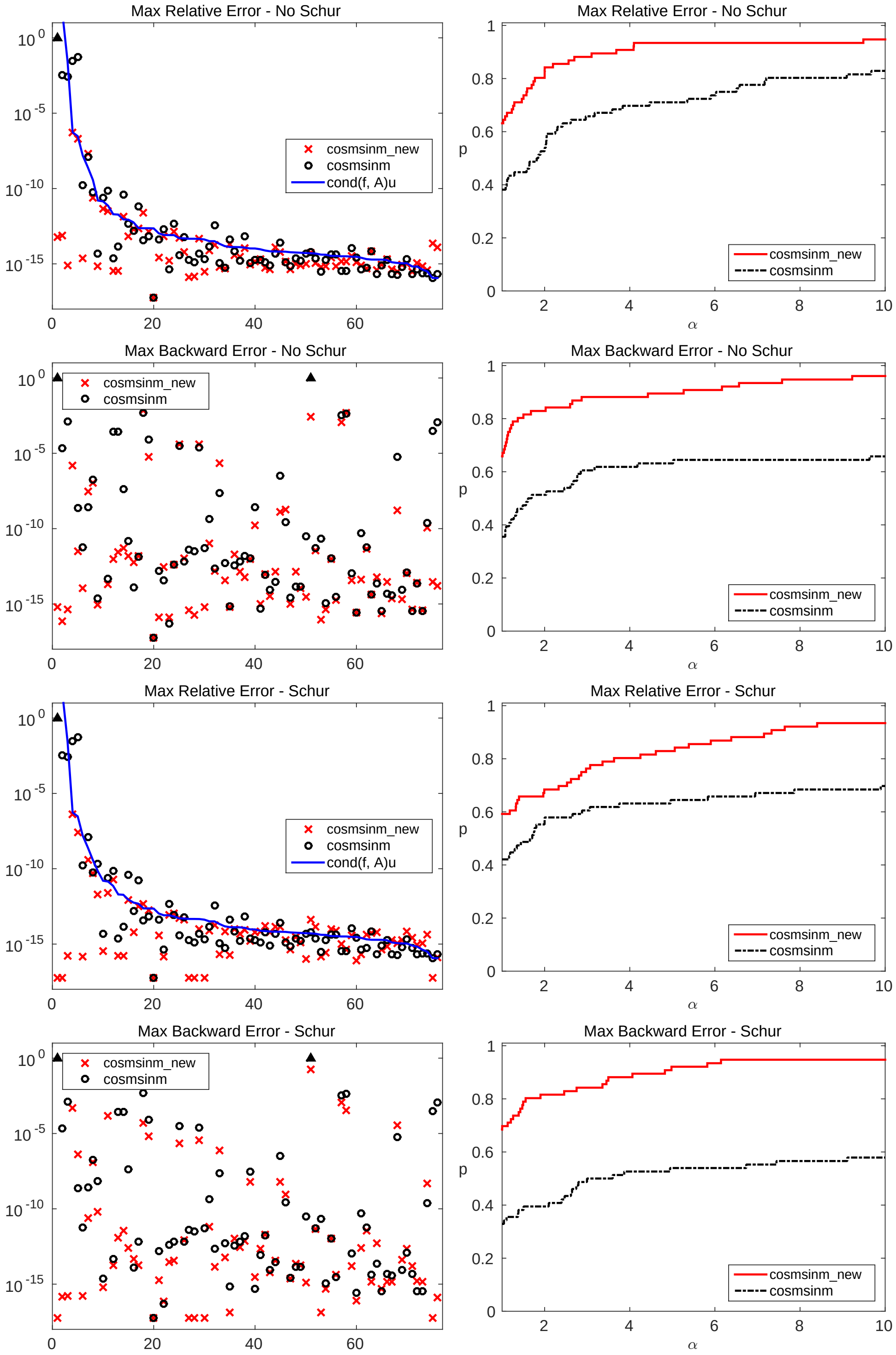

FIG. 7.3. *The maximum forward and backward errors of competing algorithms for the matrix cosine and sine, for the 76 test matrices. The first four plots are for the transformation-free version of Algorithm 6.2, while an initial Schur decomposition is used for the lower four plots. The results are ordered by decreasing condition number. Coloured triangles indicate errors > 1.*

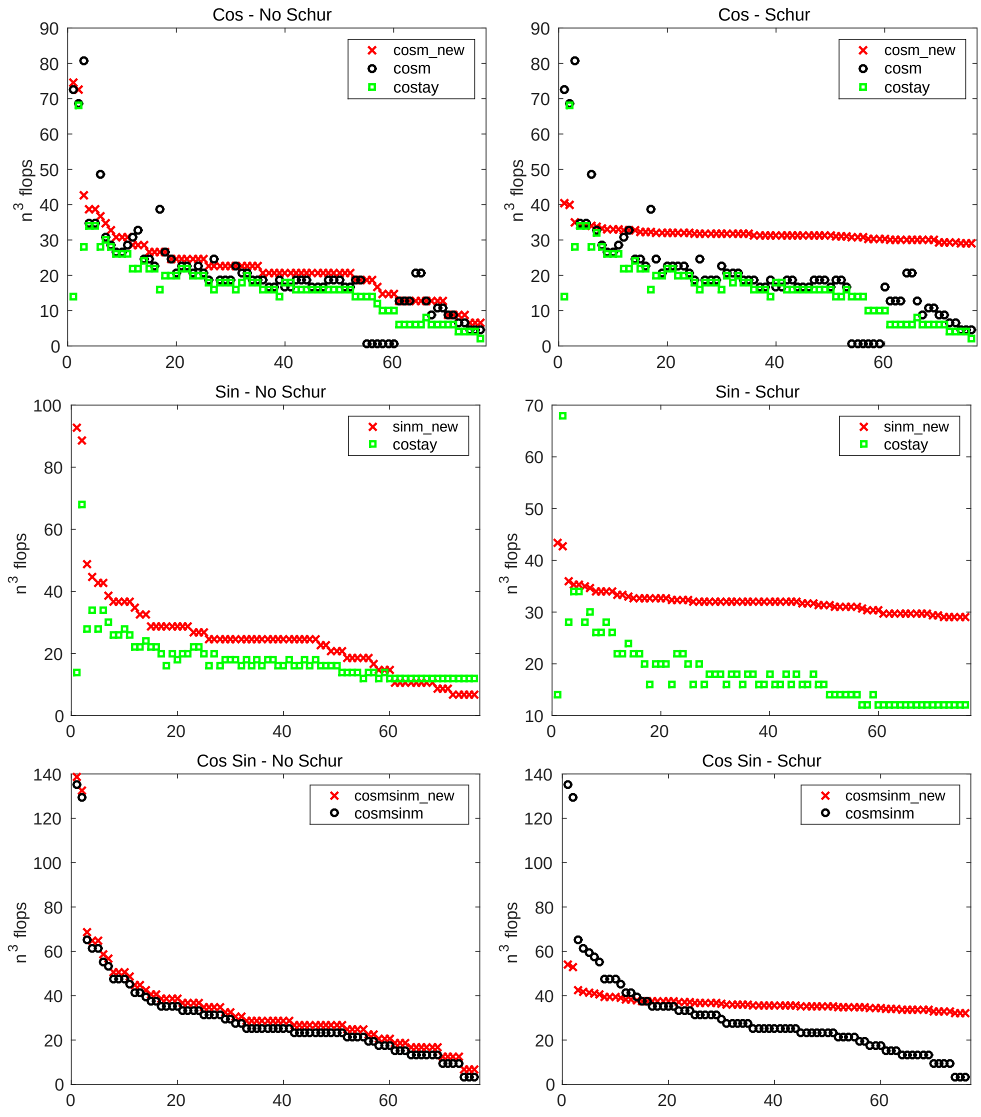

FIG. 7.4. *Cost plots for the experiments in sections 7.1, 7.2, and 7.3. The first row shows the cost of computing the matrix cosine whilst the second and third rows show the cost of computing the matrix sine and both functions together. The left column corresponds to the transformation-free versions of our new algorithms, whilst the right corresponds to an initial Schur decomposition. All plots are ordered by decreasing cost of our new algorithms.*

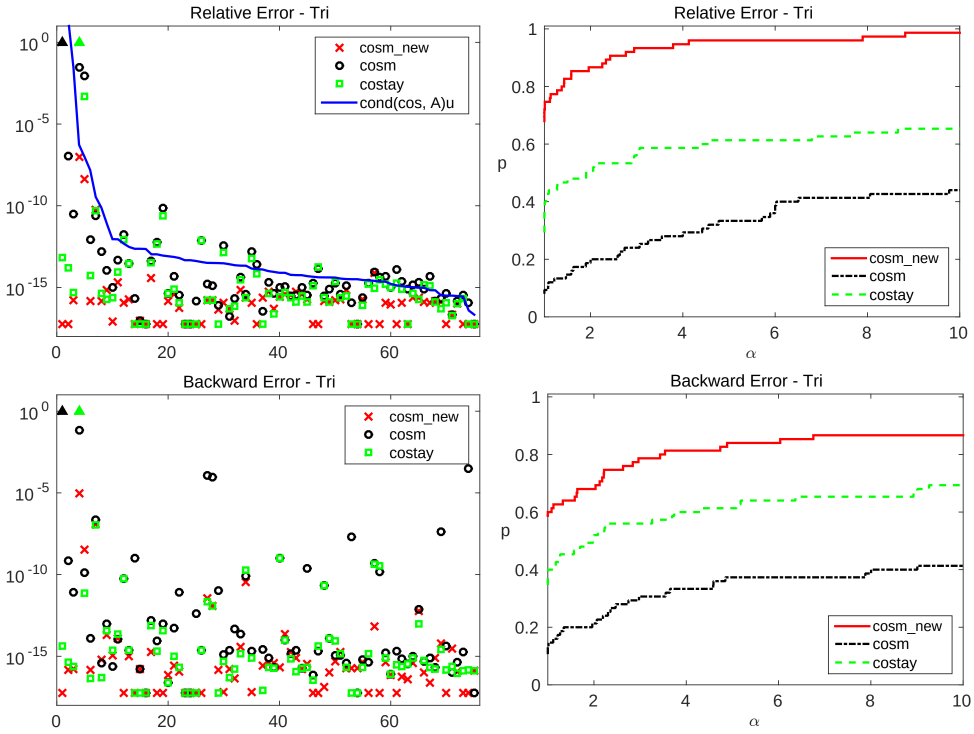

Fig. 7.5. *Forward and backward errors of competing algorithms for the matrix cosine for the triangular Schur form of the* 76 *test matrices. The results are ordered by decreasing condition number. Coloured triangles indicate errors* > 1.

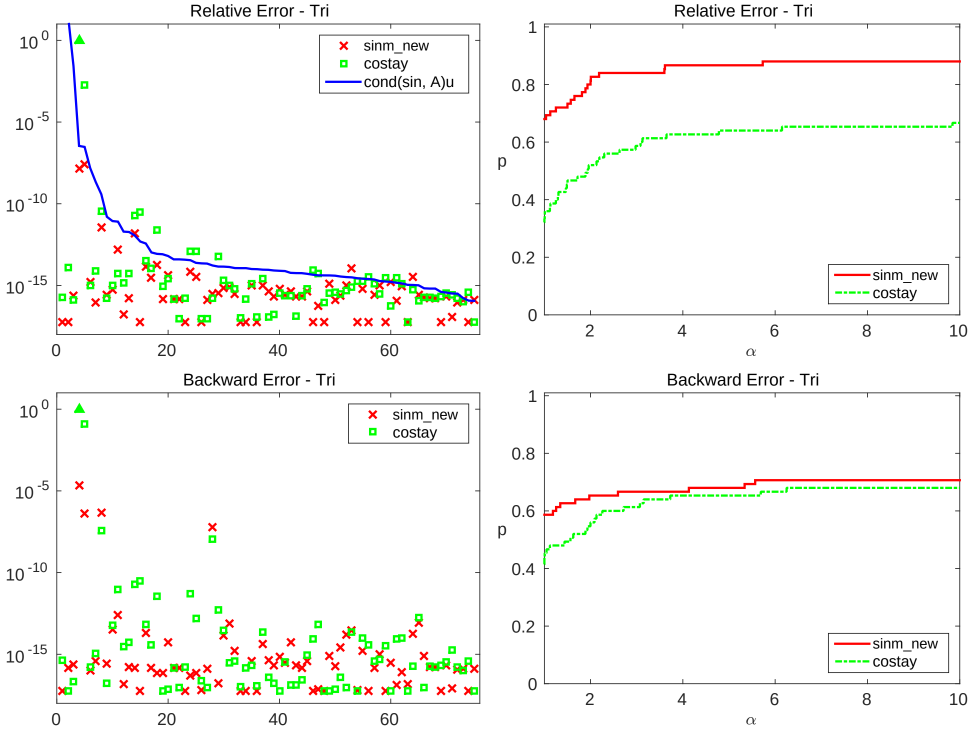

FIG. 7.6. *The forward and backward errors of competing algorithms for the matrix sine for the triangular Schur form of the* 76 *test matrices. The results are ordered by decreasing condition number. Coloured triangles indicate errors* > 1.

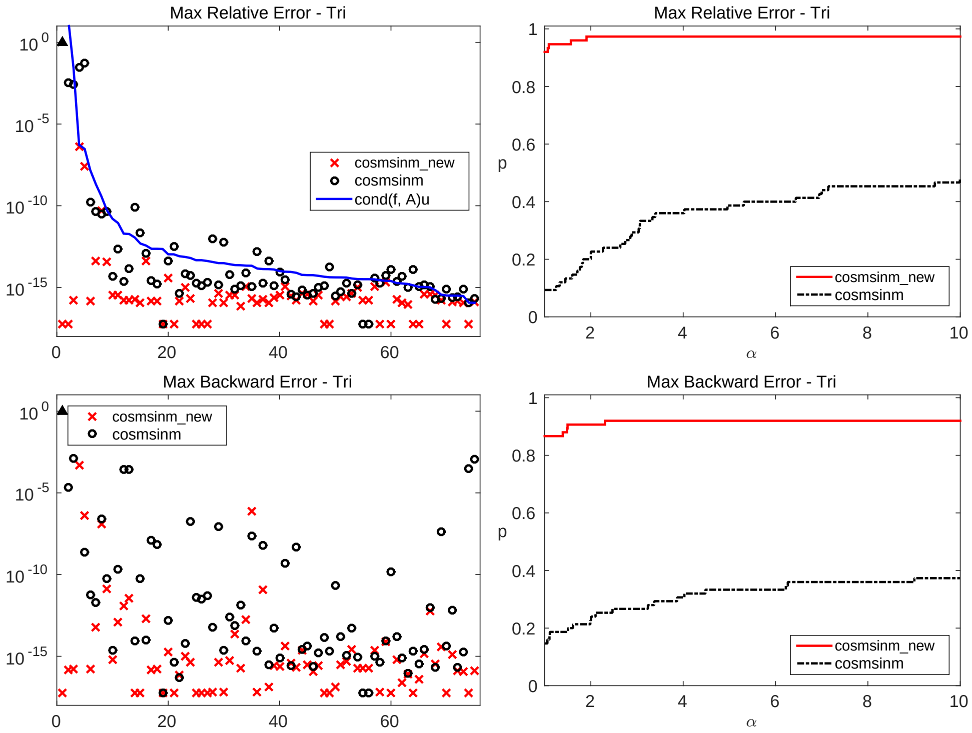

FIG. 7.7. *The maximum forward and backward errors of competing algorithms for the matrix cosine and sine, for the triangular Schur form of the 76 test matrices. The results are ordered by decreasing condition number. Coloured triangles indicate errors* > 1.

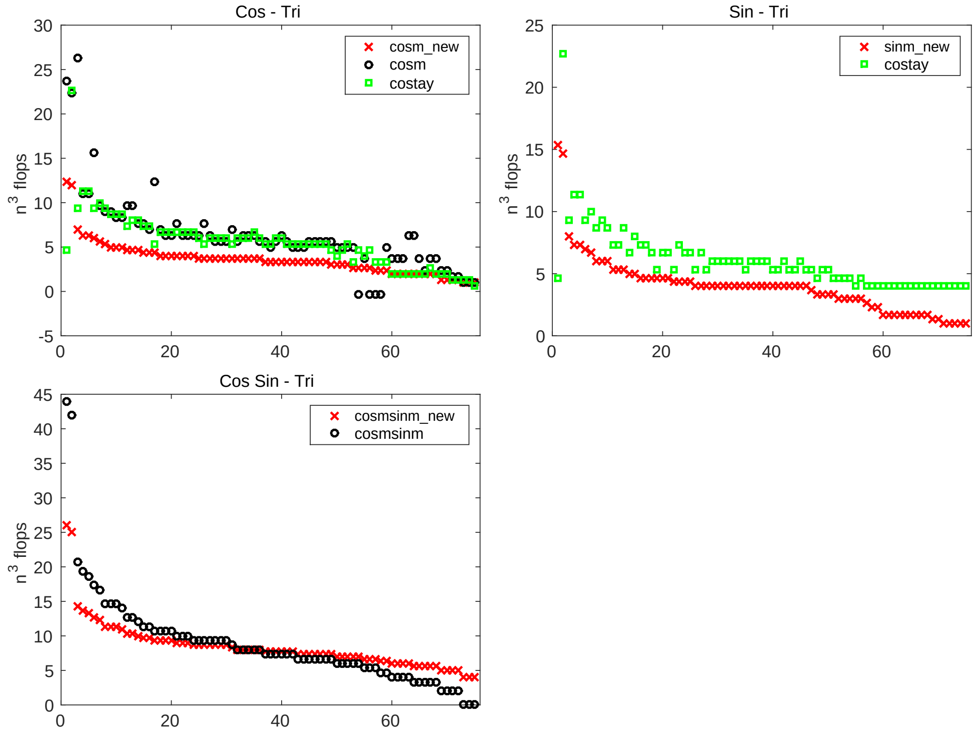

Fig. 7.8. *Cost plots for all the algorithms run on triangular matrices. All plots are ordered by decreasing cost of our new algorithms.*

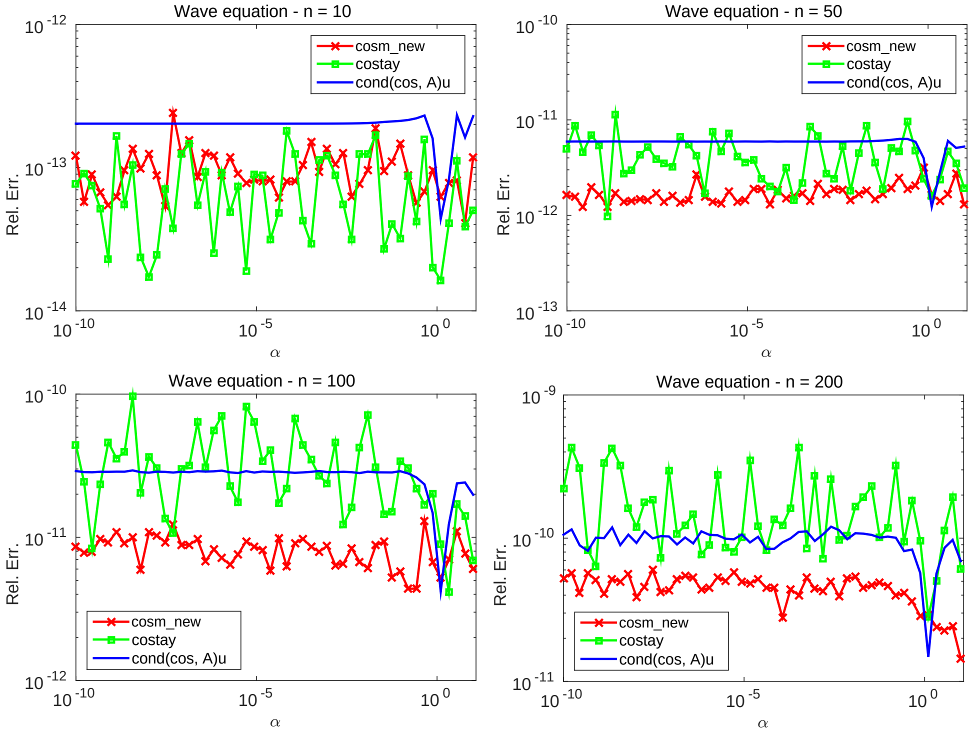

FIG. 7.9. *Forward errors of* `cosm_new` *and* `costay` *for the matrix* (7.7) *that arises from the semidiscretization of a nonlinear wave equation with parameter* α*. The matrix becomes increasingly nonnormal as* n *increases.*

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. Improved inverse scaling and squaring algorithms for the matrix logarithm. *SIAM J. Sci. Comput.*, 34(4):C153–C169, 2012.

[3] Awad H. Al-Mohy, Nicholas J. Higham, and Samuel D. Relton. Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. *SIAM J. Sci. Comput.*, 35(4):C394–C410, 2013.

[4] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. Third edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. xxvi+407 pp. ISBN 0-89871-447-8.

[5] George A. Baker, Jr. *Essentials of Padé Approximants*. Academic Press, New York, 1975. xi+306 pp.

[6] John P. Coleman. Rational approximations for the cosine function; P-acceptability and order. *Numer. Algorithms*, 3:143–158, 1992.

[7] E. B. Davies. Approximate diagonalization. *SIAM J. Matrix Anal. Appl.*, 29(4):1051–1064, 2007.

[8] Edvin Deadman and Nicholas J. Higham. Testing matrix function algorithms using identities. MIMS EPrint 2014.13, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, March 2014. 15 pp.

[9] J. Diblík, D. Ya. Khusainov, J. Lukáčová, and M. Růžičková. Control of oscillating systems with a single delay. *Advances in Difference Equations*, 108218:1–15, 2010.

[10] Nicholas J. Dingle and Nicholas J. Higham. Reducing the influence of tiny normwise relative errors on performance profiles. *ACM Trans. Math. Software*, 39(4):24:1–24:11, 2013.

[11] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.

[12] J. M. Franco. New methods for oscillatory systems based on ARKN methods. *Appl. Numer. Math.*, 56(8):1040–1053, 2006.

[13] F. R. Gantmacher. *The Theory of Matrices*, volume one. Chelsea, New York, 1959. x+374 pp. ISBN 0-8284-0131-4.

[14] V. Grimm and M. Hochbruck. Rational approximation to trigonometric operators. *BIT*, 48: 215–229, 2008.

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

[16] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005. xxiii+382 pp. ISBN 0-89871-578-4.

[17] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.maths.manchester.ac.uk/~higham/mctoolbox`.

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

[19] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[20] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.

[21] 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.

[22] Nicholas J. Higham and Lijing Lin. A Schur–Padé algorithm for fractional powers of a matrix. *SIAM J. Matrix Anal. Appl.*, 32(3):1056–1078, 2011.

[23] Nicholas J. Higham and Lijing Lin. An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives. *SIAM J. Matrix Anal. Appl.*, 34(3):1341–1360, 2013.

[24] Nicholas J. Higham and Samuel D. Relton. Higher order Fréchet derivatives of matrix functions and the level-2 condition number. *SIAM J. Matrix Anal. Appl.*, 35(3):1019–1037, 2014.

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

[26] 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.

[27] Arieh Iserles and S. P. Nørsett. *Order Stars*. Chapman and Hall, London, 1991. xi+248 pp. ISBN 0-411-35260-5.

[28] Arne Magnus and Jan Wynn. On the Padé table of $\cos z$. *Proc. Amer. Math. Soc.*, 47(2): 361–367, 1975.

[29] Dominik L. Michels, Gerrit A. Sobottka, and Andreas G. Weber. Exponential integrators for stiff elastodynamic problems. *ACM Trans. Graph.*, 33(1):7:1–7:20, 2014.

[30] Michael S. Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[31] Jorge Sastre, Javier Ibáñez, Pedro Ruiz, and Emilio Defez. Efficient computation of the matrix cosine. *Appl. Math. Comput.*, 219(14):7575–7585, 2013.

[32] 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.

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

[34] Jaita Pankaj Sharma and Raju K. George. Controllability of matrix second order systems: A trigonometric matrix approach. *Electronic Journal of Differential Equations*, 2007(80): 1–14, 2007.

[35] Lloyd N. Trefethen. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2013. viii+305 pp. ISBN 978-1-611972-39-9.

[36] Bin Wang, Kai Liu, and Xinyuan Wu. A Filon-type asymptotic approach to solving highly oscillatory second-order initial value problems. *J. Comp. Phys.*, 243:210–223, 2013.

[37] Ding Yuan and Warnick Kernan. Explicit solutions for exit-only radioactive decay chains. *J. Appl. Phys.*, 101(9):094907 1–12, 2007.