A Multiprecision Derivative-Free Schur–Parlett Algorithm for Computing Matrix Functions

Higham, Nicholas J. and Liu, Xiaobo

2020

MIMS EPrint: **2020.19**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

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

ISSN 1749-9097

A MULTIPRECISION DERIVATIVE-FREE SCHUR–PARLETT ALGORITHM FOR COMPUTING MATRIX FUNCTIONS*

NICHOLAS J. HIGHAM† AND XIAOBO LIU†

Abstract. The Schur–Parlett algorithm, implemented in MATLAB as `funm`, computes a function $f(A)$ of an $n \times n$ matrix A by using the Schur decomposition and a block recurrence of Parlett. The algorithm requires the ability to compute f and its derivatives, and it requires that f has a Taylor series expansion with a suitably large radius of convergence. We develop a version of the Schur–Parlett algorithm that requires only function values and uses higher precision arithmetic to evaluate f on the diagonal blocks of order greater than 2 (if there are any) of the reordered and blocked Schur form. The key idea is to compute by diagonalization the function of a small random diagonal perturbation of each triangular block, where the perturbation ensures that diagonalization will succeed. This multiprecision Schur–Parlett algorithm is applicable to arbitrary functions f and, like the original Schur–Parlett algorithm, it generally behaves in a numerically stable fashion. Our algorithm is inspired by Davies's randomized approximate diagonalization method, but we explain why that is not a reliable numerical method for computing matrix functions. We apply our algorithm to the matrix Mittag–Leffler function and show that it yields results of accuracy similar to, and in some cases much greater than, the state of the art algorithm for this function. The algorithm will be useful for evaluating any matrix function for which the derivatives of the underlying function are not readily available or accurately computable.

Key words. multiprecision algorithm, multiprecision arithmetic, matrix function, Schur decomposition, Schur–Parlett algorithm, Parlett recurrence, randomized approximate diagonalization, matrix Mittag–Leffler function

AMS subject classifications. 65F60

1. Introduction.

The need to compute matrix functions arises in many applications in science and engineering. Specialized methods exist for evaluating particular matrix functions, including the scaling and squaring algorithm for the matrix exponential [1], [28] Newton's method for matrix sign function [23, Chap. 5], [33], and the inverse scaling and squaring method for the matrix logarithm [2], [27]. See [25] for links to software for these and other methods. For some functions a specialized method is not available, in which case a general purpose algorithm is needed. The Schur–Parlett algorithm [8] computes a general function f of a matrix, with the function dependence restricted to the evaluation of f on the diagonal blocks of the re-ordered and blocked Schur form. It evaluates f on the nontrivial diagonal blocks via a Taylor series, so it requires the derivatives of f and it also requires the Taylor series to have a sufficiently large radius of convergence. However, the derivatives are not always available or accurately computable.

We develop a new version of the Schur–Parlett algorithm that requires only the ability to evaluate f itself and can be used whatever the distribution of the eigenvalues. Our algorithm handles close or repeated eigenvalues by an idea inspired by Davies's idea of randomized approximate diagonalization [7] together with higher precision arithmetic. We therefore assume that as well as the arithmetic of the working precision, with unit roundoff u, we can compute at a higher precision with unit roundoff $u_h < u$, where u_h can be arbitrarily chosen. Higher precisions will necessarily be done in software, and so will be expensive, but we aim to use them as little as possible.

*Version dated September 7, 2020.

Funding: This work was supported by Engineering and Physical Sciences Research Council grant EP/P020720/1 and the Royal Society.

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

We note that multiprecision algorithms have already been developed for the matrix exponential [12] and the matrix logarithm [11]. Those algorithms are tightly coupled to the functions in question, whereas here we place no restrictions on the function. Indeed the new algorithm greatly expands the range of functions f for which we can reliably compute $f(A)$. A numerically stable algorithm for evaluating the Lambert W function of a matrix was only recently developed [13]. Our algorithm can readily compute this function, as well as other special functions and multivalued functions for which the Schur–Parlett algorithm is not readily applicable.

In section 2 we review the Schur–Parlett algorithm. In section 3 we describe Davies's randomized approximate diagonalization and explain why it cannot be the basis of a reliable numerical algorithm. In section 4 we describe our new algorithm for evaluating a function of a triangular matrix using only function values. In section 5 we use this algorithm to build a new Schur–Parlett algorithm that requires only function values and we illustrate its performance on a variety of test problems. We apply the algorithm to the matrix Mittag–Leffler function in section 6 and compare it with a special purpose algorithm for this function. Conclusions are given in section 7.

We will write "normal (0,1) matrix" to mean a random matrix with elements independently drawn from the normal distribution with mean 0 and variance 1. We will use the Frobenius norm, $\|A\|_F = (\sum_{i,j} |a_{ij}|^2)^{1/2}$, and the p-norms $\|A\|_p = \max\{\, \|Ax\|_p : \|x\|_p = 1 \,\}$, where $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$.

2. Schur–Parlett algorithm. The Schur–Parlett algorithm [8] for computing a general matrix function $f(A)$ is based on the Schur decomposition $A = QTQ^* \in \mathbb{C}^{n\times n}$, with $Q \in \mathbb{C}^{n\times n}$ unitary and $T \in \mathbb{C}^{n\times n}$ upper triangular. Since $f(A) = Qf(T)Q^*$, computing $f(A)$ reduces to computing $f(T)$, the same function evaluated at a triangular matrix. If the function of the square diagonal blocks $F_{ii} = f(T_{ii})$ can be computed, the off-diagonal blocks F_{ij} of $f(T)$ can be obtained using the block form of Parlett's recurrence [31],

$$T_{ii}F_{ij} - F_{ij}T_{jj} = F_{ii}T_{ij} - T_{ij}F_{jj} + \sum_{k=i+1}^{j-1} (F_{ik}T_{kj} - T_{ik}F_{kj}), \quad i < j, \tag{2.1}$$

from which F_{ij} can be computed either a block superdiagonal at a time or a block row or block column at a time. To address the potential problems caused by close or equal eigenvalues in two diagonal blocks of T, Davies and Higham [8] devised a scheme with a blocking parameter $\delta > 0$ to reorder T into a partitioned upper triangular matrix $\widetilde{T} = U^*TU = (\widetilde{T}_{ij})$ by a unitary similarity transformation such that

- eigenvalues λ and μ from any two distinct diagonal blocks $\widetilde{T}_{ii}$ and $\widetilde{T}_{jj}$ satisfy $\min|\lambda - \mu| > \delta$, and
- the eigenvalues of every block $\widetilde{T}_{ii}$ of size larger than 1 are well clustered in the sense that either all the eigenvalues of $\widetilde{T}_{ii}$ are equal or for every eigenvalue λ_1 of $\widetilde{T}_{ii}$ there is an eigenvalue λ_2 of $\widetilde{T}_{ii}$ with $\lambda_1 \neq \lambda_2$ such that $|\lambda_1 - \lambda_2| \leq \delta$.

To evaluate $f(\widetilde{T}_{ii})$, the Schur–Parlett algorithm expands f in a Taylor series about $\sigma = \mathrm{trace}(\widetilde{T}_{ii})/m_i$, the mean of the eigenvalues of $\widetilde{T}_{ii} \in \mathbb{C}^{m_i\times m_i}$,

$$f(\widetilde{T}_{ii}) = \sum_{k=0}^{\infty} \frac{f^{(k)}(\sigma)}{k!} (\widetilde{T}_{ii} - \sigma I)^k, \tag{2.2}$$

truncating the series after an appropriate number of terms. All the derivatives of f up to a certain order are required in (2.2), where that order depends on how quickly the

powers of $\widetilde{T}_{ii}-\sigma I$ decay. Moreover, for the series (2.2) to converge we need $\lambda-\sigma$ to lie in the radius of convergence of the series for every eigenvalue λ of $\widetilde{T}_{ii}$. Obviously, this procedure for evaluating $f(\widetilde{T})$ may not be appropriate if it is difficult or expensive to accurately evaluate the derivatives of f or if the Taylor series has a finite radius of convergence.

3. Approximate diagonalization. If $A \in \mathbb{C}^{n\times n}$ is diagonalizable then $A = VDV^{-1}$, where $D = \mathrm{diag}(d_i)$ is diagonal and V is nonsingular, so $f(A) = Vf(D)V^{-1} = V\,\mathrm{diag}(f(d_i))V^{-1}$ is trivially obtained. For normal matrices, V can be chosen to be unitary and this approach is an excellent way to compute $f(A)$. However, for nonnormal A the eigenvector matrix V can be ill-conditioned, in which case an inaccurate computed $f(A)$ can be expected in floating-point arithmetic [23, sect. 4.5].

A way to handle a nonnormal matrix is to perturb it before diagonalizing it. Davies [7] suggested perturbing A to $\widetilde{A} = A + E$, computing the diagonalization $\widetilde{A} = VDV^{-1}$, and approximating $f(A)$ by $f(\widetilde{A}) = Vf(D)V^{-1}$. This approach relies on the fact that even if A is defective, $A + E$ is likely to be diagonalizable because the diagonalizable matrices are dense in $\mathbb{C}^{n\times n}$. Davies measured the quality of the approximate diagonalization by the quantity

$$\sigma(A, V, E, \epsilon) = \kappa_2(V)\epsilon + \|E\|_2, \tag{3.1}$$

where the condition number $\kappa_2(V) = \|V\|_2\|V^{-1}\|_2$ and ϵ can be thought of as the unit roundoff. Minimizing over E and V (since V is not unique) gives

$$\sigma(A, \epsilon) = \inf_{E,V} \sigma(A, V, E, \epsilon),$$

which is a measure of the best approximate diagonalization that this approach can achieve. Davies conjectured that

$$\sigma(A, \epsilon) \le c_n\epsilon^{1/2} \tag{3.2}$$

for some constant c_n, where $\|A\|_2 \le 1$ is assumed, and he proved the conjecture for Jordan blocks and triangular Toeplitz matrices (both with $c_n = 2$) and for arbitrary 3×3 matrices (with $c_3 = 4$). Davies's conjecture was recently proved by Banks, Kulkarni, Mukherjee, and Srivastava [4, Thm. 1.1] with $c_n = 4n^{3/2} + 4n^{3/4} \le 8n^{3/2}$. Building on the solution of Davies' conjecture a randomized algorithm with low computational complexity is developed in [5] for approximately computing the eigensystem. Note that (3.2) suggests it is sufficient to choose E such that $\|E\|_2 \approx \epsilon^{1/2}$ in order to obtain an error of order $\epsilon^{1/2}$.

As we have stated it, the conjecture is over $\mathbb{C}^{n\times n}$. Davies's proofs of the conjecture for Jordan blocks and triangular Toeplitz matrices have E real when A is real, which is desirable. In the proof in [4], E is not necessarily real when A is real. However, Jain, Sah, and Sawhney [26] have proved the conjecture for real A and real perturbations E.

The matrix E can be thought of as a regularizing perturbation for the diagonalization. For computing matrix functions, Davies suggests taking E as a random matrix and gives empirical evidence that normal (0,1) matrices E scaled so that $\|E\|_2 \approx u^{1/2}$ are effective at delivering a computed result with error of order $u^{1/2}$ when $\|A\|_2 \le 1$. One of us published a short MATLAB code to implement this idea [24],[1] as a way of computing $f(A)$ with error of order $u^{1/2}$. However, this approach does not give

[1] https://gist.github.com/higham/6c00f62e48c1b0116f2e9a8f43f2e02a

TABLE 3.1
Relative errors $\|f(\widetilde{A}) - f(A)\|_F/\|f(A)\|_F$ for approximation from randomized approximate diagonalization with $\|E\|_F = u^{1/2}\|A\|_F$ to the square root of the Jordan block $J(\lambda) \in \mathbb{R}^{n\times n}$.

λ	$n = 10$	$n = 20$	$n = 30$
1.0	7.46×10^{-9}	7.22×10^{-9}	9.45×10^{-9}
0.5	1.22×10^{-7}	3.42×10^{-4}	1.44
0.1	1.14	1.00	1.00

TABLE 3.2
Values of $\|L_f(A)\|_F$ corresponding to the results in Table 3.1.

λ	$n = 10$	$n = 20$	$n = 30$
1.0	1.41	2.01	2.46
0.5	2.62×10^{3}	8.55×10^{8}	4.75×10^{14}
0.1	1.13×10^{16}	4.99×10^{30}	3.24×10^{54}

a reliable numerical method for approximating matrix functions. The reason is that (3.1) does not correctly measure the effect on $f(A)$ of perturbing A by E. For small E, for any matrix norm we have

$$\|f(A+E) - f(A)\| \lesssim \|L_f(A,E)\| \le \|L_f(A)\|\|E\|, \tag{3.3}$$

where $L_f(A,E)$ is the Fréchet derivative of f at A in the direction E and $\|L_f(A)\| = \max\{\, \|L_f(A,E)\| : \|E\| = 1 \,\}$ [23, sect. 3.1]. Hence while σ in (3.1) includes $\|E\|_2$, the change in f induced by E is as much as $\|L_f(A)\|_2\|E\|_2$, and the factor $\|L_f(A)\|_2$ can greatly exceed 1.

A simple experiment with $\epsilon = u$ illustrates the point. All the experiments in this paper are carried out in MATLAB R2020a with a working precision of double ($u \approx 1.1 \times 10^{-16}$). We take A to be an $n \times n$ Jordan block with eigenvalue λ and $f(A) = A^{1/2}$ (the principal matrix square root), for which $\|L_f(A)\|_F = \|(I \otimes A^{1/2} + (A^{1/2})^T \otimes I)^{-1}\|_2$ [21]. The diagonalization and evaluation of $f(\widetilde{A})$ is done at the working precision. In Table 3.1 we show the relative errors $\|f(A) - f(\widetilde{A})\|_F/\|f(A)\|_F$, where E is a (full) normal (0,1) matrix scaled so that $\|E\|_F = u^{1/2}\|A\|_F$ and the reference solution $f(A)$ is computed in 100 digit precision using the function `sqrtm` from the Multiprecision Computing Toolbox [30]. For $\lambda = 1$ we obtain an error of order $u^{1/2}$, but the errors grow as λ decreases and we achieve no correct digits for $\lambda = 0.1$. The reason is clear from Table 3.2, which shows the values of the term that multiplies $\|E\|_F$ in (3.3), which are very large for small λ. We stress that increasing the precision at which $f(\widetilde{A})$ is evaluated does not reduce the errors; the damage done by the perturbation E cannot be recovered.

In this work we adapt the idea of diagonalizing after a regularizing perturbation, but we take a new approach that does not depend on Davies's theory.

4. Evaluating a function of a triangular matrix. Our new algorithm uses the same blocked and re-ordered Schur form as the Schur–Parlett algorithm. The key difference from that algorithm is how it evaluates a function of a triangular block. Given an upper triangular block $T \in \mathbb{C}^{m\times m}$ of the reordered Schur form and an arbitrary function f we apply a regularizing perturbation with norm of order u and evaluate $f(T)$ at precision $u_h < u$. We expect m generally to be small, in which case the overhead of using higher precision arithmetic is small. In the worst case this

approach should be competitive with the worst case for the Schur–Parlett algorithm [8, Alg. 2.6], since (2.2) requires up to $O(m^4)$ (working precision) flops.

We will consider two different approaches.

4.1. Approximate diagonalization with full perturbation. Our first approach is a direct application of approximate diagonalization, with $\epsilon = u^2$. Here, E is a multiple of a (full) normal (0,1) matrix with norm of order $\epsilon^{1/2} = u$. Whereas Davies considered only matrices A of 2-norm 1, we wish to allow any norm, and the norm of E should scale with that of A. We will scale E so that

$$\|E\|_F = u \max_{i,j} |t_{ij}|. \tag{4.1}$$

We evaluate $f(T+E)$ by diagonalization at precision $u_h = u^2$ and hope to obtain a computed result with relative error of order u. Diagonalization requires us to compute the Schur decomposition of a full matrix $T + E$, and it costs about $28\frac{2}{3}m^3$ flops in precision u_h.

Although we do not expect this approach to provide a numerical method that works well for all problems, in view of the discussion and example in section 3, it is a useful basis for comparison with the new method in the next section.

4.2. Approximate diagonalization with triangular perturbation. Instead of regularizing by a full perturbation, we now take the perturbation E to be an upper triangular normal (0,1) matrix, normalized by (4.1). An obvious advantage of taking E triangular is that $\widetilde{T} = T + E$ is triangular and we can compute the eigenvectors (needed for diagonalization) by substitution, which is substantially more efficient than computing the complete eigensystem of a full matrix. Note that the diagonal entries of $\widetilde{T}$ are distinct with probability 1, albeit perhaps differing by as little as order $\|E\|_F$.

This approach can be thought of as indirectly approximating the derivatives by finite differences. Indeed for $m = 2$ we have

$$f(T) = \begin{bmatrix} f(t_{11}) & t_{12} f[t_{11}, t_{22}] \\ 0 & f(t_{22}) \end{bmatrix}, \quad f[t_{11}, t_{22}] = \begin{cases} \dfrac{f(t_{22}) - f(t_{11})}{t_{22} - t_{11}}, & t_{11} \neq t_{22}, \\ f'(t_{11}), & t_{11} = t_{22}, \end{cases} \tag{4.2}$$

so when $t_{11} = t_{22}$, perturbing to $\widetilde{t}_{11} \neq \widetilde{t}_{22}$ results in a first order finite difference approximation to $f'(t_{11})$. For $m > 2$, these approximations are intertwined with the evaluation of $f(T)$.

In order to find the eigenvector matrix V of the perturbed triangular matrix $\widetilde{T} = T + E$ we need to compute a set of m linearly independent eigenvectors v_i, $i = 1 : m$. This can be done by solving at precision u_h the m triangular systems

$$(\widetilde{T} - \widetilde{t}_{ii} I) v_i = 0, \quad i = 1 : m, \tag{4.3}$$

where we set v_i to be 1 in its ith component, zero in components $i + 1 : m$, and solve for the first $i - 1$ components by substitution. Thus the matrix V is upper triangular. Careful scaling is required to avoid overflow [35].

To summarize, we compute in precision u_h the diagonalization

$$\widetilde{T} = V D V^{-1}, \quad D = \operatorname{diag}(\lambda_i), \tag{4.4}$$

where in practice the λ_i will be distinct. We then form $f(\widetilde{T}) = V f(D) V^{-1}$ in precision u_h, which involves solving a multiple right-hand side triangular system with a

triangular right-hand side. The cost of the computation is $\sum_{k=1}^m k^2 + m^3/3 = 2m^3/3$ flops in precision u_h.

We expect the error in the computed approximation $\widehat{F}$ to $F = f(\widetilde{T})$ to be bounded approximately by (cf. [23, p. 82])

$$\frac{\|F - \widehat{F}\|_1}{\|F\|_1} \lesssim \kappa_1(V) \frac{\|f(D)\|_1}{\|f(\widetilde{T})\|_1} u_h.$$

(The choice of norm is not crucial; the 1-norm is convenient here.) We will use this bound to determine u_h. Note that

$$\frac{1}{\kappa_1(V)} \le \frac{\|f(D)\|_1}{\|f(\widetilde{T})\|_1} \le \kappa_1(V).$$

Since we do not know $\|f(\widetilde{T})\|_1$ a priori we will approximate $\|f(D)\|_1/\|f(\widetilde{T})\|_1$ by 1 (the geometric mean of its bounds), and hence we will use

$$\frac{\|F - \widehat{F}\|_1}{\|F\|_1} \lesssim \kappa_1(V) u_h. \tag{4.5}$$

Since we need to know how to choose u_h before we compute V, we need an estimate of $\kappa(V)$ based only on $\widetilde{T}$. Since we are using a triangular perturbation its regularizing effect will be less than that of a full perturbation, so we expect that we may need a precision higher than double the working precision.

Demmel [3, sect. 5.3], [9] showed that $\kappa_2(V)$ is within a factor m of $\max_i \|P_i\|_2$, where P_i is the spectral projector corresponding to the eigenvalue λ_i. Writing

$$\widetilde{T} = \begin{bmatrix} \widetilde{t}_{11} & \widetilde{t}_{12}^* \\ 0 & \widetilde{T}_{22} \end{bmatrix},$$

the spectral projector for the eigenvalue $\lambda_1 = \widetilde{t}_{11}$ is, with the same partitioning,

$$P_1 = \begin{bmatrix} 1 & p^* \\ 0 & 0 \end{bmatrix}, \quad p^* = \widetilde{t}_{12}^*(\widetilde{t}_{11} I - \widetilde{T}_{22})^{-1}. \tag{4.6}$$

From (4.6) we have

$$\|P_1\|_1 = \max(1, \|p\|_\infty) \le \max\bigl(1, \|\widetilde{t}_{12}\|_\infty \|(\widetilde{t}_{11} I - \widetilde{T}_{22})^{-1}\|_1\bigr).$$

Now for any $m \times m$ upper triangular matrix U we have the bound [22, Thm. 8.12, Prob. 8.5]

$$\|U^{-1}\|_1 \le \frac{1}{\alpha}\left(\frac{\beta}{\alpha} + 1\right)^{m-1}, \quad \alpha = \min_i |u_{ii}|, \quad \beta = \max_{i<j} |u_{ij}|. \tag{4.7}$$

This bound will be very pessimistic if we apply it to $\widetilde{t}_{11} I - \widetilde{T}_{22}$, because for the bound to be a good approximation it is necessary that many diagonal elements of U are of order α, yet $\widetilde{t}_{11} I - \widetilde{T}_{22}$ will typically have only a few (if any) small elements. Let us group the $\widetilde{t}_{ii}$ according to the Schur–Parlett blocking criteria described in section 2, with blocking parameter $\delta = \delta_1$. Suppose the largest block has size $k \ge 2$ and suppose, without loss of generality, that it comprises the first k diagonal elements of $\widetilde{T}$. Then

we will approximate $\|(\widetilde{t}_{11}I - \widetilde{T}_{22})^{-1}\|_1$ by $\|(\widetilde{t}_{11}I - \widetilde{T}_{22}(1\colon k-1, 1\colon k-1))^{-1}\|_1$, and bound it by (4.7), leading to the approximation

$$\max_i \|P_i\|_1 \approx \max_{i<j} |\widetilde{t}_{ij}| \frac{\Big(\dfrac{\max_{i<j}|\widetilde{t}_{ij}|}{c_m u} + 1\Big)^{k-2}}{c_m u},$$

where the parameter c_m is such that $c_m u \approx \min_i |w_{ii}|$, where the w_{ii} are the diagonal elements of $\widetilde{t}_{11}I - \widetilde{T}_{22}(1\colon k-1, 1\colon k-1)$, and hence this is an estimate of $\kappa_1(V)$ by Demmel's result.

We are aiming for an error of order u, so from (4.5) we need $\kappa_1(V)u_h \lesssim u$, which gives the requirement

$$u_h \lesssim \frac{c_m u^2}{\max_{i<j}|\widetilde{t}_{ij}|\Big(\dfrac{\max_{i<j}|\widetilde{t}_{ij}|}{c_m u} + 1\Big)^{k-2}}, \quad k \geq 2. \tag{4.8}$$

In the case $k = 2$, the bound (4.8) is $u_h \lesssim c_m u^2 / \max_{i<j}|\widetilde{t}_{ij}| = O(u^2)$. If the largest block size is $k = 1$, we use $u_h = u^2$ (corresponding to Davies's conjecture (3.2)) since we do not expect $\kappa(V)$ to be so large that a precision higher than double the working precision is required.

We summarize this algorithm, based on a triangular perturbation, in Algorithm 4.1, where we have

$$u_h = \begin{cases} u^2, & k = 1, \\ \min\big(u^2, \text{right-hand side of (4.8)}\big), & k \geq 2. \end{cases} \tag{4.9}$$

Importantly, for 2×2 diagonal blocks of T with distinct eigenvalues we are able to use (4.2) at the working precision.

In summary, we make an upper triangular perturbation E of norm $O(u\|T\|_1)$ to T and evaluate $f(T+E)$. We choose the precision of the evaluation in order to be sure of obtaining an accurate evaluation of $f(T+E)$. Our approach differs fundamentally from Davies's randomized approximate diagonalization. Our triangular E has a lesser regularizing effect than a full one, so it results in a potentially larger $\kappa(V)$, but our choice of u_h takes this into account. On the other hand, since our perturbation E is upper triangular and of order $u\|T\|_1$, it corresponds to a backward error of order u and so is harmless. A full perturbation E cannot be interpreted as a backward error for the $f(T)$ evaluation as it perturbs the zeros in the lower triangle of T.

The analysis above is unchanged if E is diagonal, so we allow E to be chosen as diagonal or upper triangular in Algorithm 4.1 and will compare the two choices experimentally in the next section.

We now discuss the choice of δ_1 and c_m.

The parameter c_m is such that $c_m u \approx \min_i |w_{ii}|$, where the w_{ii} are the diagonal elements of $\widetilde{t}_{11}I - \widetilde{T}_{22}(1\colon k-1, 1\colon k-1)$. We will determine c_m by considering the extreme case when $\min_i |w_{ii}|$ is extremely small, which is when all $\widetilde{t}_{ii}$ in the largest block of size $k \leq m$ differ only by the perturbation we added (in which case the t_{ii} are exactly repeated) and thus $\widetilde{t}_{11}I - \widetilde{T}_{22}(1\colon k-1, 1\colon k-1)$ is extremely ill-conditioned. This choice of c_m makes the chosen higher precision $u_h < u^2$ pessimistic when some of the $\widetilde{t}_{ii}$ are close in the sense they are partitioned in the same block by $\delta = \delta_1$ but not all of them are exactly repeated, but it helps to ensure the accuracy of the algorithm

Algorithm 4.1 Multiprecision algorithm for function of a triangular matrix.

Given a triangular matrix $T \in \mathbb{C}^{m\times m}$ and a function f, this algorithm computes $F = f(T)$. It uses arithmetics of unit roundoff u (the working precision), u^2, and possibly a higher precision $u_h \le u^2$. Lines 9–11 are to be executed at precision u^2 and lines 12–16 are to be executed at precision u_h.

1 if $m = 1$, $f_{11} = f(t_{11})$, quit, end
2 if $m = 2$ and $t_{11} \ne t_{22}$
3 $\quad f_{11} = f(t_{11})$, $f_{22} = f(t_{22})$
4 $\quad f_{12} = t_{12}(f_{22} - f_{11})/(t_{22} - t_{11})$
5 $\quad$ quit
6 end
7 Form an $m \times m$ diagonal or upper triangular normal (0,1) matrix N.
8 $E = u(\max_{i,j} |t_{ij}|/\|N\|_F)N$
9 $\widetilde{T} = T + E$
10 $D = \mathrm{diag}(\widetilde{T})$
11 Evaluate u_h by (4.9).
12 if $u_h < u^2$, convert $\widetilde{T}$ and D to precision u_h, end
13 for $i = 1\colon m$
14 $\quad$ Set $(v_i)_i = 1$ and $(v_i)_k = 0$ for $k > i$ and solve the triangular system $(\widetilde{T} - \widetilde{t}_{ii}I)v_i = 0$ for the first $i-1$ components of v_i.
15 end
16 Form $F = Vf(D)V^{-1}$, where $V = [v_1, \dots, v_m]$.
17 Round F to precision u.
18 $f_{ii} = f(t_{ii})$, $i = 1 : m$.

in all cases. However, since the algorithm is to be employed in the next section for computing a function of $T \in \mathbb{C}^{m\times m}$ where generally m (and hence k) is expected to be small, we do not expect this approach to seriously affect the efficiency of the overall algorithm. In the case we are considering we have $|w_{ii}| = |\widetilde{t}_{11} - \widetilde{t}_{ii}| = |e_{11} - e_{ii}|$. The matrix E on line 8 of Algorithm 4.1 has entries $u(\max_{i,j} |t_{ij}|)|\widetilde{n}_{ij}|$, where $\|\widetilde{N}\|_F = 1$, and we expect $|\widetilde{n}_{ij}| \approx 1/m$. This suggests taking $c_m = \theta \max_{i,j} |t_{ij}|/m$ for some constant θ. In our experiments with different choices of θ we found $\theta = 0.5$ to be a good choice.

The blocking parameter $\delta = \delta_1$ is important in determining the largest group size k in (4.8). A smaller δ can potentially group fewer eigenvalues and decrease k, causing a larger u_h to be used. Yet too large a δ can result in a u_h that is much smaller than necessary to achieve the desired accuracy. We have found experimentally that $\delta_1 = 5 \times 10^{-3}$ is a good choice.

4.3. Numerical experiments. In this section we describe a numerical experiment with the methods of sections 4.1 and 4.2 for computing a function of a triangular matrix. Precisions higher than double precision are implemented with the Multiprecision Computing Toolbox [30].

We set the function f to be the exponential, the square root, the sign function, the logarithm, the cosine, and the sine. The algorithms for computing $f(T)$ to be tested are

- `Alg_full`: approximate diagonalization with a full perturbation and $u_h = u^2$, as described in section 4.1,

- `Alg_diag`: Algorithm 4.1 with diagonal E, $c_m = 0.5 \max_{i,j} |t_{ij}|/m$, and $\delta_1 = 5 \times 10^{-3}$.

We use the following matrices, generated from built-in MATLAB functions.

- T_1 = `gallery('kahan',m)`: upper triangular with distinct diagonal elements on the interval $(0, 1]$.
- T_2 = `schur(gallery('smoke',m),'complex')`: Schur factor of the complex matrix whose eigenvalues are the mth roots of unity times $2^{1/m}$.
- T_3 = `schur(randn(m),'complex')`.
- T_4 = `schur(rand(m),'complex')`.
- T_5 = `triu(randn(m))`.
- T_6 = `triu(rand(m))`.
- T_7 = `gallery('jordbloc',m,0.5)`: a Jordan block with eigenvalue 0.5.

Since we are computing the principal matrix square root and the principal logarithm we multiply matrices T_3, T_4, and T_5 by $1 + \mathrm{i}$ for these functions to avoid their eigenvalues being on the negative real axis.

We report the equivalent number of decimal digits for the higher precision u_h used by Algorithm 4.1 for each test matrix in the computation in Table 4.1. Since the outputs of `Alg_full` and `Alg_diag` depend on the random perturbation E, we compute the function of each matrix 10 times and report in Table 4.2 the maximum relative error $\|F - \widehat{F}\|/\|F\|_F$, where F is a reference solution computed by the functions `expm`, `sqrtm`, `logm`, `cosm`, and `sinm` provided by the Multiprecision Computing Toolbox running at 200 digit precision, and rounded back to double precision. We use a diagonal perturbation E in Algorithm 4.1. For the reference solution of the matrix sign function, we run `signm` from the Matrix Function Toolbox [20] at 200 digit precision, and round back to double precision. The same procedure is followed in the experiments in the following sections.

We show in Table 4.2 the quantity $\kappa_f(A)u$, where $\kappa_f(A)$ is the 1-norm condition number [23, Chap. 3] of f at A, which we estimate using the `funm_condest1` function provided by [20]. A numerically stable algorithm will produce forward errors bounded by a modest multiple of $\kappa_f(A)u$.

The results show that Algorithm 4.1 behaves in a numerically stable fashion in every case, typically requiring a higher precision with unit roundoff u_h equal to or not much smaller than u^2. We see that for the same class of matrices the number of digits of precision used is nondecreasing with the matrix size m, which is to be expected since we expect a larger maximum block size (equal to k in (4.8)) for a larger matrix. On the other hand, as expected in view of the discussion in section 3, the randomized approximate diagonalization method `Alg_full` is less reliable and sometimes not accurate at all when f is the matrix square root, the matrix sign function, or the matrix logarithm.

Note that our test matrices here are more general than will arise in the algorithm of the next section, for which the triangular blocks will have clustered eigenvalues.

We repeated this experiment with an upper triangular E in Algorithm 4.1. The errors were of the same order of magnitude as for diagonal E. Since a diagonal E requires slightly less computation, we will take E diagonal in the rest of this paper.

5. Overall algorithm for computing $f(A)$. Our algorithm for computing $f(A)$ follows the framework of the Schur–Parlett algorithm [8]. First the Schur decomposition $A = QTQ^*$ is computed. Then the triangular matrix T is reordered to a partitioned upper triangular matrix $\widetilde{T}$ by a unitary similarity transformation, which is achieved by Algorithms 4.1 and 4.2 in [8, sect. 4]. The diagonal blocks $\widetilde{T}_{ii}$ are

TABLE 4.1
Equivalent number of decimal digits for the higher precision u_h used by Algorithm 4.1 in the computation. 32 digits corresponds to $u_h = u^2$.

	$m = 40$	$m = 80$
T_1 = `gallery('kahan',m)`	34	743
T_2 = `schur(gallery('smoke',m),'complex')`	32	32
T_3 = `schur(randn(m),'complex')`	32	32
T_4 = `schur(rand(m),'complex')`	32	32
T_5 = `triu(randn(m))`	34	53
T_6 = `triu(rand(m))`	34	89
T_7 = `gallery('jordbloc',m,0.5)`	713	1451

computed by Algorithm 4.1 instead of by a Taylor expansion as in the Schur–Parlett algorithm, and the precision u_h used in Algorithm 4.1 is potentially different for each diagonal block. The off-diagonal blocks of $f(\widetilde{T})$ are computed using the block form of the Parlett recurrence. Finally, we undo the unitary similarity transformations from the Schur decomposition and the reordering. This gives Algorithm 5.1.

In Algorithm 5.1 we distinguish a special case: if A is normal, the Schur decomposition becomes $A = QDQ^*$ with D diagonal, and the algorithm simply computes $f(A) = Qf(D)Q^*$. We note that the algorithm preserves the advantages of the Schur–Parlett algorithm that if one wants to compute $f(A) = \sum_i f_i(A)$ then it is not necessary to compute each $f_i(A)$ separately because the Schur decomposition and its reordering can be reused.

Algorithm 5.1 Multiprecision Schur–Parlett algorithm for function of a full matrix.
Given $A \in \mathbb{C}^{n\times n}$ and a function f this algorithm computes $F = f(A)$. It uses arithmetics of unit roundoff u (the working precision), u^2, and possibly higher precisions $u_h \le u^2$ (chosen in Algorithm 4.1). It requires only function values, not derivatives.

1 Compute the Schur decomposition of $A = QTQ^*$.
2 if T is diagonal, $F = Qf(T)Q^*$, quit, end
3 Use Algorithms 4.1 and 4.2 in [8, sect. 4] with $\delta > 0$ to reorder T into
  a block $m \times m$ upper triangular matrix $\widetilde{T} = U^*TU$.
4 for $i = 1\colon m$
5 Use Algorithm 4.1 (with a diagonal E) to evaluate $F_{ii} = f(\widetilde{T}_{ii})$.
6 for $j = i-1\colon -1\colon 1$
7 Solve the Sylvester equation (2.1) for F_{ij}.
8 end
9 end
10 $F = QUFU^*Q^*$

In the reordering and blocking of the Schur–Parlett framework the blocking parameter $\delta > 0$, described in section 2, needs to be specified. A large δ leads to greater separation of the eigenvalues of the diagonal blocks, which improves the accuracy of the solutions to the Sylvester equations. In this respect, there is a significant difference between Algorithm 5.1 and the standard Schur–Parlett algorithm: the latter algorithm cannot tolerate too large a δ because it slows down convergence of the Taylor series expansion, meaning that more terms may be needed (or the series may simply not converge). Since Algorithm 4.1 performs well irrespective of the eigenvalue distribution we can choose δ without consideration of the accuracy of the evaluation

TABLE 4.2
Maximal normwise relative errors for Algorithm 4.1 with a diagonal E (`Alg_diag`) and the method of approximate diagonalization with full perturbation (`Alg_full`).

	$f = \exp$			$f = \mathrm{sqrt}$		
	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$
$T_1, m = 40$	7.4e-17	3.2e-17	7.2e-15	2.0e-16	2.3e-12	3.1e-10
$T_1, m = 80$	5.5e-17	2.8e-17	1.3e-14	4.8e-15	6.7e-6	1.6e-11
$T_2, m = 40$	6.3e-17	3.0e-17	2.2e-15	3.8e-16	2.2e-12	1.7e-9
$T_2, m = 80$	4.2e-17	2.8e-17	4.5e-15	6.6e-16	7.4e-7	1.9e-12
$T_3, m = 40$	1.4e-16	9.2e-17	8.2e-15	7.0e-17	7.5e-17	2.9e-14
$T_3, m = 80$	1.5e-16	7.6e-17	5.7e-14	1.1e-16	7.6e-17	5.7e-14
$T_4, m = 40$	5.8e-16	1.5e-16	3.6e-15	4.9e-16	5.7e-16	2.5e-14
$T_4, m = 80$	1.3e-15	1.6e-16	8.1e-13	2.7e-15	4.4e-15	2.9e-13
$T_5, m = 40$	7.8e-17	7.8e-17	2.0e-14	3.3e-15	7.5e-6	1.1e-11
$T_5, m = 80$	6.4e-17	6.2e-17	4.9e-14	7.3e-15	1.0	2.0e-18
$T_6, m = 40$	8.0e-15	4.4e-17	4.7e-14	1.3e-14	1.7e-13	5.6e-11
$T_6, m = 80$	3.2e-17	5.9e-17	2.8e-13	5.1e-15	5.2e-2	3.0e-13
$T_7, m = 40$	1.4e-17	2.1e-16	6.5e-15	3.0e-16	2.6e-6	7.6e-13
$T_7, m = 80$	1.4e-24	1.6e-15	1.3e-14	5.1e-16	1.0	1.2e-18

	$f = \mathrm{sign}$			$f = \log$		
	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$
$T_1, m = 40$	0	3.9e-26	9.8e-22	3.7e-16	2.1e-12	2.6e-10
$T_1, m = 80$	0	1.4e-19	3.4e1	4.7e-15	8.1e-6	1.4e-12
$T_2, m = 40$	3.9e-16	1.9e-12	1.6e-9	3.8e-16	1.5e-12	9.5e-10
$T_2, m = 80$	7.5e-16	5.4e-7	2.2e-12	6.9e-16	4.9e-7	1.1e-12
$T_3, m = 40$	1.1e-16	1.1e-16	2.8e-14	8.4e-17	8.8e-17	2.5e-14
$T_3, m = 80$	4.1e-16	3.3e-16	2.2e-13	1.1e-16	1.3e-16	5.3e-14
$T_4, m = 40$	7.7e-16	1.0e-15	3.4e-14	9.9e-16	1.3e-15	3.5e-14
$T_4, m = 80$	2.0e-15	2.8e-15	1.5e-13	3.5e-15	5.8e-15	2.9e-13
$T_5, m = 40$	2.4e-15	1.8e-6	3.2e-12	3.2e-15	7.8e-6	2.7e-12
$T_5, m = 80$	4.8e-15	1.0	4.0e-19	4.2e-15	1.0	4.2e-19
$T_6, m = 40$	0	4.5e-16	5.4e-21	3.2e-15	2.3e-13	1.1e-10
$T_6, m = 80$	0	2.3e-16	2.2e3	5.1e-15	8.8e-2	3.4e-14
$T_7, m = 40$	0	2.7e-16	3.5e1	4.1e-16	2.8e-6	1.4e-13
$T_7, m = 80$	0	1.0	1.6e2	5.6e-16	1.0	1.8e-19

	$f = \cos$			$f = \sin$		
	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$	`Alg_diag`	`Alg_full`	$\kappa_f(A)u$
$T_1, m = 40$	3.3e-17	3.4e-17	8.6e-15	4.5e-17	4.5e-17	1.1e-14
$T_1, m = 80$	2.4e-17	2.3e-17	1.9e-14	3.9e-17	4.4e-17	4.1e-14
$T_2, m = 40$	4.4e-17	2.4e-17	3.9e-15	5.0e-17	3.4e-17	4.7e-15
$T_2, m = 80$	4.0e-17	2.5e-17	7.8e-15	4.0e-17	2.6e-17	8.3e-15
$T_3, m = 40$	1.3e-16	8.4e-17	1.6e-14	1.4e-16	7.3e-17	1.7e-14
$T_3, m = 80$	1.6e-16	7.5e-17	8.6e-14	1.3e-16	7.4e-17	8.2e-14
$T_4, m = 40$	3.6e-16	2.6e-16	1.0e-14	3.6e-16	2.5e-16	8.2e-15
$T_4, m = 80$	4.5e-16	3.1e-16	2.2e-14	4.4e-16	3.1e-16	2.2e-14
$T_5, m = 40$	6.3e-17	6.3e-17	2.2e-14	6.9e-17	6.5e-17	2.3e-14
$T_5, m = 80$	4.8e-17	5.5e-17	2.5e-8	5.5e-17	8.1e-17	2.1e-8
$T_6, m = 40$	3.9e-14	3.0e-17	1.3e-14	6.2e-14	1.4e-16	6.8e-14
$T_6, m = 80$	2.6e-17	4.8e-17	7.6e-14	2.5e-17	3.3e-17	8.3e-14
$T_7, m = 40$	3.2e-17	1.8e-16	3.5e-15	3.1e-17	7.9e-16	3.3e-15
$T_7, m = 80$	2.1e-17	5.1e-16	6.9e-15	1.5e-17	1.2e-15	6.9e-15

of f on the diagonal blocks and larger δ will in general do no harm to accuracy. In the extreme case where δ is so large that one block is employed, Algorithm 5.1 does not solve Sylvester equations and thus avoids the potential error incurred in the process, and in general this is when our algorithm attains its optimal accuracy, but the price

to pay is that it becomes very expensive because higher precision arithmetic is being used on an $n \times n$ matrix. We investigate the choice of δ experimentally in the next subsection.

5.1. Numerical experiments. In the Schur–Parlett algorithm [8] the blocking parameter $\delta = 0.1$ is chosen, which is shown there to perform well most of the time. In order to investigate a suitable value for δ in Algorithm 5.1, we compare the following four algorithms, where "nd" stands for "no derivative".

- `funm_nd_0.1`, Algorithm 5.1 with $\delta = 0.1$;
- `funm_nd_0.2`, Algorithm 5.1 with $\delta = 0.2$;
- `funm_nd_norm`, Algorithm 5.1 with $\delta = 0.1 \max_i |t_{ii}|$; and
- `funm_nd_`∞, Algorithm 5.1 with $\delta = \infty$ (no blocking, so the whole Schur factor T is computed by Algorithm 4.1).

The 35 tested matrices are nonnormal taken from

- the MATLAB `gallery`;
- the Matrix Computation Toolbox [19];
- other MATLAB matrices: `magic`, `rand`, and `randn`.

We set their size to be 32×32, and we also test the above matrices multiplied by $10^{\pm 2}$ to examine the robustness of the algorithms under scaling. We set the function f to be the matrix sine; similar results were obtained with the other functions.

Figure 5.1, in which the solid line is $\kappa_f(A)u$, shows that Algorithm 5.1 with a constant δ is fairly stable under scaling while using a δ that scales with the matrix A (`funm_nd_norm`) can produce large errors when $\|A\|$ is small. This is not unexpected since a smaller δ results in a smaller separation of the blocks and more ill-conditioned Sylvester equations.

In most cases there is no difference in accuracy between the algorithms. The results show no significant benefit of $\delta = 0.2$ over $\delta = 0.1$, and the former produces larger blocks in general so increases the cost.

In general, the choice δ in Algorithm 5.1 must be a balance between speed and accuracy, and the optimal choice of δ will be problem-dependent. We suggest taking $\delta = 0.1$ as the default blocking parameter in Algorithm 5.1.

Next we set the function f to the sine, the cosine, the hyperbolic sine, and the hyperbolic cosine and use the same set of 35 test matrices as in the previous experiment. We compare the following three algorithms:

- `funm`, the built-in MATLAB function implementing the standard Schur–Parlett algorithm [8] with $\delta = 0.1$;
- `funm_nd`, Algorithm 5.1 with $\delta = 0.1$.
- `funm_nd_`∞, Algorithm 5.1 with $\delta = \infty$ (no blocking, so the whole Schur factor T is computed by Algorithm 4.1).

Note that since we are comparing with the Schur–Parlett algorithm `funm` we are restricted to functions f having a Taylor expansion with an infinite radius of convergence and for which derivatives of all orders can be computed. Also, we exclude the exponential, square root, and logarithm because for these functions the specialized codes `expm`, `sqrtm`, and `logm` are preferred to `funm`.

From Figure 5.2 we observe that, overall, there is no significant difference between `funm_nd` and `funm` in accuracy, and `funm_nd_`∞ is superior to the other algorithms in accuracy, as expected.

We list the computational cost of the three algorithms in flops in Table 5.1. We note that the cost of reordering and blocking, and solving the Sylvester equations that are executed in precision u, is usually negligible compared with the overall cost.

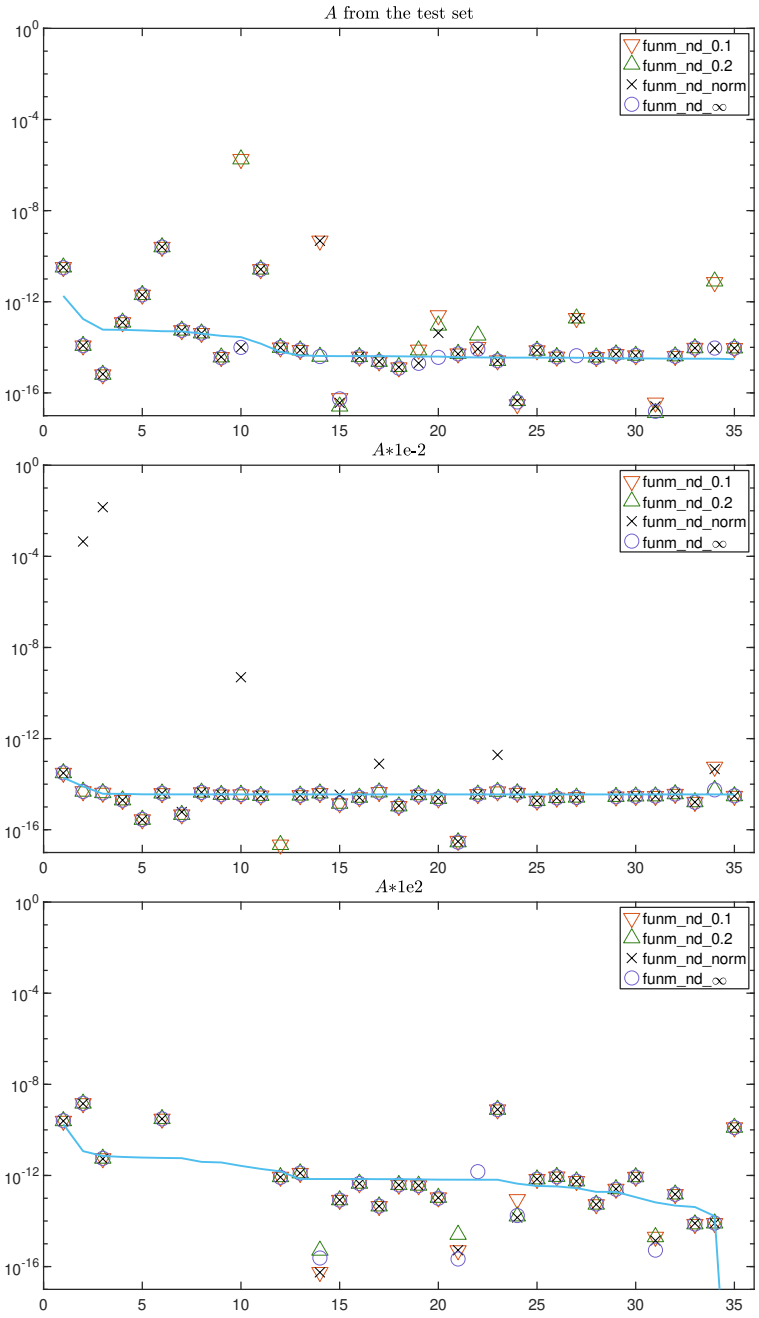

FIG. 5.1. *Normwise relative errors for* `funm_nd_0.1`, `funm_nd_0.2`, `funm_nd_norm` *and* `funm_nd_`∞ *on the test set, for the matrix sine. The solid line is* $\kappa_{\sin}(A)u$.

For more details of the reordering and partitioning processes of T and evaluating the upper triangular part of $f(A)$ via the block Parlett recurrence see [8]. In most cases the blocks are expected to be of much smaller dimension than A, especially when n is large. Obviously, `funm_nd` is not more expensive than `funm_nd_`∞ and it can be

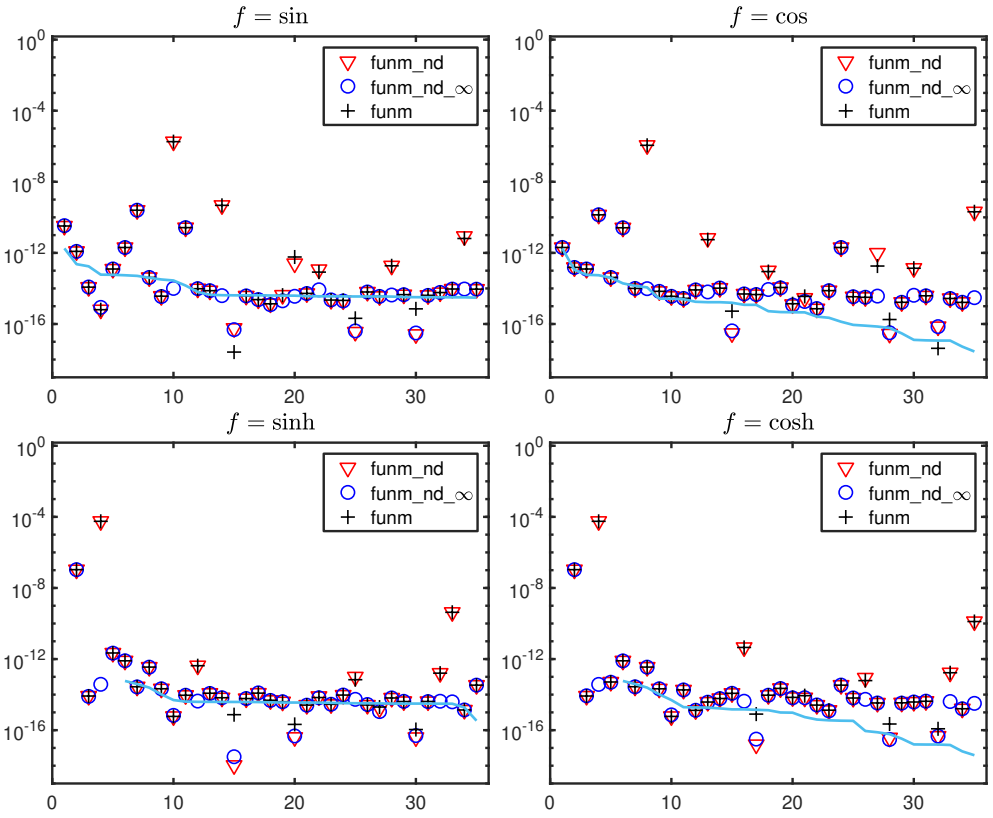

FIG. 5.2. *Normwise relative errors for* `funm`, `funm_nd_`∞ *and* `funm_nd`. *The solid line is* $\kappa_f(A)u$.

substantially cheaper; indeed `funm_nd` requires no higher than the working precision to compute 1×1 and 2×2 diagonal blocks in the Schur form.

Table 5.2 compares in a working precision of double the mean execution times in seconds and the maximal forward errors of `funm`, `funm_nd`, and `funm_nd_`∞ over ten runs, and reports the maximal block size in the reordered and blocked Schur form for each matrix and the maximal number of equivalent decimal digits used by `funm_nd`. We choose $f = \sin$ and $f = \cosh$ and consider the following matrices, generated from built-in MATLAB functions and scaled to different degrees to have non-trivial blocks of the reordered and blocked Schur form in the Schur–Parlett algorithms.

- $A_1 =$ `rand(n)`$/5$.
- $A_2 =$ `randn(n)`$/10$.
- $A_3 =$ `gallery('triw',n,-5)`: upper triangular with 1s on the diagonal and -5s off the diagonal.

We see from Table 5.2 that `funm`, `funm_nd`, and `funm_nd_`∞ provide the same level of accuracy except for one case: $f = \sin$ and A_3. In this case `funm` requires about n Taylor series terms and produces an error several orders of magnitude larger than that of other algorithms. For the matrix A_3 with repeated eigenvalues, `funm_nd` is much slower than `funm` due to the use of higher precision arithmetic in a large block, and in this case there is no noticeable difference in execution time between `funm_nd` and `funm_nd_`∞, which confirms that the cost of the reordering and blocking in `funm_nd`

TABLE 5.1
Asymptotic cost in flops of `funm`, `funm_nd`, *and* `funm_nd_∞`. *Here,* $n = \sum_{i=1}^{s} m_i$ *is the size of the original matrix* A, s *is the number of the diagonal blocks in the Schur form after reordering and blocking, and* m_i *is the size of the* i*th block.*

	`funm`	`funm_nd`		`funm_nd_∞`	
Precision	u	u	u_h	u	u_h
Flops	$28n^3$ to $n^4/3$	$28n^3$	$2/3\sum_{i=1}^{s} m_i^3$	$28n^3$	$2n^3/3$

TABLE 5.2
Mean execution times (in seconds) and the maximal normwise relative errors over ten runs for `funm`, `funm_nd`, *and* `funm_nd_∞`, *and the maximal block size and the maximal number of equivalent decimal digits used by* `funm_nd`.

	Maximal relative error			Mean execution time (secs)				
$f = \sin$	`funm`	`funm_nd`	`funm_nd_∞`	`funm`	`funm_nd`	`funm_nd_∞`	size	digits
A_1, $n = 40$	4.2e-15	4.2e-15	4.3e-15	3.7e-2	1.4e-1	1.9e-1	11	32
A_2, $n = 40$	4.5e-15	4.5e-15	4.4e-15	4.2e-2	6.4e-2	1.9e-1	4	32
A_3, $n = 40$	1.5e-14	9.4e-17	9.1e-17	3.8e-3	6.8e-2	6.7e-2	40	713
A_1, $n = 100$	6.3e-15	6.3e-15	6.3e-15	1.2e-1	3.3e-1	1.2	15	32
A_2, $n = 100$	6.6e-15	6.6e-15	6.6e-15	2.5e-1	2.6e-1	1.2	3	32
A_3, $n = 100$	1.0e-12	4.0e-17	4.5e-17	3.2e-2	1.0	1.0	100	1824
$f = \cosh$	`funm`	`funm_nd`	`funm_nd_∞`	`funm`	`funm_nd`	`funm_nd_∞`	size	digits
A_1, $n = 40$	1.8e-15	1.8e-15	1.9e-15	3.4e-2	1.2e-1	1.9e-1	11	32
A_2, $n = 40$	3.0e-15	3.1e-15	3.0e-15	3.5e-2	5.7e-2	1.9e-1	4	32
A_3, $n = 40$	7.9e-16	1.2e-16	1.4e-16	2.3e-3	6.0e-2	5.8e-2	40	713
A_1, $n = 100$	2.2e-15	2.2e-15	2.5e-15	1.2e-1	3.4e-1	1.2	15	32
A_2, $n = 100$	5.6e-15	5.6e-15	5.6e-15	2.5e-1	2.8e-1	1.2	3	32
A_3, $n = 100$	8.1e-16	1.9e-17	2.7e-17	3.4e-2	1.1	1.1	100	1824

is negligible. For the randomly generated matrices (A_1 and A_2) `funm` can be up to 3.8 times faster than `funm_nd`, but in some cases when the block size is small `funm_nd` is competitive with `funm` in speed. For these matrices, `funm_nd` is much faster than `funm_nd_∞`.

Finally, we note that Algorithm 5.1 is not restricted only to a working precision of double since its framework is precision independent. For other working precisions suitable values for the parameters c_m, δ_1 and δ may be different, but they can be determined in an approach similar to the one used in this work.

The reason for developing Algorithm 5.1 is that it requires only accurate function values and not derivative values. In the next section we consider a function for which accurate derivative values are not easy to compute.

6. An application to the matrix Mittag-Leffler function. The matrix Mittag–Leffler function is the two-parameter function defined by the convergent series

$$E_{\alpha,\beta}(A) = \sum_{k=0}^{\infty} \frac{A^k}{\Gamma(\alpha k + \beta)},$$

where $A \in \mathbb{C}^{n\times n}$, $\alpha, \beta \in \mathbb{C}$, and $\operatorname{Re}\alpha > 0$. Analogously to the matrix exponential in the solution of systems of linear differential equations, the Mittag–Leffler function plays an important role in the solution of linear systems of fractional differential equations [17], [34], including time-fractional Schrödinger equations [15], [16] and multiterm fractional differential equations [32]. Despite the importance of the matrix

Mittag–Leffler function, little work has been devoted to its numerical computation. In [29], the computation of the action of matrix Mittag–Leffler functions based on Krylov methods is analysed. In [10], the Jordan canonical form and minimal polynomial or characteristic polynomial are considered for computing the matrix Mittag–Leffler function, but this approach is unstable in floating-point arithmetic.

The recent paper by Garrappa and Popolizio [18] employs the Schur–Parlett algorithm to compute the matrix Mittag–Leffler function. The derivatives of the scalar Mittag–Leffler function are given by

$$E_{\alpha,\beta}^{(k)}(z) = \sum_{j=k}^{\infty} \frac{(j)_k}{\Gamma(\alpha j+\beta)} z^{j-k}, \quad k \in \mathbb{N},\ (j)_k := j(j-1)\cdots(j-k+1)$$

and are difficult to compute accurately. Garrappa and Popolizio use three approaches, based on series expansion, numerical inversion of the Laplace transform, and summation formulas to compute the derivatives. They exploit certain identities [18, Props. 3–4] to express high-order derivatives in terms of lower order ones, since they observe that all three methods tend to have reduced accuracy for high order derivatives. In fact, almost all of [18] is devoted to the computation of the derivatives. By combining derivative balancing techniques with algorithms for computing the derivatives the authors show in their experiments that the computed $\widehat{E}_{\alpha,\beta}^{(k)}(z)$ have errors

$$\frac{|E_{\alpha,\beta}^{(k)}(z) - \widehat{E}_{\alpha,\beta}^{(k)}(z)|}{1+|E_{\alpha,\beta}^{(k)}(z)|}$$

that lie "in a range $10^{-13} \sim 10^{-15}$" [18, p. 146]. Now if

$$\frac{|E_{\alpha,\beta}^{(k)}(z) - \widehat{E}_{\alpha,\beta}^{(k)}(z)|}{1+|E_{\alpha,\beta}^{(k)}(z)|} = \epsilon, \tag{6.1}$$

then the relative error

$$\phi = \frac{|E_{\alpha,\beta}^{(k)}(z) - \widehat{E}_{\alpha,\beta}^{(k)}(z)|}{|E_{\alpha,\beta}^{(k)}(z)|} = \frac{\epsilon}{|E_{\alpha,\beta}^{(k)}(z)|} + \epsilon,$$

so ϵ approximates the relative error for large function values $|E_{\alpha,\beta}^{(k)}(z)|$ and the absolute error when $|E_{\alpha,\beta}^{(k)}(z)|$ is small. However, in floating-point arithmetic it is preferred to use the relative error ϕ to quantify the quality of an approximation. Because they only satisfy (6.1), there can be large relative errors in the derivatives computed by the methods of [18] when $|E_{\alpha,\beta}^{(k)}(z)| \ll 1$. It is hard to identify the range of z, α, β, and k for which $|E_{\alpha,\beta}^{(k)}(z)| < 1$, but intuitively we expect that the kth order derivatives $|E_{\alpha,\beta}^{(k)}(z)|$ will generally decrease with decreasing $|z|$ or increasing β. Since the algorithm of [18] is so far the most practical algorithm for computing the matrix Mittag–Leffler function, we use it as a comparison in testing Algorithm 5.1.

In order to compute a matrix function by Algorithm 5.1 it is necessary to be able to accurately evaluate its corresponding scalar function. For the Mittag–Leffler function, the state-of-the-art algorithm `ml_opc` [14] for computing the scalar function aims to achieve

$$\frac{|E_{\alpha,\beta}(z) - \widehat{E}_{\alpha,\beta}(z)|}{1+|E_{\alpha,\beta}(z)|} \le 10^{-15}.$$

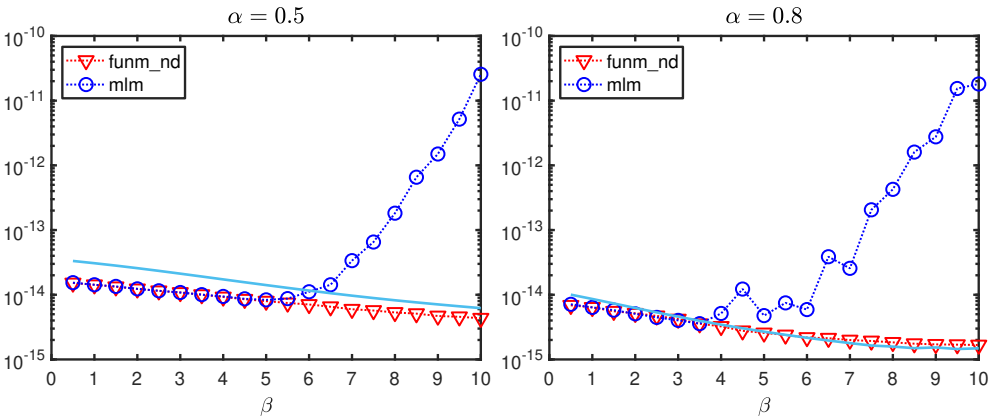

FIG. 6.1. *Normwise relative errors in the computed $E_{\alpha,\beta}(-R)$ for the Redheffer matrix and different α and β. The solid lines are $\kappa_{ML}(A)u$.*

Hence `ml_opc` can produce large relative errors when $|E_{\alpha,\beta}(z)| \ll 1$. By the power series definition we intuitively expect that the function value $|E_{\alpha,\beta}(z)|$ will generally decrease with decreasing $|z|$ or increasing β. Hence we do not expect `ml_opc` to provide small relative errors for all arguments.

6.1. Numerical experiments. In this section we present numerical tests of Algorithm 5.1 (`funm_nd`). In `funm_nd` the ability to accurately evaluate the scalar Mittag–Leffler function in precisions beyond the working precision is required. We evaluate the scalar Mittag–Leffler function by truncating the series definition and we use a precision a few digits more than the highest precision required by the algorithms for the evaluation of the triangular blocks.

In the literature particular attention has been paid to the Mittag–Leffler functions with $0 < \alpha < 1$ and $\beta > 0$ as this is the case that occurs most frequently in applications [16], [29]. In addition to the Mittag–Leffler functions with $\beta \approx 1$ that are often tested in the literature, we will also investigate the cases when β takes other positive values that appear in actual applications. For example, in linear multiterm fractional differential equations the source term can often be approximated by polynomials, and then the exact solution involves evaluating the matrix Mittag–Leffler function with $\beta = \alpha + \ell, \ell = 1, 2, \ldots$ [18].

We compare the accuracy of our algorithm `funm_nd` with that of `mlm`, the numerical scheme proposed by Garrappa and Popolizio [18]. The normwise relative forward error $\|\widehat{X} - E_{\alpha,\beta}(A)\|_F / \|E_{\alpha,\beta}(A)\|_F$ of the computed $\widehat{X}$ is reported, where the reference solution $E_{\alpha,\beta}(A)$ is computed by randomized approximate diagonalization at 200 digit precision. In the plots we also show $\kappa_{ML}(A)u$, where $\kappa_{ML}(A)$ is an estimate of the 1-norm condition number of the matrix Mittag–Leffler function.

Example 1*: the Redheffer matrix.* We first use the Redheffer matrix, which is `gallery('redheff')` in MATLAB and has been used for test purposes in [18]. It is a square matrix R with $r_{ij} = 1$ if i divides j or if $j = 1$ and otherwise $r_{ij} = 0$. The Redheffer matrix has $n - \lfloor \log_2 n \rfloor - 1$ eigenvalues equal to 1 [6], which makes it necessary to evaluate high order derivatives in computing $E_{\alpha,\beta}(-R)$ by means of the standard Schur–Parlett algorithm. The dimension of the matrix is set to $n = 20$.

In this case the Schur–Parlett algorithm `funm_nd` chooses five blocks: one 16×16

TABLE 6.1
Eigenvalues (with multiplicities/numbers) for the matrices in Example 2. Here, $[\ell, r](k)$ means that we take k eigenvalues from the uniform distribution on the interval $[\ell, r]$.

Matrix	Eigenvalues (multiplicities/numbers)	Size
A_{21}	$0(3), \quad \pm 1.0(6), \quad \pm 5(6), \quad -10(3)$	30×30
A_{22}	$\pm[0.9, 1.0](5), \quad \pm[1.2, 1.3](4), \quad \pm[1.4, 1.5](3), \quad \pm[0.9, 1.0] \pm 1i(4)$	40×40

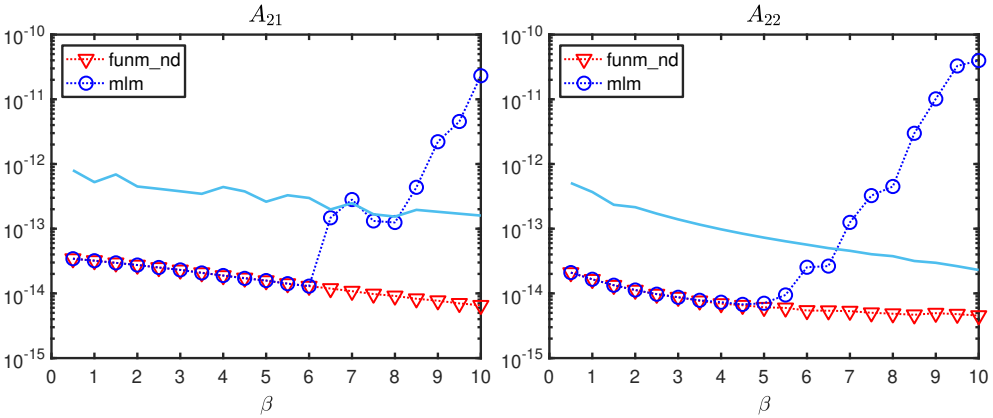

FIG. 6.2. *Forward errors in the computed $E_{\alpha,\beta}(A)$ for $\alpha = 0.8$ and different β for the matrices in Table 6.1. The solid lines are $\kappa_{ML}(A)u$.*

block and four 1×1 blocks to compute the matrix Mittag–Leffler functions. Figure 6.1 shows that the errors for `funm_nd` are all $O(10^{-14})$ and are below $\kappa_{ML}(A)u$ for all tested α and β, showing the forward stability of `funm_nd`. On the other hand, for $\beta \geq 6$, `mlm` produces errors that grow with β and become much larger than $\kappa_{ML}(A)u$, so it is behaving numerically unstably. It is not surprising to see that `mlm` becomes numerically unstable when $\beta = 8.0$, as it aims to achieve (6.1) and $|E_{\alpha,\beta}(z)|$ decays to 0 when β increases; for example, $|E_{0.5,10}(1)| \approx 4.0\text{e-}6$.

Example 2: matrices with clustered eigenvalues. In the second experiment we test two matrices A_1 and A_2 of size 30×30 and 40×40 with both fixed and randomly generated eigenvalues that are clustered to different degrees, as explained in Table 6.1.

The test matrices were designed to have nontrivial diagonal blocks in the reordered and blocked Schur form. We assigned the specified values to the diagonal matrices and performed similarity transformations with random matrices having a condition number of order the matrix size to obtain the full matrice A_{21} and A_{22} with the desired spectrum.

In this example, `funm_nd` chooses six blocks for A_{21} and ten blocks for A_{22}. Figure 6.2 shows that for these matrices `funm_nd` performs in a numerically stable fashion, whereas `mlm` does not for $\beta \geq 6$.

Example 3: matrices from the MATLAB `gallery`. Now we take 10×10 matrices from the MATLAB `gallery` and test the algorithm using the matrix Mittag–Leffler functions with $\alpha = 0.8$ and $\beta = 1.2$ or $\beta = 8.0$. The forward errors are shown in Figure 6.3. We see that `mlm` is mostly numerically unstable for $\beta = 8$ while `funm_nd` remains largely numerically stable.

One conclusion from these experiments is that by exploiting higher precision

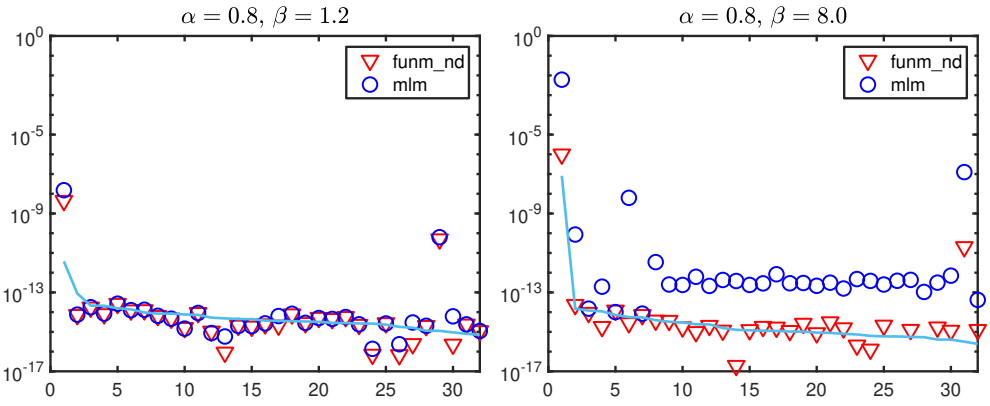

FIG. 6.3. *Forward errors in the computed $E_{\alpha,\beta}(A)$ for matrices A of size 10×10 from the MATLAB* `gallery`. *The solid lines are $\kappa_{ML}(A)u$.*

arithmetic it is possible to evaluate the Mittag–Leffler function with small relative error even when the function has small norm.

7. Conclusions. We have built an algorithm for evaluating arbitrary matrix functions $f(A)$ that requires only function values and not derivatives. The inspiration for our algorithm is Davies's randomized approximate diagonalization. We have shown that the measure of error that underlies randomized approximate diagonalization makes it unsuitable for computing matrix functions. Nevertheless, we have exploited the approximate diagonalization idea within the Schur–Parlett algorithm by making random diagonal perturbations to the nontrivial blocks of order greater than 2 in the reordered and blocked triangular Schur factor and then diagonalizing the perturbed blocks in higher precision. Our new multiprecision algorithm, Algorithm 5.1, is applicable to any sufficiently smooth function and requires only function values. By contrast, the standard Schur–Parlett algorithm, implemented as `funm` in MATLAB, requires derivatives and is applicable only to functions that have a Taylor series with a sufficiently large radius of convergence.

Numerical experiments show similar accuracy of our algorithm to `funm`. We found that when applied to the Mittag–Leffler function $E_{\alpha,\beta}$ our algorithm provides results of accuracy at least as good as, and systematically for $\beta \geq 6$ much greater than, the special-purpose algorithm `mlm` of [18].

Our multiprecision Schur–Parlett algorithm requires at most $2n^3/3$ flops to be carried out in higher precisions in addition to the approximately $28n^3$ flops at the working precision, and the amount of higher precision arithmetic needed depends on the eigenvalue distribution of the matrix. When there are only 1×1 and 2×2 blocks on the diagonal of the reordered and blocked triangular Schur factor no higher precision arithmetic is required.

Our new algorithm is a useful companion to `funm` that greatly expands the class of readily computable matrix functions. Our MATLAB code `funm_nd` is available from https://github.com/Xiaobo-Liu/mp-spalg.

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] Zhaojun Bai, James W. Demmel, and Alan McKenney. On computing condition numbers for the nonsymmetric eigenproblem. *ACM Trans. Math. Software*, 19(2):202–223, 1993.

[4] Jess Banks, Archit Kulkarni, Satyaki Mukherjee, and Nikhil Srivastava. Gaussian regularization of the pseudospectrum and Davies' conjecture. ArXiv:1906.11819, 2019.

[5] Jess Banks, Jorge G. Vargas, Archit Kulkarni, and Nikhil Srivastava. Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time. 2019. Revised April 2020.

[6] Wayne W. Barrett and Tyler J. Jarvis. Spectral properties of a matrix of Redheffer. *Linear Algebra Appl.*, 162-164:673–683, 1992.

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

[8] Philip I. Davies and Nicholas J. Higham. A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[9] James W. Demmel. The condition number of equivalence transformations that block diagonalize matrix pencils. *SIAM J. Numer. Anal.*, 20(3):599–610, 1983.

[10] Junsheng Duan and Lian Chen. Solution of fractional differential equation systems and computation of matrix Mittag-Leffler functions. *Symmetry*, 10(10):503, 2018.

[11] Massimiliano Fasi and Nicholas J. Higham. Multiprecision algorithms for computing the matrix logarithm. *SIAM J. Matrix Anal. Appl.*, 39(1):472–491, 2018.

[12] Massimiliano Fasi and Nicholas J. Higham. An arbitrary precision scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 40(4):1233–1256, 2019.

[13] Massimiliano Fasi, Nicholas J. Higham, and Bruno Iannazzo. An algorithm for the matrix Lambert W function. *SIAM J. Matrix Anal. Appl.*, 36(2):669–685, 2015.

[14] Roberto Garrappa. Numerical evaluation of two and three parameter Mittag-Leffler functions. *SIAM J. Numer. Anal.*, 53(3):1350–1369, 2015.

[15] Roberto Garrappa, Igor Moret, and Marina Popolizio. Solving the time-fractional Schrödinger equation by Krylov projection methods. *J. Comp. Phys.*, 293:115–134, 2015.

[16] Roberto Garrappa, Igor Moret, and Marina Popolizio. On the time-fractional Schrödinger equation: Theoretical analysis and numerical solution by matrix Mittag-Leffler functions. *Computers Math. Applic.*, 74(5):977–992, 2017.

[17] Roberto Garrappa and Marina Popolizio. On the use of matrix functions for fractional partial differential equations. *Math. Comput. Simulation*, C-25(81):1045–1056, 2011.

[18] Roberto Garrappa and Marina Popolizio. Computing the matrix Mittag-Leffler function with applications to fractional calculus. *J. Sci. Comput.*, 77(1):129–153, 2018.

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

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

[21] Nicholas J. Higham. Computing real square roots of a real matrix. *Linear Algebra Appl.*, 88/89: 405–430, 1987.

[22] 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.

[23] 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.

[24] Nicholas J. Higham. Short codes can be long on insight. *SIAM News*, 50(3):2–3, 2017.

[25] Nicholas J. Higham and Edvin Hopkins. A catalogue of software for matrix functions. Version 3.0. MIMS EPrint 2020.7, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, March 2020. 24 pp.

[26] Vishesh Jain, Ashwin Sah, and Mehtaab Sawhney. On the real Davies' conjecture. ArXiv:2005.08908v2, July 2020.

[27] Charles S. Kenney and Alan J. Laub. Condition estimates for matrix functions. *SIAM J. Matrix Anal. Appl.*, 10(2):191–209, 1989.

[28] Cleve B. Moler and Charles F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[29] Igor Moret and Paolo Novati. On the convergence of Krylov subspace methods for matrix Mittag–Leffler functions. *SIAM J. Numer. Anal.*, 49(5):2144–2164, 2011.
[30] Multiprecision Computing Toolbox. Advanpix, Tokyo. http://www.advanpix.com.
[31] Beresford N. Parlett. Computation of functions of triangular matrices. Memorandum ERL-M481, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, November 1974. 18 pp.
[32] Marina Popolizio. Numerical solution of multiterm fractional differential equations using the matrix Mittag–Leffler functions. *Mathematics*, 6(1):7, 2018.
[33] J. D. Roberts. Linear model reduction and solution of the algebraic Riccati equation by use of the sign function. *Internat. J. Control*, 32(4):677–687, 1980. First issued as report CUED/B-Control/TR13, Department of Engineering, University of Cambridge, 1971.
[34] Marianito R. Rodrigo. On fractional matrix exponentials and their explicit calculation. *J. Differential Equations*, 261(7):4223–4243, 2016.
[35] Angelika Schwarz, Carl Christian Kjelgaard Mikkelsen, and Lars Karlsson. Robust parallel eigenvector computation for the non-symmetric eigenvalue problem. Report UMINF 20.02, Department of Computing Science, University of Umeå, Sweden, 2020. 25 pp.