

***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<sup>†</sup> AND XIAOBO LIU<sup>†</sup>

**Abstract.** The Schur–Parlett algorithm, implemented in MATLAB as `fnumm`, evaluates an analytic function  $f$  at an  $n \times n$  matrix argument 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 not derivatives. The algorithm requires access to arithmetic of a matrix-dependent precision at least double the working precision, which is used 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 diagonal block, where the perturbation ensures that diagonalization will succeed. 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. This multiprecision Schur–Parlett algorithm is applicable to arbitrary analytic functions  $f$  and, like the original Schur–Parlett algorithm, it generally behaves in a numerically stable fashion. The algorithm is especially useful when the derivatives of  $f$  are not readily available or accurately computable. 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.

**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.** In this work we are concerned with functions mapping  $\mathbb{C}^{n \times n}$  to  $\mathbb{C}^{n \times n}$  that are defined in terms of an underlying scalar function  $f$  and a Cauchy integral formula, a Hermite interpolating polynomial, or the Jordan canonical form (see, for example, [26, Chap. 1] for details). We assume that  $f$  is analytic on a closed convex set whose interior contains the eigenvalues of the matrix argument. The need to compute such matrix functions arises in numerous applications in science and engineering. Specialized methods exist for evaluating particular matrix functions, including the scaling and squaring algorithm for the matrix exponential [1], [34], Newton’s method for matrix sign function [26, Chap. 5], [39], and the inverse scaling and squaring method for the matrix logarithm [2], [32]. See [28] for a survey and [29] 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 [11] 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 eigen-

---

\*Version dated June 6, 2021.

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

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

values. Our algorithm handles close or repeated eigenvalues by an idea inspired by Davies's idea of randomized approximate diagonalization [10] 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 implemented in software, and so will be expensive, but we aim to use them as little as possible.

We note that multiprecision algorithms have already been developed for the matrix exponential [15] and the matrix logarithm [14]. 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 [16]. 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 “standard Gaussian matrix” to mean a random matrix with entries independently drawn from a standard normal distribution (mean 0, variance 1). A diagonal or triangular standard Gaussian matrix has the entries on the diagonal or in the triangle drawn in the same way. 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  $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$ . We will also need the condition number  $\kappa(A) = \|A\|\|A^{-1}\|$ , and we indicate the norm with a subscript on  $\kappa(A)$ .

**2. Schur–Parlett algorithm.** The Schur–Parlett algorithm [11] 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. Let  $T = (T_{ij})$  be partitioned to be block upper triangular with square diagonal blocks, possibly of different sizes. Then  $F = f(T)$  has the same block structure and if the function of the square diagonal blocks  $F_{ii} = f(T_{ii})$  can be computed then the off-diagonal blocks  $F_{ij}$  can be obtained using the block form of Parlett's recurrence [37]. To be more specific, by equating  $(i, j)$  blocks in  $TF = FT$ , we obtain a triangular Sylvester equation for  $F_{ij}$ ,

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

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 [11] devised a scheme with a blocking parameter  $\delta > 0$  to reorder  $T$  into a partitioned upper triangular matrix  $\tilde{T} = U^*TU = (\tilde{T}_{ij})$  by a unitary similarity transformation such that

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

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

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

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  $\tilde{T}_{ii} - \sigma I$  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  $\lambda$  of  $\tilde{T}_{ii}$ . Obviously, this procedure for evaluating  $f(\tilde{T})$  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  $D = \text{diag}(d_i)$  is diagonal and  $V$  is nonsingular, so  $f(A) = Vf(D)V^{-1} = V \text{diag}(f(d_i))V^{-1}$  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 [26, sect. 4.5].

A way to handle a nonnormal matrix is to perturb it before diagonalizing it. Davies [10] suggested perturbing  $A$  to  $\tilde{A} = A + E$ , computing the diagonalization  $\tilde{A} = VDV^{-1}$ , and approximating  $f(A)$  by  $f(\tilde{A}) = Vf(D)V^{-1}$ . 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

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

where  $\epsilon$  measures the accuracy of the finite precision computations and so can be taken 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

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

for some constant  $c_n$ , where  $\|A\|_2 \leq 1$  is assumed, and he proved the conjecture for Jordan blocks and triangular Toeplitz matrices (both with  $c_n = 2$ ) and for arbitrary  $3 \times 3$  matrices (with  $c_3 = 4$ ). Davies's conjecture was recently proved by Banks, Kulkarni, Mukherjee, and Srivastava [7, Thm. 1.1] with  $c_n = 4n^{3/2} + 4n^{3/4} \leq 8n^{3/2}$ . Building on the solution of Davies' conjecture a randomized algorithm with low computational complexity is developed in [5], [6] for approximately computing the eigensystem. Note

TABLE 3.1

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

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

that (3.2) suggests it is sufficient to choose  $E$  such that  $\|E\|_2 \approx \epsilon^{1/2}$  in order to obtain  $\sigma(A, \epsilon)$  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 [7],  $E$  is not necessarily real when  $A$  is real. However, Jain, Sah, and Sawhney [31, Thm 1.1] have proved the conjecture for real  $A$  and real perturbations  $E$  up to an extra factor of  $(\log(1/\epsilon))^{1/4}$ , while a weaker bound for which the extra factor is  $\epsilon^{1/4}$  was independently derived by Banks et al. [8, Thm. 1.7G].

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 standard Gaussian 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  $\|A\|_2 \leq 1$ . One of us published a short MATLAB code to implement this idea [27],<sup>1</sup> as a way of computing  $f(A)$  with error of order  $u^{1/2}$ . However, this approach does not give 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

$$(3.3) \quad \|f(A + E) - f(A)\| \lesssim \|L_f(A, E)\| \leq \|L_f(A)\| \|E\|,$$

where  $L_f(A, E)$  is the Fréchet derivative of  $f$  at  $A$  in the direction  $E$  and  $\|L_f(A)\| = \max\{\|L_f(A, E)\| : \|E\| = 1\}$  [26, sect. 3.1]. Hence while  $\sigma$  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 this point. All the experiments in this paper are carried out in MATLAB R2020b 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  $\lambda$  and  $f(A) = A^{1/2}$  (the principal matrix square root), for which  $\|L_f(A)\|_F = \|(I \otimes A^{1/2} + (A^{1/2})^T \otimes I)^{-1}\|_2$  [24]. The diagonalization and evaluation of  $f(\tilde{A})$  is done at the working precision. In Table 3.1 we show the relative errors  $\|f(A) - f(\tilde{A})\|_F/\|f(A)\|_F$ , where  $E$  is a (full) standard Gaussian matrix scaled so that  $\|E\|_F = u^{1/2}\|A\|_F$  and the reference solution  $f(A)$  is computed in 200 digit precision using the function `sqrtm` from the Multiprecision Computing Toolbox [36]. For  $\lambda = 1$  we obtain an error of order  $u^{1/2}$ , but the errors grow as  $\lambda$  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  $\lambda$ . We stress that increasing the precision at which  $f(\tilde{A})$  is evaluated does not reduce the errors; the damage done by the perturbation  $E$  cannot be recovered.

<sup>1</sup><https://gist.github.com/higham/6c00f62e48c1b0116f2e9a8f43f2e02a>

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

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

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 reordered Schur form as the Schur–Parlett algorithm. The key difference from that algorithm is how it evaluates  $f$  on the (upper triangular) diagonal blocks. Given a diagonal block  $T \in \mathbb{C}^{m \times m}$  of the blocked and 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 [11, 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 standard Gaussian 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

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

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 standard Gaussian matrix, normalized by (4.1). An obvious advantage of taking  $E$  triangular is that  $\tilde{T} = T + E$  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  $\tilde{T}$  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

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

so when  $t_{11} = t_{22}$ , perturbing to  $\tilde{t}_{11} \neq \tilde{t}_{22}$  results in a first order finite difference approximation to  $f'(t_{11})$ . If  $m = 2$  we can simply use (4.2) at the working precision, so the rest of this section is aimed at the case  $m \geq 3$ .

In order to find the eigenvector matrix  $V$  of the perturbed triangular matrix  $\tilde{T} = T + E$  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

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

where we set  $v_i$  to be 1 in its  $i$ th component, zero in components  $i+1:m$ , and solve for the first  $i-1$  components by substitution. Thus the matrix  $V$  is upper triangular.

To summarize, we compute in precision  $u_h$  the diagonalization

$$(4.4) \quad \tilde{T} = VDV^{-1}, \quad D = \text{diag}(\lambda_i),$$

where in practice the  $\lambda_i$  will be distinct. We then form  $f(\tilde{T}) = Vf(D)V^{-1}$  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  $\sum_{k=1}^m k^2 + m^3/3 = 2m^3/3 + O(m^2)$  flops in precision  $u_h$ .

**4.2.1. Determining the precision.** Now we determine the precision at which to carry out the diagonalization.

We expect the error in the computed approximation  $\hat{F}$  to  $F = f(\tilde{T})$  to be bounded approximately by (cf. [26, p. 82])

$$(4.5) \quad \frac{\|F - \hat{F}\|_1}{\|F\|_1} \lesssim \kappa_1(V) \frac{\|f(D)\|_1}{\|f(\tilde{T})\|_1} u_h.$$

(The choice of norm is not crucial; the 1-norm is convenient here.) We will use this bound to determine  $u_h$ . We do not know  $\|f(\tilde{T})\|_1$  a priori, but we have the bound

$$(4.6) \quad \frac{\|f(D)\|_1}{\|f(\tilde{T})\|_1} \leq 1,$$

which follows from the fact that the spectral radius of a matrix is bounded above by any norm. Hence we can certainly expect that

$$(4.7) \quad \frac{\|F - \hat{F}\|_1}{\|F\|_1} \lesssim \kappa_1(V) u_h.$$

Since we need to know how to choose  $u_h$  before we compute  $V$ , we need an estimate of  $\kappa(V)$  based only on  $\tilde{T}$ . 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 [4, sect. 5.3], [12] 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  $\lambda_i$ . Writing

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

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

$$(4.8) \quad P_1 = \begin{bmatrix} 1 & p^* \\ 0 & 0 \end{bmatrix}, \quad p^* = \tilde{t}_{12}^*(\tilde{t}_{11}I - \tilde{T}_{22})^{-1}.$$

From (4.8) we have

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

Now for any  $m \times m$  upper triangular matrix  $U$  we have the bound [25, Thm. 8.12, Prob. 8.5]

$$(4.9) \quad \|U^{-1}\|_1 \leq \frac{1}{\alpha} \left( \frac{\beta}{\alpha} + 1 \right)^{m-1}, \quad \alpha = \min_i |u_{ii}|, \quad \beta = \max_{i < j} |u_{ij}|.$$

This bound will be very pessimistic if we apply it to  $\tilde{t}_{11}I - \tilde{T}_{22}$ , because for the bound to be a good approximation it is necessary that many diagonal elements of  $U$  are of order  $\alpha$ , yet  $\tilde{t}_{11}I - \tilde{T}_{22}$  will typically have only a few (if any) small elements. Let us group the  $\tilde{t}_{ii}$  according to the Schur–Parlett blocking criteria described in section 2, with blocking parameter  $\delta = \delta_1$ . Suppose the largest block has size  $k \geq 2$  and suppose, without loss of generality, that it comprises the first  $k$  diagonal elements of  $\tilde{T}$ . Then we will approximate  $\|(\tilde{t}_{11}I - \tilde{T}_{22})^{-1}\|_1$  by  $\|(\tilde{t}_{11}I - \tilde{T}_{22}(1:k-1, 1:k-1))^{-1}\|_1$ , and bound it by (4.9), leading to the approximation

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

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  $\tilde{t}_{11}I - \tilde{T}_{22}(1:k-1, 1:k-1)$ , 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 (4.7) we need  $\kappa_1(V)u_h \lesssim u$ , which gives the requirement

$$(4.10) \quad u_h \lesssim \frac{c_m u^2}{\max_{i < j} |\tilde{t}_{ij}| \left( \frac{\max_{i < j} |\tilde{t}_{ij}|}{c_m u} + 1 \right)^{k-2}}, \quad k \geq 2.$$

In the case  $k = 2$ , the bound (4.10) is  $u_h \lesssim c_m u^2 / \max_{i < j} |\tilde{t}_{ij}| = O(u^2)$ . 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.

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

$$(4.11) \quad u_h = \begin{cases} u^2, & k = 1, \\ \min(u^2, \text{right-hand side of (4.10)}), & k \geq 2. \end{cases}$$

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  $T$ .

**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  $u_h \leq u^2$ . Lines 11–13 are to be executed at precision  $u^2$  and lines 14–18 are to be executed at precision  $u_h$ , as indicated to the right of the line numbers.

---

```

1      if  $m = 1$ 
2           $f_{11} = f(t_{11})$ , quit
3      end
4      if  $m = 2$  and  $t_{11} \neq t_{22}$ 
5           $f_{11} = f(t_{11})$ ,  $f_{22} = f(t_{22})$ 
6           $f_{12} = t_{12}(f_{22} - f_{11})/(t_{22} - t_{11})$ 
7          quit
8      end
9      Form an  $m \times m$  diagonal or upper triangular standard Gaussian matrix  $N$ .
10      $E = u(\max_{i,j} |t_{ij}|/\|N\|_F)N$ 
11      $u^2 \quad \tilde{T} = T + E$ 
12      $u^2 \quad D = \text{diag}(\tilde{T})$ 
13      $u^2 \quad$  Evaluate  $u_h$  by (4.11).
14      $u_h \quad$  if  $u_h < u^2$ , convert  $\tilde{T}$  and  $D$  to precision  $u_h$ , end
15     for  $i = 1:m$ 
16      $u_h \quad$  Set  $(v_i)_i = 1$  and  $(v_i)_k = 0$  for  $k > i$  and solve the triangular system
            $(\tilde{T} - \tilde{t}_{ii}I)v_i = 0$  for the first  $i - 1$  components of  $v_i$ .
17   end
18    $u_h \quad$  Form  $F = Vf(D)V^{-1}$ , where  $V = [v_1, \dots, v_m]$ .
19   Round  $F$  to precision  $u$ .
```

---

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.

In practice, overflow could occur when solving the triangular systems (4.3), especially when there is a large cluster of eigenvalues. However, we solve (4.3) in high precision arithmetic, and these arithmetics have a very large exponent range, which reduces the likelihood of overflow. For example, IEEE quadruple precision arithmetic has largest element of order  $10^{4932}$  [30]. In any case, scaling techniques are available that avoid overflow [3], [42] (and likewise for the Sylvester equations (2.1) in the Schur–Parlett algorithm [41]).

**4.2.3. Specifying the parameters.** We now discuss the choice of the parameters  $\delta_1$  and  $c_m$  in Algorithm 4.1. The parameter  $c_m$  is such that  $c_m u \approx \min_i |w_{ii}|$ , where the  $w_{ii}$  are the diagonal elements of  $\tilde{t}_{11}I - \tilde{T}_{22}(1:k-1, 1:k-1)$ . We will determine  $c_m$  by considering the extreme case when  $\min_i |w_{ii}|$  is extremely small, which is when all  $t_{ii}$  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  $\tilde{t}_{11}I - \tilde{T}_{22}(1:k-1, 1:k-1)$  is extremely ill-conditioned. This choice of  $c_m$  makes the chosen higher precision  $u_h < u^2$  pessimistic when some of the  $\tilde{t}_{ii}$  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 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  $|w_{ii}| = |\tilde{t}_{11} - \tilde{t}_{ii}| = |e_{11} - e_{ii}|$ . The matrix  $E$  on line 10 of Algorithm 4.1 has entries  $u(\max_{i,j} |t_{ij}|) |\tilde{n}_{ij}|$ , where  $\|\tilde{N}\|_F = 1$ , and we expect that  $|\tilde{n}_{ij}| \approx 1/\sqrt{m}$  if  $N$  is chosen to be diagonal, or  $|\tilde{n}_{ij}| \approx \sqrt{2}/m$  if  $N$  is triangular [33]. This suggests taking

$$c_m = \begin{cases} \theta_d \max_{i,j} |t_{ij}|/\sqrt{m}, & \text{if } N \text{ is diagonal,} \\ \theta_t \max_{i,j} |t_{ij}|/m, & \text{if } N \text{ is triangular,} \end{cases}$$

for some constants  $\theta_d$  and  $\theta_t$ . The constants are introduced to empirically quantify the uncertainty caused by the stochasticity of the left-hand side in the approximation  $\min_i |e_{11} - e_{ii}| \approx c_m u$ . In our experiments with different choices of  $\theta_d$  and  $\theta_t$  we found  $\theta_d = 0.4$  and  $\theta_t = 0.5$  to be good choices.

The blocking parameter  $\delta = \delta_1$  is important in determining the largest group size  $k$  in (4.10). A smaller  $\delta$  can potentially group fewer eigenvalues and decrease  $k$ , causing a larger  $u_h$  to be used. Yet too large a  $\delta$  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, which worked well for the experiments presented in this work. In general there is no optimal (fixed) value for  $\delta_1$  as this blocking parameter controls the size  $k$  in the approximation  $\|(\tilde{t}_{11}I - \tilde{T}_{22})^{-1}\|_1 \approx \|(\tilde{t}_{11}I - \tilde{T}_{22}(1:k-1, 1:k-1))^{-1}\|_1$  used in section 4.2.1 for determining the precision  $u_h$ , and a suitable value for  $k$  that results in a good approximation is clearly problem-dependent. In general a larger  $\delta_1$  is more likely to produce a larger  $k$  and thus result in a higher precision to be used, so an overestimated  $\delta_1$  does no harm to accuracy but may impair the efficiency.

**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 [36].

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  $E$ ,  $c_m = 0.4 \max_{i,j} |t_{ij}|/\sqrt{m}$ , and  $\delta_1 = 5 \times 10^{-3}$ .

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

- $T_1 = \text{gallery('kahan', m)}$ : upper triangular with distinct diagonal elements on the interval  $(0, 1]$ .
- $T_2 = \text{schur}(\text{gallery('smoke', m)}, 'complex')$ : Schur factor of the complex matrix whose eigenvalues are the  $m$ th roots of unity times  $2^{1/m}$ .
- $T_3 = \text{schur}(\text{randn}(m), 'complex')$ .
- $T_4 = \text{schur}(\text{rand}(m), 'complex')$ .
- $T_5 = \text{triu}(\text{randn}(m))$ .
- $T_6 = \text{triu}(\text{rand}(m))$ .
- $T_7 = \text{gallery('jordbloc', m, 0.5)}$ : 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  $1 + i$  for these functions to avoid their eigenvalues being on the negative real axis, where  $i$  is the imaginary unit.

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  $u_h = u^2$ .*

	$m = 35$	$m = 75$
$T_1 = \text{gallery}('kahan', m)$	32	623
$T_2 = \text{schur}(\text{gallery}('smoke', m), 'complex')$	32	32
$T_3 = \text{schur}(\text{randn}(m), 'complex')$	32	32
$T_4 = \text{schur}(\text{rand}(m), 'complex')$	32	32
$T_5 = \text{triu}(\text{randn}(m))$	34	68
$T_6 = \text{triu}(\text{rand}(m))$	51	68
$T_7 = \text{gallery}('jordbloc', m, 0.5)$	599	1296

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  $\|F - \hat{F}\|_F / \|F\|_F$ , 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. For the reference solution of the matrix sign function, we run `signm` from the Matrix Function Toolbox [23] at 200 digit precision, and round back to double precision. The same procedure is followed in the experiments in the following sections.

We also show in Table 4.2 the quantity  $\kappa_f(A)u$ , where  $\kappa_f(A)$  is the 1-norm condition number [26, Chap. 3] of  $f$  at  $A$ , which we estimate using the `funm_condest1` function provided by [23]. 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 (4.10)) 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. The failure of the method occurs for the square root, sign, and logarithm functions, all of which have singularities.

Note that our test matrices here are more general than will arise in the algorithm of the next section, for which the diagonal 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 [11]. First the Schur decomposition  $A = QTQ^*$  is computed. Then the triangular matrix  $T$  is reordered to a partitioned upper triangular matrix  $\tilde{T}$  by a unitary similarity transformation, which is achieved by Algorithms 4.1 and 4.2 in [11, sect. 4]. The function is evaluated on the diagonal blocks  $\tilde{T}_{ii}$  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  $f(\tilde{T})$  are computed using the block form (2.1) of the Parlett recurrence. Finally, we undo the unitary simi-

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`). Exceptionally large errors are highlighted in bold red text.

$f = \exp$						$f = \text{sqrt}$					
	<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$
$T_1, m = 35$	5.6e-17	3.3e-17	6.4e-15		2.7e-16	3.1e-13	5.4e-11				
$T_1, m = 75$	5.4e-17	2.5e-17	1.2e-14		2.1e-15	<b>8.2e-7</b>	3.2e-11				
$T_2, m = 35$	5.2e-17	4.2e-17	1.9e-15		5.9e-16	3.9e-13	5.6e-11				
$T_2, m = 75$	3.0e-17	2.2e-17	3.6e-15		5.5e-16	<b>1.0e-7</b>	4.5e-12				
$T_3, m = 35$	1.7e-16	9.2e-17	1.9e-14		1.4e-16	1.4e-16	3.3e-14				
$T_3, m = 75$	1.1e-16	6.2e-17	1.1e-13		1.8e-16	1.3e-16	1.5e-13				
$T_4, m = 35$	6.6e-16	2.2e-16	3.2e-15		1.4e-15	1.3e-15	4.9e-14				
$T_4, m = 75$	1.1e-15	2.1e-17	2.6e-14		1.4e-15	1.7e-15	1.5e-13				
$T_5, m = 35$	8.9e-17	7.7e-17	1.0e-14		1.5e-15	<b>9.7e-9</b>	3.8e-10				
$T_5, m = 75$	5.5e-17	6.7e-17	5.6e-14		2.5e-14	<b>1.0</b>	4.3e-22				
$T_6, m = 35$	4.6e-17	4.8e-17	3.7e-14		1.0e-15	6.7e-12	5.6e-9				
$T_6, m = 75$	2.8e-17	5.4e-17	2.6e-13		1.9e-15	<b>1.0</b>	2.7e-14				
$T_7, m = 35$	5.8e-17	1.8e-16	5.7e-15		4.1e-16	<b>8.5e-8</b>	3.9e-12				
$T_7, m = 75$	1.1e-19	3.6e-16	1.2e-14		3.4e-16	<b>1.0</b>	6.6e-18				
$f = \text{sign}$						$f = \log$					
	<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$
$T_1, m = 35$	0	1.6e-25	7.7e-22		2.3e-16	5.1e-13	7.9e-11				
$T_1, m = 75$	0	5.2e-20	1.3e2		4.1e-15	<b>1.1e-6</b>	3.1e-12				
$T_2, m = 35$	3.6e-16	3.8e-13	4.2e-11		5.1e-16	6.3e-13	5.7e-11				
$T_2, m = 75$	5.1e-16	<b>1.2e-7</b>	5.6e-12		8.8e-16	<b>1.7e-7</b>	2.5e-12				
$T_3, m = 35$	1.6e-16	1.4e-16	4.1e-14		2.0e-16	1.9e-16	3.2e-14				
$T_3, m = 75$	8.6e-17	9.1e-17	4.2e-14		1.9e-16	1.8e-16	1.2e-13				
$T_4, m = 35$	9.4e-16	9.2e-16	3.7e-14		1.3e-15	2.2e-15	6.1e-14				
$T_4, m = 75$	1.2e-15	2.2e-15	9.8e-14		1.5e-15	3.1e-15	1.5e-13				
$T_5, m = 35$	1.8e-15	<b>7.2e-9</b>	8.9e-11		1.6e-15	<b>1.1e-8</b>	1.0e-10				
$T_5, m = 75$	1.8e-14	<b>1.0</b>	5.8e-23		3.6e-14	<b>1.0</b>	6.0e-23				
$T_6, m = 35$	0	3.0e-17	4.1e-21		1.9e-15	8.8e-12	2.1e-9				
$T_6, m = 75$	0	<b>1.0</b>	3.8e3		3.0e-15	<b>1.0</b>	3.4e-15				
$T_7, m = 35$	0	1.1e-16	2.3e1		2.3e-16	<b>1.4e-7</b>	7.2e-13				
$T_7, m = 75$	0	<b>1.0</b>	1.4e2		7.1e-16	<b>1.0</b>	1.0e-18				
$f = \cos$						$f = \sin$					
	<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$		<code>Alg_diag</code>	<code>Alg_full</code>	$\kappa_f(A)u$
$T_1, m = 35$	3.8e-17	3.5e-17	7.9e-15		4.6e-17	4.7e-17	8.3e-15				
$T_1, m = 75$	3.2e-17	2.6e-17	1.7e-14		3.6e-17	4.4e-17	3.7e-14				
$T_2, m = 35$	5.4e-17	4.4e-17	3.5e-15		4.1e-17	3.4e-17	4.3e-15				
$T_2, m = 75$	4.1e-17	3.0e-17	7.1e-15		3.8e-17	1.7e-17	8.3e-15				
$T_3, m = 35$	1.8e-16	8.6e-17	1.6e-14		1.4e-16	<b>9.7e-7</b>	1.6e-14				
$T_3, m = 75$	1.3e-16	6.8e-17	6.6e-14		1.7e-16	6.6e-17	5.6e-14				
$T_4, m = 35$	3.4e-16	2.5e-16	8.7e-15		3.3e-16	2.4e-16	8.1e-15				
$T_4, m = 75$	4.8e-16	3.0e-16	2.1e-14		4.5e-16	2.9e-16	1.9e-14				
$T_5, m = 35$	7.0e-17	7.5e-17	1.3e-14		7.0e-17	7.4e-17	1.5e-14				
$T_5, m = 75$	5.1e-17	6.1e-17	5.2e-9		4.9e-17	5.9e-17	4.0e-9				
$T_6, m = 35$	3.8e-17	4.1e-17	2.1e-11		3.8e-17	4.8e-17	8.4e-11				
$T_6, m = 75$	4.0e-17	7.2e-17	1.1e-13		2.2e-17	3.5e-17	1.1e-13				
$T_7, m = 35$	4.0e-17	4.8e-16	3.0e-15		3.7e-17	4.7e-16	2.9e-15				
$T_7, m = 75$	2.0e-17	9.3e-16	6.5e-15		1.5e-17	1.4e-15	6.3e-15				

larity 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 decom-

position 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  $u_h \leq u^2$  (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 [11, sect. 4] with  $\delta > 0$  to reorder  $T$  into
   a block  $m \times m$  upper triangular matrix  $\tilde{T} = U^*TU$ .
4 for  $i = 1:m$ 
5   Use Algorithm 4.1 (with a diagonal  $E$ ) to evaluate  $F_{ii} = f(\tilde{T}_{ii})$ .
6   for  $j = i-1:-1:1$ 
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  $\delta$  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  $\delta$  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  $\delta$  without consideration of the accuracy of the evaluation of  $f$  on the diagonal blocks and larger  $\delta$  will in general do no harm to accuracy. In the extreme case where  $\delta$  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  $\delta$  experimentally in the next subsection.

**5.1. Numerical experiments.** In the Schur–Parlett algorithm [11] 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  $\delta$  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_infinity`, 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 [22];
- other MATLAB matrices: `magic`, `rand`, and `randn`.

We set their size to be  $32 \times 32$ , 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  $\delta$  is fairly stable under scaling while using a  $\delta$  that scales with the matrix  $A$  (`funm_nd_norm`) can produce large errors when  $\|A\|$  is small. This is not unexpected since a smaller  $\delta$  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  $\delta$  in Algorithm 5.1 must be a balance between speed and accuracy, and the optimal choice of  $\delta$  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 [11] with  $\delta = 0.1$ ;
- `funm_nd`, Algorithm 5.1 with  $\delta = 0.1$ .
- `funm_nd_infinity`, 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 MATLAB 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_infinity` 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. 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 [11]. 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_infinity` and it can be substantially cheaper; indeed `funm_nd` requires no higher than the working precision to evaluate the function on the  $1 \times 1$  and  $2 \times 2$  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 normwise relative errors of `funm`, `funm_nd`, and `funm_nd_infinity` 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.

- $A_1 = \text{rand}(n)/5$ .
- $A_2 = \text{randn}(n)/10$ .

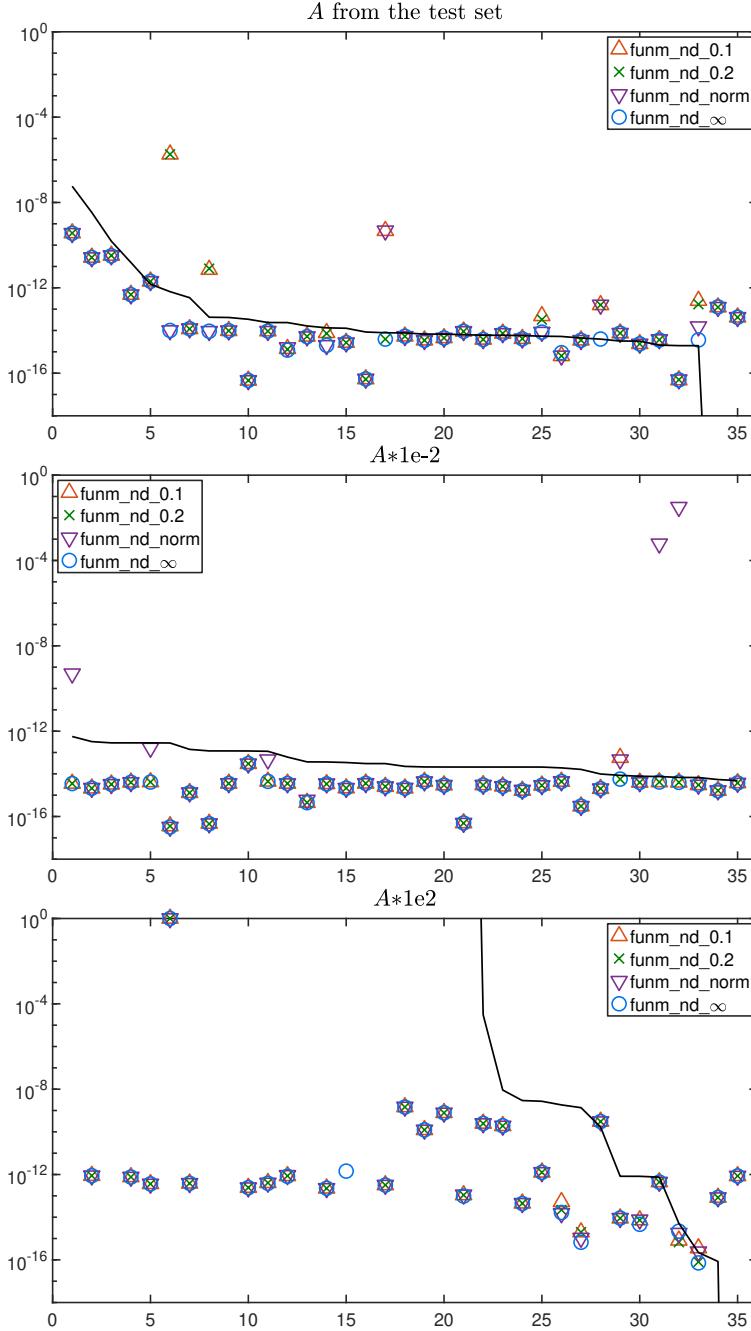


FIG. 5.1. Normwise relative errors for `funm_nd_0.1`, `funm_nd_0.2`, `funm_nd_norm` and `funm_nd_infinity` on the test set of 35 matrices, for the matrix sine. The solid line is  $\kappa_{\sin}(A)u$ .

- $A_3 = \text{gallery}(\text{'triw'}, n, -5)$ : 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_infinity` provide the same level of accuracy except for one case:  $f = \sin$  and  $A_3$ . In this case `funm` requires about  $n$

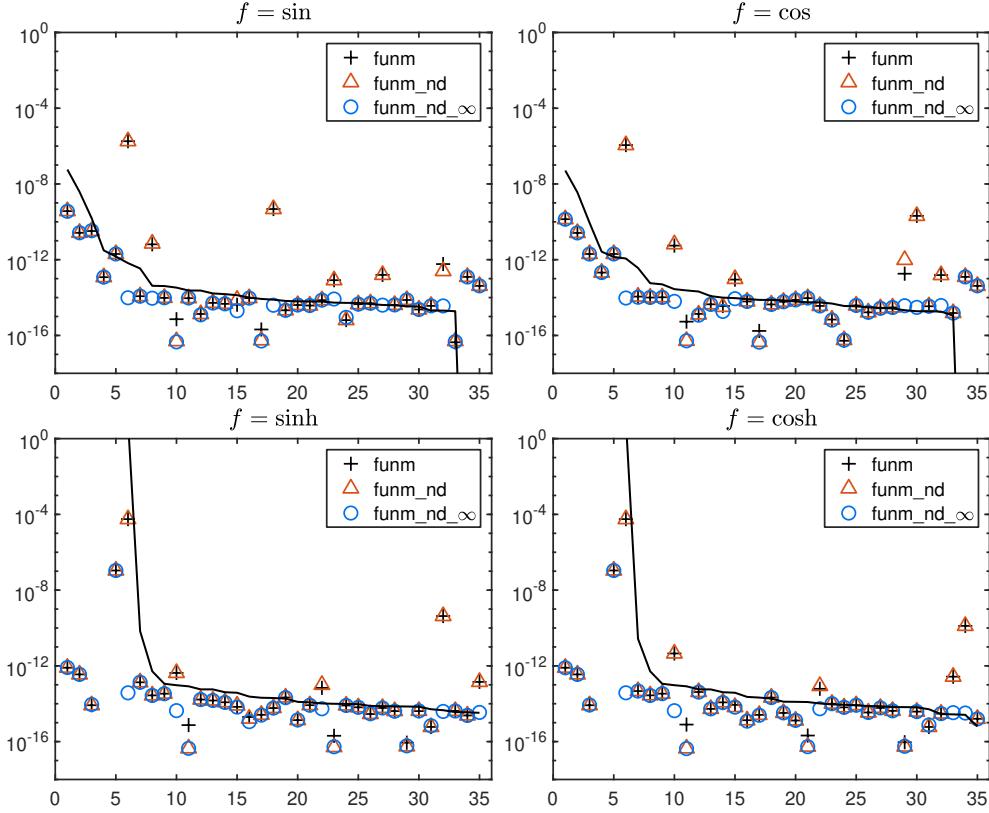


FIG. 5.2. Normwise relative errors for **funm**, **funm\_nd** $_{\infty}$  and **funm\_nd** on the test set of 35 matrices. The solid line is  $\kappa_f(A)u$ .

TABLE 5.1

Asymptotic costs in flops of **funm**, **funm\_nd**, and **funm\_nd** $_{\infty}$ . Here,  $n = \sum_{i=1}^s m_i$  is the size of the original matrix  $A$ ,  $s$  is the number of diagonal blocks in the Schur form after reordering and blocking, and  $m_i$  is the size of the  $i$ th block.

Precision	<b>funm</b>		<b>funm_nd</b>		<b>funm_nd</b> $_{\infty}$	
	$u$	$u_h$	$u$	$u_h$	$u$	$u_h$
Flops	$28n^3$ to $n^4/3$		$28n^3$	$2/3 \sum_{i=1}^s m_i^3$	$28n^3$	$2n^3/3$

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** $_{\infty}$ , which confirms that the cost of the reordering and blocking in **funm\_nd** is negligible. For the randomly generated matrices ( $A_1$  and  $A_2$ ) **funm** can be up to about 2.4 times faster than **funm\_nd** ( $f = \sin$  and  $A_1$  with  $n = 100$ ), 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** $_{\infty}$ .

Finally, we note that Algorithm 5.1 is not restricted only to a working precision

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`.*

$f = \sin$	Maximal relative error			Mean execution time (secs)			size	digits
	<code>funm</code>	<code>funm_nd</code>	<code>funm_nd_∞</code>	<code>funm</code>	<code>funm_nd</code>	<code>funm_nd_∞</code>		
$A_1, n = 40$	4.6e-15	4.6e-15	4.6e-15	2.1e-2	4.5e-2	1.4e-1	8	32
$A_2, n = 40$	4.0e-15	4.0e-15	3.9e-15	2.2e-2	2.4e-2	1.4e-1	3	32
$A_3, n = 40$	1.5e-14	7.1e-17	6.8e-17	1.8e-3	4.1e-2	4.1e-2	40	685
$A_1, n = 100$	6.7e-15	6.7e-15	6.7e-15	6.6e-2	1.6e-1	9.7e-1	13	32
$A_2, n = 100$	6.3e-15	6.4e-15	6.3e-15	1.9e-1	1.9e-1	1.0	4	32
$A_3, n = 100$	1.0e-12	5.8e-17	5.8e-17	2.7e-2	7.3e-1	7.4e-1	100	1734
$f = \cosh$	<code>funm</code>	<code>funm_nd</code>	<code>funm_nd_∞</code>	<code>funm</code>	<code>funm_nd</code>	<code>funm_nd_∞</code>	size	digits
	7.1e-15	7.1e-15	7.1e-15	2.1e-2	4.6e-2	1.4e-1		
$A_1, n = 40$	3.0e-15	2.9e-15	2.9e-15	2.2e-2	2.3e-2	1.4e-1	3	32
$A_2, n = 40$	8.3e-16	9.0e-17	8.0e-17	2.0e-3	4.3e-2	4.3e-2	40	685
$A_3, n = 40$	1.4e-14	1.4e-14	1.4e-14	6.8e-2	1.6e-1	9.9e-1	13	32
$A_1, n = 100$	5.9e-15	5.9e-15	5.9e-15	2.0e-1	2.0e-1	1.0	4	32
$A_2, n = 100$	8.0e-16	5.7e-17	5.8e-17	2.9e-2	7.6e-1	7.6e-1	100	1734

of double since its framework is precision independent. For other working precisions suitable values for the parameters  $c_m$ ,  $\delta_1$  and  $\delta$  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.** For  $A \in \mathbb{C}^{n \times n}$ , the matrix Mittag–Leffler with two parameters  $\alpha, \beta \in \mathbb{C}$ ,  $\operatorname{Re}(\alpha) > 0$ , is defined by the convergent series

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

where  $\Gamma(\gamma)$ ,  $\gamma \in \mathbb{C} \setminus \{0, -1, -2, \dots\}$  is the Euler gamma function. 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 [20], [40], including time-fractional Schrödinger equations [18], [19] and multiterm fractional differential equations [38]. Despite the importance of the matrix Mittag–Leffler function, little work has been devoted to its numerical computation. In [35], the computation of the action of matrix Mittag–Leffler functions based on Krylov methods is analysed. The Jordan canonical form and minimal polynomial or characteristic polynomial are considered in [13] for computing the matrix Mittag–Leffler function, but this approach is unstable in floating-point arithmetic.

The work by Garrappa and Popolizio [21] employs the Schur–Parlett algorithm to compute the matrix Mittag–Leffler function. For  $z \in \mathbb{C}$ , the derivatives of the scalar Mittag–Leffler function are given by

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

where  $(j)_k = j(j-1) \cdots (j-k+1)$  is the falling factorial, 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 [21, 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 [21] 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  $10^{-13} \sim 10^{-15}$ ” [21, p. 146]. Now if

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

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  $\epsilon$  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  $\phi$  to quantify the quality of an approximation. Because they only satisfy (6.1), derivatives computed by the methods of [21] can have large relative errors when  $|E_{\alpha,\beta}^{(k)}(z)| \ll 1$ . It is hard to identify the range of  $z$ ,  $\alpha$ ,  $\beta$ , and  $k$  for which  $|E_{\alpha,\beta}^{(k)}(z)| < 1$ , but intuitively we expect that the  $k$ th order derivatives  $|E_{\alpha,\beta}^{(k)}(z)|$  will generally decrease with decreasing  $|z|$  or increasing  $\beta$ . Since the algorithm of [21] 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 `m1 opc` proposed by Garrappa [17] for computing the scalar function aims to achieve

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

Hence, in view of the discussion above `m1 opc` can produce large relative errors when  $|E_{\alpha,\beta}(z)| \ll 1$  and we do not expect it 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 diagonal blocks.

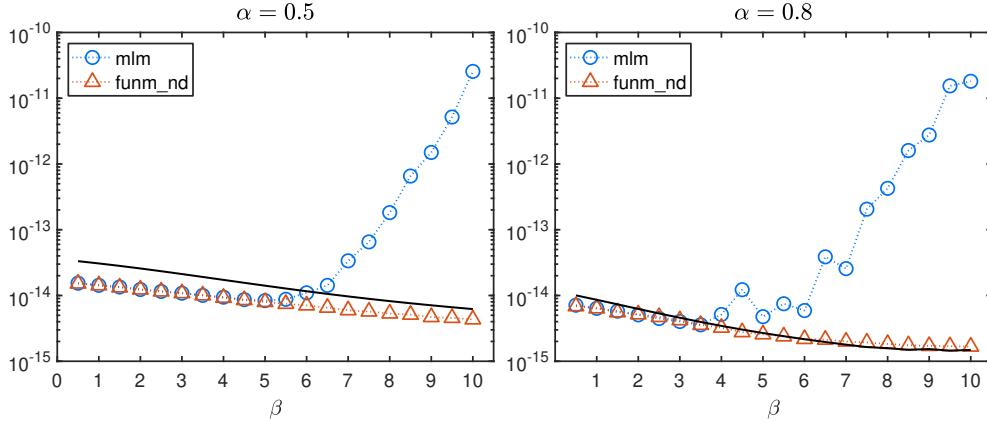


FIG. 6.1. Normwise relative errors in the computed  $E_{\alpha,\beta}(-R)$  for the Redheffer matrix and different  $\alpha$  and  $\beta$ . The solid lines are  $\kappa_{ML}(A)u$ .

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 [19], [35]. 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  $\beta$  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 a polynomial, say,  $p(t) = \sum_{i=0}^s c_i t^i$ , and then the solution involves evaluating the matrix Mittag–Leffler functions with  $\beta = \alpha + \ell$  for  $\ell = 1, 2, \dots, s+1$  [21].

We compare the accuracy of our algorithm **funm\_nd** with that of **mlm**, the numerical scheme proposed by Garrappa and Popolizio [21]. The normwise relative error  $\|\hat{X} - E_{\alpha,\beta}(A)\|_F / \|E_{\alpha,\beta}(A)\|_F$  of the computed  $\hat{X}$  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 [21]. 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 [9], 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  $16 \times 16$  block and four  $1 \times 1$  blocks to compute the matrix Mittag–Leffler functions with  $\alpha = 0.5$  or  $\alpha = 0.8$  and  $\beta$  starting from 0.5 to 10 with increment 0.5. 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  $\alpha$  and  $\beta$ , showing the forward stability of **funm\_nd**. On the other hand, for  $\beta \geq 6.5$ , **mlm** produces errors that in general grow with  $\beta$  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  $\beta = 10$ , as it aims to achieve (6.1) and  $|E_{\alpha,\beta}(z)|$  decays to 0 when  $\beta$  increases; for example,  $|E_{0.5,10}(1)| \approx 4.0 \times 10^{-6}$ .

*Example 2: matrices with clustered eigenvalues.* In the second experiment we test two matrices  $A_1$  and  $A_2$  of size  $30 \times 30$  and  $40 \times 40$  with both fixed and randomly

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
$A_{21}$	$0(3), \pm 1.0(6), \pm 5(6), -10(3)$	$30 \times 30$
$A_{22}$	$\pm[0.9, 1.0](5), \pm[1.2, 1.3](4), \pm[1.4, 1.5](3), \pm[0.9, 1.0] \pm 1i(4)$	$40 \times 40$

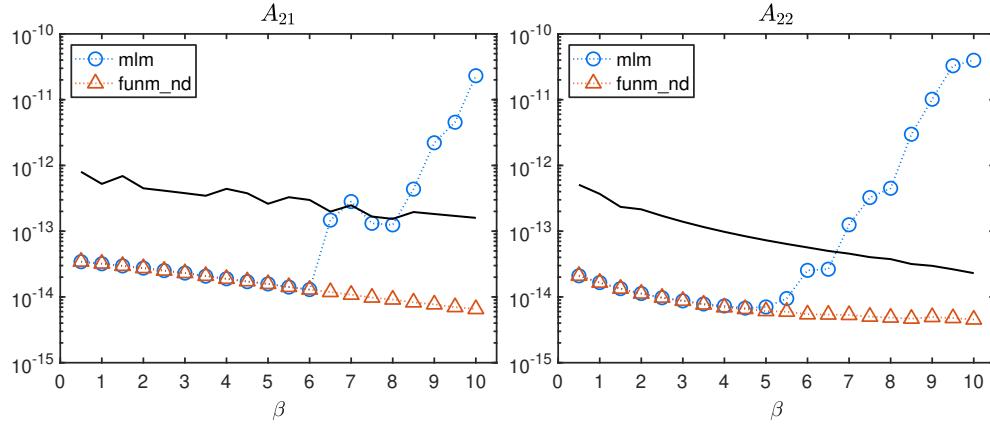


FIG. 6.2. Normwise relative errors in the computed  $E_{\alpha,\beta}(A)$  for  $\alpha = 0.8$  and different  $\beta$  for the matrices in Table 6.1. The solid lines are  $\kappa_{ML}(A)u$ .

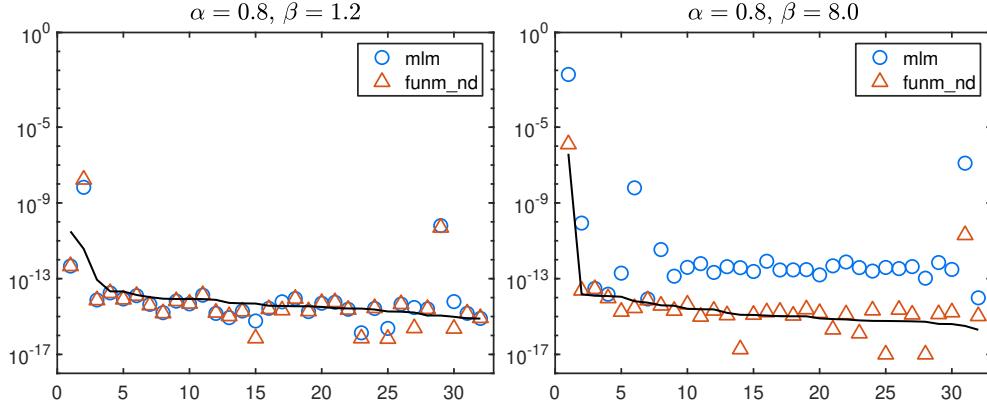


FIG. 6.3. Normwise relative errors in the computed  $E_{\alpha,\beta}(A)$  for  $A$  of size  $10 \times 10$  from the set of 32 matrices from the MATLAB gallery. The solid lines are  $\kappa_{ML}(A)u$ .

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 matrix  $A_{21}$  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 `m1m` does not for  $\beta \geq 7$ .

*Example 3: matrices from the MATLAB gallery.* Now we take 32 matrices of size  $10 \times 10$  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 errors are shown in Figure 6.3. We see that `m1m` 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 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 a multiprecision algorithm for evaluating analytic matrix functions  $f(A)$  that requires only function values and not derivatives. 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. Our algorithm needs arithmetic of precision at least double the working precision to evaluate  $f$  on the diagonal blocks of order greater than 2 (if there are any) of the reordered and blocked Schur form.

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 as a practical means 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 Schur form and then diagonalizing the perturbed blocks in higher precision.

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 `m1m` of [21].

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  $1 \times 1$  and  $2 \times 2$  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] A. H. AL-MOHY AND N. J. HIGHAM, *A new scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 970–989, <https://doi.org/10.1137/09074721X>.
- [2] A. H. AL-MOHY AND N. J. HIGHAM, *Improved inverse scaling and squaring algorithms for the matrix logarithm*, SIAM J. Sci. Comput., 34 (2012), pp. C153–C169, <https://doi.org/10.1137/110852553>.
- [3] E. ANDERSON, *Robust triangular solves for use in condition estimation*, Technical Report CS-91-142, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, Aug. 1991, <http://www.netlib.org/lapack/lawnpdf/lawn36.pdf>. LAPACK Working Note 36.

- [4] Z. BAI, J. W. DEMMEL, AND A. MCKENNEY, *On computing condition numbers for the non-symmetric eigenproblem*, ACM Trans. Math. Software, 19 (1993), pp. 202–223, <https://doi.org/10.1145/152613.152617>.
- [5] J. BANKS, J. GARZA-VARGAS, A. KULKARNI, AND N. SRIVASTAVA, *Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time*, ArXiv:1912.08805, 2019, <https://arxiv.org/abs/1912.08805v3>. Revised September 2020.
- [6] J. BANKS, J. GARZA-VARGAS, A. KULKARNI, AND N. SRIVASTAVA, *Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time*, in 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS), 2020, pp. 529–540, <https://doi.org/10.1109/FOCS46700.2020.00056>.
- [7] J. BANKS, A. KULKARNI, S. MUKHERJEE, AND N. SRIVASTAVA, *Gaussian regularization of the pseudospectrum and Davies' conjecture*, ArXiv:1906.11819, 2019, <https://arxiv.org/abs/1906.11819v4>. Revised April 2020.
- [8] J. BANKS, J. G. VARGAS, A. KULKARNI, AND N. SRIVASTAVA, *Overlaps, eigenvalue gaps, and pseudospectrum under real Ginibre and absolutely continuous perturbations*, ArXiv:2005.08930, May 2020, <https://arxiv.org/abs/2005.08930>.
- [9] W. W. BARRETT AND T. J. JARVIS, *Spectral properties of a matrix of Redheffer*, Linear Algebra Appl., 162–164 (1992), pp. 673–683, [https://doi.org/10.1016/0024-3795\(92\)90401-U](https://doi.org/10.1016/0024-3795(92)90401-U).
- [10] E. B. DAVIES, *Approximate diagonalization*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1051–1064, <https://doi.org/10.1137/060659909>.
- [11] P. I. DAVIES AND N. J. HIGHAM, *A Schur–Parlett algorithm for computing matrix functions*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 464–485, <https://doi.org/10.1137/S0895479802410815>.
- [12] J. W. DEMMEL, *The condition number of equivalence transformations that block diagonalize matrix pencils*, SIAM J. Numer. Anal., 20 (1983), pp. 599–610, <https://doi.org/10.1137/0720040>.
- [13] J. DUAN AND L. CHEN, *Solution of fractional differential equation systems and computation of matrix Mittag-Leffler functions*, Symmetry, 10 (2018), p. 503, <https://doi.org/10.3390/sym10100503>.
- [14] M. FASI AND N. J. HIGHAM, *Multiprecision algorithms for computing the matrix logarithm*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 472–491, <https://doi.org/10.1137/17M1129866>.
- [15] M. FASI AND N. J. HIGHAM, *An arbitrary precision scaling and squaring algorithm for the matrix exponential*, SIAM J. Matrix Anal. Appl., 40 (2019), pp. 1233–1256, <https://doi.org/10.1137/18M1228876>.
- [16] M. FASI, N. J. HIGHAM, AND B. IANNAZZO, *An algorithm for the matrix Lambert W function*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 669–685, <https://doi.org/10.1137/140997610>.
- [17] R. GARRAPPA, *Numerical evaluation of two and three parameter Mittag-Leffler functions*, SIAM J. Numer. Anal., 53 (2015), pp. 1350–1369, <https://doi.org/10.1137/140971191>.
- [18] R. GARRAPPA, I. MORET, AND M. POPOLIZIO, *Solving the time-fractional Schrödinger equation by Krylov projection methods*, J. Comp. Phys., 293 (2015), pp. 115–134, <https://doi.org/10.1016/j.jcp.2014.09.023>.
- [19] R. GARRAPPA, I. MORET, AND M. POPOLIZIO, *On the time-fractional Schrödinger equation: Theoretical analysis and numerical solution by matrix Mittag-Leffler functions*, Computers Math. Applic., 74 (2017), pp. 977–992, <https://doi.org/10.1016/j.camwa.2016.11.028>.
- [20] R. GARRAPPA AND M. POPOLIZIO, *On the use of matrix functions for fractional partial differential equations*, Math. Comput. Simulation, C-25 (2011), pp. 1045–1056, <https://doi.org/10.1016/j.matcom.2010.10.009>.
- [21] R. GARRAPPA AND M. POPOLIZIO, *Computing the matrix Mittag-Leffler function with applications to fractional calculus*, J. Sci. Comput., 77 (2018), pp. 129–153, <https://doi.org/10.1007/s10915-018-0699-5>.
- [22] N. J. HIGHAM, *The Matrix Computation Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mctoolbox>.
- [23] N. J. HIGHAM, *The Matrix Function Toolbox*. <http://www.maths.manchester.ac.uk/~higham/mftoolbox>.
- [24] N. J. HIGHAM, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430, [https://doi.org/10.1016/0024-3795\(87\)90118-2](https://doi.org/10.1016/0024-3795(87)90118-2).
- [25] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002, <https://doi.org/10.1137/1.9780898718027>.
- [26] N. J. HIGHAM, *Functions of Matrices: Theory and Computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008, <https://doi.org/10.1137/1.9780898717778>.

- [27] N. J. HIGHAM, *Short codes can be long on insight*, SIAM News, 50 (2017), pp. 2–3, <https://sinews.siam.org/Details-Page/short-codes-can-be-long-on-insight>.
- [28] N. J. HIGHAM AND A. H. AL-MOHY, *Computing matrix functions*, Acta Numerica, 19 (2010), pp. 159–208, <https://doi.org/10.1017/S0962492910000036>.
- [29] N. J. HIGHAM AND E. 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, Mar. 2020, <http://eprints.maths.manchester.ac.uk/2754/>.
- [30] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008), The Institute of Electrical and Electronics Engineers, New York, USA, 2019, <https://doi.org/10.1109/IEEESTD.2019.8766229>.
- [31] V. JAIN, A. SAH, AND M. SAWHNEY, *On the real Davies’ conjecture*, ArXiv:2005.08908v2, May 2020, <https://arxiv.org/abs/2005.08908v2>.
- [32] C. S. KENNEY AND A. J. LAUB, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209, <https://doi.org/10.1137/0610014>.
- [33] C. S. KENNEY AND A. J. LAUB, *Small-sample statistical condition estimates for general matrix functions*, SIAM J. Sci. Comput., 15 (1994), pp. 36–61, <https://doi.org/10.1137/0915003>.
- [34] C. B. MOLER AND C. F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49, <https://doi.org/10.1137/S00361445024180>.
- [35] I. MORET AND P. NOVATI, *On the convergence of Krylov subspace methods for matrix Mittag-Leffler functions*, SIAM J. Numer. Anal., 49 (2011), pp. 2144–2164, <https://doi.org/10.1137/080738374>.
- [36] *Multiprecision Computing Toolbox*. Advanpix, Tokyo. <http://www.advanpix.com>.
- [37] B. N. PARLETT, *Computation of functions of triangular matrices*, Memorandum ERL-M481, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, Nov. 1974.
- [38] M. POPOLIZIO, *Numerical solution of multiterm fractional differential equations using the matrix Mittag-Leffler functions*, Mathematics, 6 (2018), p. 7, <https://doi.org/10.3390/math601007>.
- [39] J. D. ROBERTS, *Linear model reduction and solution of the algebraic Riccati equation by use of the sign function*, Internat. J. Control, 32 (1980), pp. 677–687, <https://doi.org/10.1080/00207178008922881>. First issued as report CUED/B-Control/TR13, Department of Engineering, University of Cambridge, 1971.
- [40] M. R. RODRIGO, *On fractional matrix exponentials and their explicit calculation*, J. Differential Equations, 261 (2016), pp. 4223–4243, <https://doi.org/10.1016/j.jde.2016.06.023>.
- [41] A. SCHWARZ AND C. C. K. MIKKELSEN, *Robust task-parallel solution of the triangular Sylvester equation*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, E. Deelman, J. Dongarra, and K. Karczewski, eds., Springer, Cham, Switzerland, 2020, pp. 82–92, [https://doi.org/10.1007/978-3-030-43229-4\\_8](https://doi.org/10.1007/978-3-030-43229-4_8).
- [42] A. SCHWARZ, C. C. K. MIKKELSEN, AND L. KARLSSON, *Robust parallel eigenvector computation for the non-symmetric eigenvalue problem*, Parallel Comput., 100 (2020), p. 102707, <https://doi.org/10.1016/j.parco.2020.102707>.