SIAM J. Matrix Anal. Appl. © 2003 Society for Industrial and Applied Mathematics
Vol. 25, No. 2, pp. 464–485

A SCHUR–PARLETT ALGORITHM FOR COMPUTING MATRIX FUNCTIONS[*]

PHILIP I. DAVIES[†] AND NICHOLAS J. HIGHAM[†]

Abstract. An algorithm for computing matrix functions is presented. It employs a Schur decomposition with reordering and blocking followed by the block form of a recurrence of Parlett, with functions of the nontrivial diagonal blocks evaluated via a Taylor series. A parameter is used to balance the conflicting requirements of producing small diagonal blocks and keeping the separations of the blocks large. The algorithm is intended primarily for functions having a Taylor series with an infinite radius of convergence, but it can be adapted for certain other functions, such as the logarithm. Novel features introduced here include a convergence test that avoids premature termination of the Taylor series evaluation and an algorithm for reordering and blocking the Schur form. Numerical experiments show that the algorithm is competitive with existing special-purpose algorithms for the matrix exponential, logarithm, and cosine. Nevertheless, the algorithm can be numerically unstable with the default choice of its blocking parameter (or in certain cases for all choices), and we explain why determining the optimal parameter appears to be a very difficult problem. A MATLAB implementation is available that is much more reliable than the function `funm` in MATLAB 6.5 (R13).

Key words. matrix function, matrix exponential, matrix logarithm, matrix cosine, Taylor series, Schur decomposition, Parlett recurrence, sep function, LAPACK, MATLAB

AMS subject classification. 65F30

DOI. 10.1137/S0895479802410815

1. Introduction.

Matrix functions play a diverse role in science and engineering. They arise most frequently in connection with the solution of differential equations, with application areas including control theory [2], nuclear magnetic resonance [6], [15], Lie group methods for geometric integration [22, sect. 8], and the numerical solution of stiff ordinary differential equations [9]. A large body of theory on matrix functions exists, with comprehensive treatments available in [12] and [21], for example. In this work a function $f(A)$ of a matrix $A \in \mathbb{C}^{n\times n}$ has the usual meaning, which can be defined in terms of a Cauchy integral formula, a Hermite interpolating polynomial, or the Jordan canonical form, and we assume that f is "defined on the spectrum of A" (see any of the above references for details). The main property we need is that for each A, $f(A)$ is expressible as a polynomial in A (and of course that polynomial depends on A).

A wide variety of computational methods have been proposed, most of them geared to particular functions such as the exponential, the logarithm, and the square root. However, apart from the method of Kågström [24] discussed below, no numerically reliable method exists for computing $f(A)$ for a general function f. Such a method is needed for several reasons. First, software packages cannot provide special-purpose routines for all the functions that might be required. For example, MATLAB 6.5 (R13) provides routines to evaluate the matrix functions e^A (`expm`) and $A^{1/2}$ (`sqrtm`), but the matrix logarithm and matrix cosine, for example, must be computed via the routine `funm` for general f. (MATLAB has a routine `logm` that computes

[*]Received by the editors July 4, 2002; accepted for publication (in revised form) by B. Kågström March 31, 2003; published electronically September 9, 2003. This research was supported by Engineering and Physical Sciences Research Council grant GR/R22612.

http://www.siam.org/journals/simax/25-2/41081.html

[†]Department of Mathematics, University of Manchester, Manchester, M13 9PL, England (pdavies@ma.man.ac.uk, http://www.ma.man.ac.uk/~ieuan, higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham/).

the matrix logarithm, but it calls `funm`). MATLAB's `funm` has the capabilities that we are arguing for, but it is not numerically reliable, as is shown by our numerical experiments in section 7. The second benefit of a general purpose routine is that it provides a benchmark for comparison. Methods for specific f can be rejected if they offer no advantage over the best general method.

A general approach to compute $f(A)$ for $A \in \mathbb{C}^{n\times n}$ is to employ a similarity transformation

$$A = ZBZ^{-1}, \tag{1.1}$$

where $f(B)$ is easily computable. Then

$$f(A) = Zf(B)Z^{-1}. \tag{1.2}$$

If A is diagonalizable, for example, we can take $B = \operatorname{diag}(\lambda_i)$ and then $f(B) = \operatorname{diag}(f(\lambda_i))$ is trivially obtained. The drawback with this approach is that errors in evaluating $f(B)$ are multiplied by as much as $\kappa(Z) = \|Z\|\|Z^{-1}\| \geq 1$, yet the conditioning of $f(A)$ is not necessarily related to $\kappa(Z)$, so this approach may be numerically unstable. It is therefore natural to restrict to well conditioned transformations Z. Two ways do so are to take (1.1) to be a Schur decomposition, so that Z is unitary and B triangular, and to block diagonalize A using well conditioned transformations. We consider these two possibilities in the next two subsections.

1.1. Schur method. Computation of a Schur decomposition $A = QTQ^*$, where Q is unitary and T is upper triangular, is achieved with perfect backward stability by the QR algorithm [13, Chap. 7], so in computing $f(A) = Qf(T)Q^*$ the interest is in how to obtain $F = f(T)$. Since T is upper triangular, so is F (since it is a polynomial in T). Parlett [33] proposed using the following recurrence, which comes from equating (i, j) elements $(i < j)$ in the commutativity relation $FT = TF$:

$$f_{ij} = t_{ij}\frac{f_{ii} - f_{jj}}{t_{ii} - t_{jj}} + \sum_{k=i+1}^{j-1} \frac{f_{ik}t_{kj} - t_{ik}f_{kj}}{t_{ii} - t_{jj}}. \tag{1.3}$$

From (1.3) we see that any element of F can be calculated so long as all the elements to the left and below it are known. Thus the recurrence allows us to compute F a superdiagonal at a time, starting with the diagonal elements $f_{ii} = f(t_{ii})$. MATLAB's `funm` implements this Schur method.

Unfortunately, Parlett's recurrence breaks down when $t_{ii} = t_{jj}$ for some $i \neq j$, that is, when T has repeated eigenvalues, and it can give inaccurate results in floating point arithmetic when T has close eigenvalues. For example, if all the elements of F and T are $O(1)$ but T has two close eigenvalues with $t_{ii} - t_{jj} = O(\epsilon)$ (a not unreasonable scenario), then $t_{ij}(f_{ii} - f_{jj}) + \sum_{k=i+1}^{j-1}(f_{ik}t_{kj} - t_{ik}f_{kj}) = O(\epsilon)$, so that the sum suffers massive, and probably very damaging, cancellation.

Parlett [32] notes that if $T = (T_{ij})$ is block upper triangular, then $F = (F_{ij})$ has the same block structure and, for $i < j$,

$$T_{ii}F_{ij} - F_{ij}T_{jj} = F_{ii}T_{ij} - T_{ij}F_{jj} + \sum_{k=i+1}^{j-1}(F_{ik}T_{kj} - T_{ik}F_{kj}). \tag{1.4}$$

This recurrence can be used to compute F a block superdiagonal at a time, provided we can evaluate the blocks $F_{ii} = f(T_{ii})$ and solve the Sylvester equations (1.4) for the

F_{ij}. For the Sylvester equation (1.4) to be nonsingular we need that T_{ii} and T_{jj} have no eigenvalue in common. Moreover, for the Sylvester equations to be well conditioned a necessary condition is that the eigenvalues of T_{ii} and T_{jj} are well separated. Therefore to implement this block form of Parlett's recurrence we need first to reorder the Schur factor T into a block triangular matrix having two properties: distinct diagonal blocks have "sufficiently distinct" eigenvalues and, to aid the evaluation of f on the diagonal blocks, the eigenvalues within a block are "close." A parameter is required to define "close" and "sufficiently distinct."

1.2. Block diagonalization. An alternative approach is first to compute $A = XDX^{-1}$, where X is well conditioned and D is block diagonal. Then $f(A) = Xf(D)X^{-1}$ and the problem reduces to computing $f(D)$. The usual way to compute a block diagonalization is first to compute the Schur form and then to eliminate off-diagonal blocks by solving Sylvester equations [4], [13, sect. 7.6.3], [28]. In order to guarantee a well-conditioned X a bound must be imposed on the condition of the individual transformations; this bound will be a parameter in the algorithm.

Computing $f(D)$ reduces to computing $f(D_{ii})$ for each diagonal block D_{ii}. The D_{ii} are triangular but, unlike for the Schur method, no particular eigenvalue distribution is guaranteed, because of the limitations on the condition of the transformations; therefore $f(D_{ii})$ is still a nontrivial calculation.

1.3. Choice of method. The Schur method and the block diagonalization method are closely related. Both employ a Schur decomposition, both solve Sylvester equations, and both must compute $f(T_{ii})$ for atomic triangular blocks T_{ii} ("atomic" refers to the fact that these blocks cannot be further reduced). Parlett and Ng [34, sect. 5] show that the two methods are mathematically equivalent, differing only in the order in which two commuting Sylvester operators are applied. In this work we have chosen to use the Schur method, because it has the advantage that it produces atomic blocks with "close" eigenvalues—a property that we can exploit.

Our algorithm for computing $f(A)$ consists of several stages. The Schur decomposition $A = QTQ^*$ is computed, T is reordered to $\widetilde{T}$, the diagonal blocks $f(\widetilde{T}_{ii})$ are computed, the rest of $f(\widetilde{T})$ is computed using the block form of the Parlett recurrence, and finally the unitary similarity transformations from the Schur decomposition and the reordering are applied. We consider first, in section 2, the evaluation of f on the atomic blocks, for which we use a Taylor series expansion. This approach is mainly intended for functions whose Taylor series have an infinite radius of convergence, such as the exponential and the trigonometric and hyperbolic functions, but for some other functions, such as the logarithm, this step can be adapted or replaced by another technique. In section 3 we analyze the use of Parlett's recurrence. Based on the conflicting requirements of these two stages we describe our Schur reordering strategy in section 4.

Our algorithm is summarized in section 5 and the relevance of several preprocessing techniques is discussed in section 6. An extensive set of numerical experiments is described in section 7.

For real matrices, it is natural to use the real Schur decomposition in the first step of the algorithm and to attempt to work entirely in real arithmetic. However, the algorithm's strategy of placing eigenvalues that are not close in different blocks requires splitting complex conjugate pairs of eigenvalues having large imaginary parts, forcing complex arithmetic, so the algorithm does not lend itself to exploitation of the real Schur form.

We note that an attraction of the algorithm developed here is that it allows a function of the form $f(A) = \sum_i f_i(A)$ (e.g., $f(A) = \sin A + \cos A$) to be computed with less work than is required to compute each $f_i(A)$ separately, since the Schur decomposition and its reordering need only be computed once.

We emphasize that our goal is to develop a method applicable for a wide range of f. For particular f it will usually be possible to produce a more efficient or a more accurate algorithm. For example, for matrix pth roots reordering the Schur form is not necessary—the Schur-based methods of [5], [17], and [36] achieve essentially perfect numerical stability by exploiting elegant recurrences for pth roots of triangular matrices. In the case of the logarithm function our algorithm in its general form is not applicable, but we will specialize it to the logarithm and thereby obtain a method that is a candidate for the best general purpose $\log A$ method.

Ours is not the first work to exploit reordered Schur decompositions or the Parlett recurrence for computing matrix functions. Parlett's recurrence was used by Kågström in his thesis [24]. There are three main differences between Kågström's approach and ours. First, he used an initial block diagonalization, carried out with the method of Kågström and Ruhe [26], whereas we compute a Schur decomposition and reorder the triangular form. Second, Kågström uses the scalar rather than the block form of the Parlett recurrence and when t_{ii} and t_{jj} are sufficiently close he uses an explicit formula for f_{ij} involving derivatives (this formula is given in [13, Thm. 11.1.3], for example). Finally, we use a combination of Taylor series and the Parlett recurrence, whereas Kågström investigated the separate use of these two tools upon his block diagonal form. More recently, Parlett and Ng [34] developed an algorithm specifically for the matrix exponential that employs the Schur form with reordering and two levels of blocking, exponentiates the diagonal blocks using the Newton divided difference form of the interpolating polynomial, and uses the Parlett recurrence to obtain the off-diagonal blocks.

2. Evaluating functions of the atomic blocks.

Given an upper triangular matrix $T \in \mathbb{C}^{n\times n}$ whose eigenvalues are "close" and an arbitrary function f, we need a method for evaluating $f(T)$ efficiently and accurately. One approach, suggested by Stewart [30, Method 18] for the matrix exponential and investigated for general f by Kågström [24], is to expand f in a Taylor series about the mean of the eigenvalues of T. Write

$$T = \sigma I + M, \qquad \sigma = \operatorname{trace}(T)/n, \tag{2.1}$$

which defines M as T shifted by the mean of its eigenvalues, and let $\lambda(T)$ denote the set of eigenvalues of T. If f has a Taylor series representation

$$f(\sigma + z) = \sum_{k=0}^{\infty} \frac{f^{(k)}(\sigma)}{k!} z^k \tag{2.2}$$

for z in an open disk containing $\lambda(T - \sigma I)$, then

$$f(T) = \sum_{k=0}^{\infty} \frac{f^{(k)}(\sigma)}{k!} M^k. \tag{2.3}$$

If T has just one eigenvalue, so that $t_{ii} \equiv \sigma$, then M is strictly upper triangular and hence is nilpotent with $M^n = 0$; the series (2.3) is then finite. More generally, if the eigenvalues of T are sufficiently close, then the powers of M can be expected to decay quickly after the $(n-1)$st, and so a suitable truncation of (2.3) should yield good accuracy. We make this notion precise in the following lemma, in which we represent $M = D + N$, with D diagonal and N strictly upper triangular (that is, having zero diagonal) and hence nilpotent. For matrices, absolute values and inequalities are defined componentwise.

LEMMA 2.1. *Let $D \in \mathbb{C}^{n\times n}$ be diagonal with $|D| \le \delta I$ and let $N \in \mathbb{C}^{n\times n}$ be strictly upper triangular. Then*

$$|(D+N)^k| \le \sum_{i=0}^{\min(k,n-1)} \binom{k}{i} \delta^{k-i}|N|^i$$

and the same inequality holds with absolute values replaced by any consistent matrix norm.

Proof. The bound follows from

$$|(D+N)^k| \le (|D|+|N|)^k \le (\delta I + |N|)^k,$$

followed by a binomial expansion of the last term. Since $|N|^{n-1} = 0$ we can drop the terms involving $|N|^i$ for $i \ge n-1$. An analogous argument holds for any consistent matrix norm. $\square$

If $\delta < 1$ and $\delta \ll \|N\|$ in Lemma 2.1, then, for $k \ge n-1$,

$$\|(D+N)^k\| = O(\delta^{k+1-n}\|N\|^{n-1}),$$

and hence the powers of $D+N$ decay rapidly after the $(n-1)$st, irrespective of N.

This analysis shows that as long as the scalar multipliers $f^{(k)}(\sigma)/k!$ in (2.3) are not too large we should be able to truncate the series (2.3) soon after the $(n-1)$st term (and possibly much earlier if M is small).

We need a reliable criterion for deciding when to truncate the Taylor series. When summing a series whose terms decrease monotonically it is safe to stop as soon as a term is smaller than the desired error. Unfortunately, our matrix Taylor series can exhibit very nonmonotonic convergence. Indeed, when $n = 2$, $M = T - \sigma I$ always has the form

$$M = \begin{bmatrix} \epsilon & \alpha \\ 0 & -\epsilon \end{bmatrix}, \tag{2.4}$$

and its powers are

$$M^{2k} = \begin{bmatrix} \epsilon^{2k} & 0 \\ 0 & \epsilon^{2k} \end{bmatrix}, \quad M^{2k+1} = \begin{bmatrix} \epsilon^{2k+1} & \alpha\epsilon^{2k} \\ 0 & -\epsilon^{2k+1} \end{bmatrix}.$$

For $|\epsilon| < 1$, $\|M^k\| \to 0$ as $k \to \infty$, but $\|M^{2k+1}\| \gg \|M^{2k}\|$ for $\alpha \gg 1$. The next theorem shows that this phenomenon of the "disappearing nonnormal part" is connected with the fact that f can map distinct λ_i into the same value.

THEOREM 2.2. *Let $D \in \mathbb{C}^{n\times n}$ be diagonal with distinct eigenvalues $\lambda_1, \dots, \lambda_p$ $(1 \le p \le n)$ of multiplicity $k_1, \dots, k_p$, respectively, and let $f(z)$ be an analytic function on an open set containing $\lambda_1, \dots, \lambda_p$. Then $f(D+N) = f(D)$ for all strictly triangular $N \in \mathbb{C}^{n\times n}$ if and only if $f(D) = f(\lambda_1)I$ and*

$$f^{(j)}(\lambda_i) = 0, \quad j = 1\colon k_i - 1. \tag{2.5}$$

Note that (2.5) *is vacuous when $k_i = 1$.*

Proof. ($\Leftarrow$) For any strictly triangular N let $D+N = Z\mathrm{diag}(J_1,\dots,J_q)Z^{-1}$ $(q \geq p)$ be the Jordan canonical form of $D+N$ with Jordan blocks

$$J_i = \begin{bmatrix} \lambda_i & 1 & & & \\ & \lambda_i & 1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & 1 \\ & & & & \lambda_i \end{bmatrix} \in \mathbb{C}^{m_i \times m_i},$$

where, necessarily, m_i does not exceed the k_j corresponding to λ_i. Then

$$f(D+N) = Z\mathrm{diag}(f(J_1),\dots,f(J_q))Z^{-1},$$

where (from (2.3), for example)

$$f(J_i) = \begin{bmatrix} f(\lambda_i) & f'(\lambda_i) & \dots & \dots & \frac{f^{(m_i-1)}(\lambda_i)}{(m_i-1)!} \\ & f(\lambda_i) & f'(\lambda_i) & \dots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & f'(\lambda_i) \\ & & & & f(\lambda_i) \end{bmatrix}. \tag{2.6}$$

Since the derivatives of f are zero on any repeated eigenvalue and $f(\lambda_i) = f(\lambda_1)$ for all i, $f(D+N) = Zf(D)Z^{-1} = Zf(\lambda_1)IZ^{-1} = f(\lambda_1)I = f(D)$.

($\Rightarrow$) Let $F = f(D+N)$, and note that by assumption $F = f(D)$ and hence F is diagonal. The equation $F(D+N) = (D+N)F$ reduces to $FN = NF$, and equating (i,j) elements for $j > i$ gives $(f_{ii} - f_{jj})n_{ij} = 0$. Since this equation holds for all strictly triangular N, it follows that $f_{ii} = f_{jj}$ for all i and j and hence that $F = f(\lambda_1)I$.

If at least one of the λ_i is repeated, then we can find a permutation matrix P and a strictly upper bidiagonal matrix B such that $PDP^T + B = P(D + P^TBP)P^T$ is nonderogatory and is in Jordan canonical form, and $N = P^TBP$ is strictly upper triangular. We have $\lambda(D) = \lambda(D+N)$ and so the requirement $f(D+N) = f(D)$ implies that $f(PDP^T + B) = Pf(D)P^T = f(\lambda_1)I$, and hence, in view of (2.6), (2.5) holds. ☐

Applying Theorem 2.2 to the function $f(x) = x^k$ we obtain the following corollary.

COROLLARY 2.3. *Let $D \in \mathbb{C}^{n\times n}$ be a nonzero diagonal matrix and let $k \geq 2$. Then $(D+N)^k = D^k$ for all strictly triangular matrices $N \in \mathbb{C}^{n\times n}$ if and only if*

$$D = \beta\,\mathrm{diag}(e^{2k_1\pi i/k}, e^{2k_2\pi i/k}, \dots, e^{2k_n\pi i/k}),$$

where $\beta \neq 0$, $k_i \in \{0,1,\dots,k-1\}$ and the k_i are distinct (and hence $k \geq n$).

Proof. By Theorem 2.2, all the diagonal elements of D must be kth roots of the same number, β^k say. The condition (2.5) implies that any repeated diagonal element d_{ii} must satisfy $f'(d_{ii}) = kd_{ii}^{k-1} = 0$, which implies $d_{ii} = 0$ and hence $D = 0$; therefore D has distinct diagonal elements. ☐

As a check, we note that the diagonal of M in (2.4) is of the form in the corollary for even powers k. The corollary shows that this phenomenon of very nonmonotonic convergence of the Taylor series can occur when the eigenvalues are a constant multiple of kth roots of unity. As is well known, the computed approximations to multiple eigenvalues occurring in a single Jordan block tend to have this distribution. We will see in Experiment 4 in section 7 that this eigenvalue distribution also causes problems in finding a good blocking.

We now develop a strict bound for the truncation error of the Taylor series, which we will use to decide when to terminate the series.

THEOREM 2.4 ([13, Thm. 11.2.2]). *Let $Q^*AQ = T = \operatorname{diag}(\lambda_i) + N$ be a Schur decomposition of $A \in \mathbb{C}^{n\times n}$, where N is strictly upper triangular. If $f(z)$ is analytic on a closed convex set Ω whose interior contains $\lambda(A)$, then*

$$\|f(A)\|_\infty \le \left\| \sum_{r=0}^{n-1} \omega_r \frac{|N|^r}{r!} \right\|_\infty \le \max_{0\le r\le n-1} \frac{\omega_r}{r!} \ \|(I-|N|)^{-1}\|_\infty,$$

where

$$\omega_r = \sup_{z\in\Omega} |f^{(r)}(z)|.$$

THEOREM 2.5 ([29, Cor. 2]). *If f has the Taylor series*

$$f(\sigma + y) = \sum_{k=0}^{\infty} \alpha_k y^k, \quad \alpha_k = \frac{f^{(k)}(\sigma)}{k!}$$

for y in an open disk containing the eigenvalues of $Y \in \mathbb{C}^{n\times n}$, then

$$\left\| f(\sigma I + Y) - \sum_{k=0}^{s-1} \alpha_k Y^k \right\|_\infty \le \frac{1}{s!} \max_{0\le t\le 1} \|Y^s f^{(s)}(\sigma I + tY)\|_\infty. \tag{2.7}$$

We need to apply Theorem 2.5 with $Y = M$ in (2.1), and so we need to be able to bound $\max_{0\le t\le 1} \|M^s f^{(s)}(\sigma I + tM)\|_\infty$. The term M^s is needed anyway if we form the next term of the series. To bound $\max_{0\le t\le 1} \|f^{(s)}(\sigma I + tM)\|_\infty$ we can use Theorem 2.4 to show that

$$\max_{0\le t\le 1} \|f^{(s)}(\sigma I + tM)\|_\infty \le \max_{0\le r\le n-1} \frac{\omega_{s+r}}{r!} \ \|(I-|N|)^{-1}\|_\infty, \tag{2.8}$$

where $\omega_{s+r} = \sup_{z\in\Omega} |f^{(s+r)}(z)|$. By using (2.8) in (2.7) we can therefore bound the truncation error. The term $\|(I-|N|)^{-1}\|_\infty$ can be evaluated in just $O(n^2)$ flops[1] for the ∞-norm, since $I-|N|$ is an M-matrix: we solve the triangular system $(I-|N|)y = e$, where $e = [1,\ldots,1]^T$, and then $\|y\|_\infty = \|(I-|N|)^{-1}\|_\infty$ [20, sect. 8.3].

We now state our algorithm for evaluating a function of an atomic block via the Taylor series. We denote by u the unit roundoff.

ALGORITHM 2.6 (evaluating function of atomic block). *Given a triangular matrix $T \in \mathbb{C}^{n\times n}$ whose eigenvalues $\lambda_1,\ldots,\lambda_n$ are "close," a function f having the Taylor series (2.2) for z in an open disk containing $\lambda_i - \sigma$, $i = 1{:}\,n$, where $\sigma = n^{-1}\sum_{i=1}^n \lambda_i$, and the ability to evaluate derivatives of f, this algorithm computes $F = f(T)$ using a truncated Taylor series.*

```
    $\sigma = n^{-1}\sum_{i=1}^n \lambda_i$, $M = T - \sigma I$, tol $= u$
    $\mu = \|y\|_\infty$, where $y$ solves $(I-|N|)y = e$ and $N$ is the strictly
```

[1] One flop is a floating point addition, multiplication, or division.

```
upper triangular part of $T$.
$F_0 = f(\sigma)I_n$
$P = M$
for $s = 1{:}\infty$
    $F_s = F_{s-1} + f^{(s)}(\sigma)P$
    $P = PM/(s+1)$
    if $\|F_s - F_{s-1}\|_\infty \le \mathrm{tol}\|F_s\|_\infty$
        % Successive terms are close so check the truncation error bound.
        Estimate or bound $\Delta = \max_{0\le r\le n-1} \omega_{s+r}/r!$, where
        $\omega_{s+r} = \sup_{z\in\Omega} |f^{(s+r)}(z)|$, with $\Omega$ a closed convex set containing $\lambda(T)$.
        if $\mu\Delta\|P\|_\infty \le \mathrm{tol}\|F_s\|_\infty$, quit, end if
    end if
end for
```

Unless we are able to exploit particular properties of f, we can in practice take $\omega_{s+r} = \max\{\, |f^{(s+r)}(\lambda_i)| : \lambda_i \in \lambda(T)\,\}$.

Algorithm 2.6 costs $O(n^4)$ flops, since even if T has constant diagonal, so that M is nilpotent, the algorithm may need to form the first $n-1$ powers of M. Although we usually insist on $O(n^3)$ flops algorithms in numerical linear algebra, this higher order operation count is mitigated by three factors. First, n here is the size of a block, and in most cases the blocks will be of much smaller dimension than the original matrix. Second, M is an upper triangular matrix, so forming all the powers $M^2, \dots, M^{n-1}$ costs $n^4/3$ flops—a factor 6 less than the flop count for multiplying full matrices. Third, for certain particular f the function of the atomic blocks can be evaluated in $O(n^3)$ flops by a method particular to that f.

Since in our overall $f(A)$ algorithm we are not able to impose a fixed bound on the spread $\max_{i,j} |t_{ii} - t_{jj}|$ of the diagonal of T, Algorithm 2.6 is suitable in its stated form only for functions that have a Taylor series with an infinite radius of convergence, such as exp, cos, sin, cosh, and sinh.

We now turn to the effects of rounding errors on Algorithm 2.6. Ignoring truncation errors, standard error analysis [20] shows that the best possible forward error bound is of the form

$$|F - \widehat{F}| \le \frac{nu}{1-nu} \sum_{k=0}^{\infty} \frac{|f^{(k)}(\lambda)|}{k!} |M|^k.$$

If there is heavy cancellation in the sum (2.3), then a large relative error $\|F - \widehat{F}\|/\|F\|$ is possible. This danger is well known, particularly in the case of the matrix exponential [30]. A mitigating factor here is that our matrix T is chosen to have eigenvalues that are clustered, which tends to limit the amount of cancellation in the sum. However, for sufficiently far from normal T, damaging cancellation can take place. For general functions there is little we can do to improve the accuracy; for particular f we can of course apply alternative methods, as illustrated in the next subsection for the logarithm.

2.1. Matrix logarithm. We show how Algorithm 2.6 can be adapted in the important case of the matrix logarithm. We need to evaluate $\log T$, where log denotes the principal logarithm [8] and T is triangular with close eigenvalues. The basic approximation tools at our disposal are a Taylor series and a Padé approximation, both of which are applicable to $\log(I+E)$ with $\|E\| < 1$. We write $\log T = \log(I+E)$, with $E = T - I$. If $\|E\|_\infty \le \theta$, for some tolerance $\theta < 1$, then we will compute a degree

m diagonal Padé approximation to $\log(I+E)$ for a suitable m. If $\|E\|_\infty > \theta$, then we compute the principal square root of T, using the method of Björck and Hammarling [5], and make the same test on the square root. Since $T^{1/2^k} \to I$ as $k \to \infty$, we will eventually be able to apply the Padé approximation, after which we recover the desired logarithm from the relation (see, e.g., [8])

$$\log T = 2^k \log T^{1/2^k}. \tag{2.9}$$

The method we have described is the inverse scaling and squaring method introduced by Kenney and Laub [27]. Note that this method does not exploit the clustered nature of the eigenvalues of T. We might hope to exploit this property by writing $\log T = \log(\alpha \cdot \alpha^{-1} T) = \log(\alpha^{-1}T) + (\log \alpha)I$, where $\alpha = n^{-1}\sum_i t_{ii}$ (say), so that $\operatorname{diag}(\alpha^{-1}T) \approx I$. However, the multivalued nature of the log function can cause the second equality to fail (more precisely, it holds only if some of the logarithms are interpreted as a nonprincipal logarithm) and so we have not pursued this approach.

3. Evaluating the upper triangular part of $f(A)$.

We evaluate the upper triangular part of $F = f(T)$ using Parlett's recurrence (1.4), which we rewrite here as

$$T_{ii}F_{ij} - F_{ij}T_{jj} = F_{ii}T_{ij} - T_{ij}F_{jj} + \sum_{k=i+1}^{j-1} (F_{ik}T_{kj} - T_{ik}F_{kj}). \tag{3.1}$$

We assume that T has been reordered and blocked so that T_{ii} and T_{jj} have no eigenvalue in common for all $i \neq j$. This Sylvester equation is therefore nonsingular and it is easy to see that F_{ij} can be computed a column at a time, with each column obtained as the solution of a triangular system. Of particular concern is the propagation of errors in the recurrence. These errors are of two sources: errors in the evaluation of the diagonal blocks F_{ii}, and rounding errors in the formation and solution of (3.1). To gain insight into both types of error we consider the residual of the computed solution $\widehat{F}$:

$$T\widehat{F} - \widehat{F}T =: R, \tag{3.2}$$

where R_{ij} is the residual from the solution of the Sylvester equation (3.1). Although it is possible to obtain precise bounds on R, these are not important to our argument. Writing $\widehat{F} = F + \mathit{\Delta} F$, on subtracting $TF - FT = 0$ from (3.2) we obtain

$$T\mathit{\Delta} F - \mathit{\Delta} F T = R.$$

As for the original equation $TF - FT = 0$, this equation uniquely determines the off-diagonal blocks $\mathit{\Delta} F$ in terms of the diagonal blocks. Equating (i,j) blocks yields

$$\begin{aligned} T_{ii}\mathit{\Delta} F_{ij} - \mathit{\Delta} F_{ij}T_{jj} &= R_{ij} + \mathit{\Delta} F_{ii}T_{ij} - T_{ij}\mathit{\Delta} F_{jj} + \sum_{k=i+1}^{j-1} (\mathit{\Delta} F_{ik}T_{kj} - T_{ik}\mathit{\Delta} F_{kj}) \\ &=: B_{ij}, \end{aligned} \tag{3.3}$$

and these equations can be solved to determine $\mathit{\Delta} F_{ij}$ a block superdiagonal at a time.

It is straightforward to show that

$$\|\mathit{\Delta} F_{ij}\|_F \le \operatorname{sep}(T_{ii}, T_{jj})^{-1} \|B_{ij}\|_F, \tag{3.4}$$

where sep is the separation of T_{ii} and T_{jj} [13, sect. 7.2.4], [38],

$$\operatorname{sep}(T_{ii}, T_{jj}) = \min_{X \neq 0} \frac{\|T_{ii}X - XT_{jj}\|_F}{\|X\|_F}.$$

It follows that rounding errors introduced during the stage at which F_{ij} is computed (i.e., represented by R_{ij}) can lead to an error ΔF_{ij} of norm proportional to $\operatorname{sep}(T_{ii}, T_{jj})^{-1}\|R_{ij}\|$. Moreover, earlier errors (represented by the ΔF_{ij} terms on the right-hand side of (3.3)) can be magnified by a factor $\operatorname{sep}(T_{ii}, T_{jj})^{-1}$. It is also clear from (3.3) that even if $\operatorname{sep}(T_{ii}, T_{jj})^{-1}$ is not large, serious growth of errors in the recurrence (3.3) is possible if some off-diagonal blocks T_{ij} are large.

To maximize the accuracy of the computed $f(T)$ we clearly need the blocks T_{ii} to be as well separated as possible in the sense of sep. However, trying to maximize the separations between the diagonal blocks T_{ii} tends to produce larger blocks with less tightly clustered eigenvalues, which increases the difficulty of evaluating $f(T_{ii})$, so any strategy for reordering the Schur form is necessarily a compromise. Moreover, the unitary transformations that produce and then reorder the Schur form may be ill-determined functions of the original matrix A and can be the dominant source of error in the whole computation (see Experiment 9 in section 7), making attempts to maximize the separations ineffective.

Computing $\operatorname{sep}(T_{ii}, T_{jj})$ exactly when both blocks are $m \times m$ costs $O(m^4)$ flops, while condition estimation techniques allow an estimate to be computed at the cost of solving a few Sylvester equations, that is, in $O(m^3)$ flops [7], [18], [25]. It is unclear how to develop a reordering and blocking strategy for producing "large seps" at reasonable cost; in particular, it is unclear how to define "large." Indeed the maximal separations are likely to be connected with the conditioning of $f(T)$, but little or nothing is known about any such connections. More generally, how to characterize matrices for which the condition number of f is large is not well understood, even for the matrix exponential [13, sect. 11.3.1], [23], [37]. Recalling the equivalence mentioned in section 1.3 between block diagonalization and the use of the Parlett recurrence, a result of Gu [14] provides further indication of the difficulty of maximizing the seps: he shows that, given a constant τ, finding a similarity transformation with condition number bounded by τ that block diagonalizes a triangular matrix is NP-hard.

In the next section we will adopt a reordering and blocking strategy that bounds the right-hand side of the approximation

$$\operatorname{sep}(T_{ii}, T_{jj})^{-1} \approx \frac{1}{\min\{\, |\lambda - \mu| : \lambda \in \lambda(T_{ii}),\ \mu \in \lambda(T_{jj})\,\}}$$

by the reciprocal of a given tolerance. The right-hand side is a lower bound for the left that can be arbitrarily weak, but it is a reasonable approximation for matrices not too far from being normal.

It is natural to look for ways of improving the accuracy of the computed $\widehat{F}$ from the Parlett recurrence. One candidate is fixed precision iterative refinement of the systems (3.1). However, these systems are essentially triangular, and standard error analysis shows that the backward error is already small componentwise [20, Thm. 8.5]; fixed precision iterative refinement therefore cannot help. The only possibility is to use extended precision when solving the systems.

4. Reordering and blocking the Schur form. Given the upper triangular Schur factor T we will reorder it into a partitioned upper triangular matrix $\widetilde{T} = U^*TU = (\widetilde{T}_{ij})$, where U is unitary and two conditions hold:

1. *separation between blocks*:

$$\min\{\, |\lambda-\mu| : \lambda \in \lambda(\widetilde{T}_{ii}),\, \mu \in \lambda(\widetilde{T}_{jj}),\, i \neq j \,\} > \delta, \tag{4.1}$$

2. *separation within blocks*: for every block $\widetilde{T}_{ii}$ with dimension bigger than 1, for every $\lambda \in \lambda(\widetilde{T}_{ii})$ there is a $\mu \in \lambda(\widetilde{T}_{ii})$ with $\mu \neq \lambda$ such that $|\lambda-\mu| \le \delta$.

Here, $\delta > 0$ is a tolerance. The second property implies that for $\widetilde{T}_{ii} \in \mathbb{R}^{m\times m}$ $(m > 1)$

$$\max\{\, |\lambda-\mu| : \lambda,\mu \in \lambda(\widetilde{T}_{ii}),\, \lambda \neq \mu \,\} \le (m-1)\delta,$$

and this bound is attained when, for example, $\lambda(\widetilde{T}_{ii}) = \{\delta, 2\delta, \dots, m\delta\}$.

The following algorithm is the first step in obtaining the ordering. It can be interpreted as finding the connected components of the graph on the eigenvalues of T in which there is an edge between two nodes if the corresponding eigenvalues are a distance at most δ apart.

ALGORITHM 4.1 (block pattern). *Given a triangular matrix $T \in \mathbb{C}^{n\times n}$ with eigenvalues $\lambda_i \equiv t_{ii}$ and a tolerance $\delta > 0$, this algorithm produces a block pattern, defined by an integer vector q, for the block version of Parlett's method: the eigenvalue λ_i is assigned to the set S_{q_i}, and it satisfies the conditions that $\min\{|\lambda_i - \lambda_j| : \lambda_i \in S_p, \lambda_j \in S_q, p \neq q\} > \delta$ and, for each set S_i with more than one element, every element of S_i is within distance at most δ from some other element in the set. For each such set S_q, all the eigenvalues in S_q are intended to appear together in an upper triangular block $\widetilde{T}_{ii}$ of $\widetilde{T} = U^*TU$.*

```
p = 1
Initialize the $S_p$ to empty sets.
for $i = 1\colon n$
   if $\lambda_i \notin S_q$ for all $1 \le q < p$
      Assign $\lambda_i$ to $S_p$.
      $p = p + 1$
   end if
   for $j = i + 1\colon n$
      Denote by $S_{q_i}$ the set that contains $\lambda_i$.
      if $\lambda_j \notin S_{q_i}$
         if $|\lambda_i - \lambda_j| \le \delta$
            if $\lambda_j \notin S_k$ for all $1 \le k < p$
               Assign $\lambda_j$ to $S_{q_i}$.
            else
               Move the elements of $S_{\max(q_i,q_j)}$ to $S_{\min(q_i,q_j)}$.
               Reduce by 1 the indices of sets $S_q$ for $q > \max(q_i, q_j)$.
               $p = p - 1$
            end if
         end if
      end if
   end for
end for
```

Algorithm 4.1 provides a mapping from each eigenvalue λ_i of T to an integer q_i such that the set S_{q_i} contains λ_i. Our remaining problem is equivalent to finding a method for swapping adjacent elements in q to obtain a confluent permutation q'. A confluent permutation of n integers, $q_1, \dots, q_n$, is a permutation such that any repeated integers q_i are next to each other. For example, there are 3! confluent permutations of $(1,2,1,3,2,1)$ which include $(1,1,1,3,2,2)$ and $(3,2,2,1,1,1)$. Ideally we would like a confluent permutation that requires a minimal number of swaps to transform q to q'. Ng [31] notes that finding such a permutation is an NP-complete problem. He proves that the minimum number of swaps required to obtain a given confluent permutation is bounded above by $\frac{n^2}{2}(1-\frac{1}{k})$, where k is the number of distinct q_i, and that this bound is attainable [31, Thm. A.1]. In practice, since the QR algorithm tends to order the eigenvalues by absolute value in the Schur form, complicated strategies for determining a confluent permutation are not needed. The following method works well in practice: find the average index of the integers in q and then order the integers in q' in ascending average index. If we take our example $(1,2,1,3,2,1)$ and let g_k denoted the average index of the integer k, we see that $g_1 = (1+3+6)/3 = 3\frac{1}{3}$, $g_2 = (2+5)/2 = 3\frac{1}{2}$, and $g_3 = 4$. Therefore we try to obtain the confluent permutation $q' = (1,1,1,2,2,3)$ by a sequence of swaps of adjacent elements:

$$\begin{aligned} q = (1,2,1,3,2,1) &\to (1,1,2,3,2,1) \\ &\to (1,1,2,3,1,2) \qquad (4.2)\\ &\to (1,1,2,1,3,2) \qquad (4.3)\\ &\to (1,1,1,2,3,2) \qquad (4.4)\\ &\to (1,1,1,2,2,3) = q'. \end{aligned}$$

Swapping adjacent diagonal elements of T requires $20n$ flops, plus another $20n$ flops to update the Schur vectors, so the cost of the swapping is $40n$ times the number of swaps. The total cost is usually small compared with the overall cost of the algorithm.

Having determined the blocking and the desired confluent permutation we can make repeated calls to the LAPACK routine `xTREXC` [1] to obtain it. This routine applies a unitary similarity transformation to move the diagonal element of T with row index $j =$ `IFST` to row $i =$ `ILST`, which is achieved by performing a sequence of $|j-i|$ swaps of adjacent diagonal elements. For example, if $j > i$, the diagonal of T has the ordering

$$\ldots, \lambda_{i-1}, \lambda_j, \lambda_i, \lambda_{i+1}, \ldots, \lambda_{j-1}, \lambda_{j+1} \qquad (4.5)$$

after application of `xTREXC`. Notice that swaps (4.2)–(4.4) can be achieved through one call to the LAPACK routine `xTREXC` by requesting that $\lambda_6 \in S_1$ be moved to row 3. The following algorithm is expressed with MATLAB indexing notation for conciseness.

ALGORITHM 4.2 (obtaining a confluent permutation). *Given a vector $q \in \mathbb{R}^n$ containing all the integers $1, \ldots, k$ (some repeated if $k < n$), this algorithm obtains a confluent permutation according to the average indices of the integers in q. Returned is a swapping strategy, stored in vectors* `ILST` *and* `IFST`*, to be used by the LAPACK routine* `xTREXC` *to obtain a block form of T.*

```
Let $\phi(j)$ denote the number of $j$'s in $q$. $\beta = 1$.
for $i = 1\colon k$
    $g_i = (\sum_{q_j = i} j)/\phi(i)$
end for
Sort $g$ into ascending order $g_{y_1} \le \cdots \le g_{y_k}$, where $y$ is an index vector.
for $i = y$
    if any$(q(\beta\colon \beta + \phi(i) - 1) \neq i)$
```

```
            $f = \mathrm{find}(q == i); g = \beta{:}\, \beta + \phi(i) - 1$
            Concatenate $g(f \sim= g)$ and $f(f \sim= g)$ to the end of ILST and IFST,
            respectively.
            Let $v = \beta{:}\, f(\mathrm{end})$ and delete all elements of $v$ that are elements of $f$.
            $q(g(\mathrm{end}) + 1{:}\, f(\mathrm{end})) = q(v)$
            $q(g) = [i, \ldots, i]$
            $\beta = \beta + \phi(i)$
        end if
    end for
```

The routine xTREXC implements the swapping algorithm of Bai and Demmel [3], which has guaranteed backward stability and, since we are swapping only 1×1 blocks, always succeeds.

5. Overall algorithm.

Our complete Schur algorithm for computing $f(A)$ is as follows.

ALGORITHM 5.1 (Computing $f(A)$). *Given $A \in \mathbb{C}^{n\times n}$, a function f analytic on a closed convex set Ω whose interior contains the eigenvalues of A, and the ability to evaluate derivatives of f, this algorithm computes $F = f(A)$.*

```
    Compute the Schur decomposition $A = QTQ^*$ ($Q$ unitary, $T$ upper triangular).
    If $T$ is diagonal, $F = f(T)$, goto $(*)$, end if
    Using Algorithm 4.1 with $\delta = 0.1$, assign each eigenvalue $\lambda_i$ to a set $S_{q_i}$.
    Apply Algorithm 4.2 to the vector $q$ to produce a swapping strategy
    in ILST and IFST.
    for $k = 1{:}\, \mathrm{length}(\mathtt{ILST})$
        call xTREXC(V,n,T,n,Q,n,IFST(k),ILST(k),info)
    end for
    % Now $A = QTQ^*$ is our reordered Schur decomposition, with block $m \times m$ $T$.
    for $i = 1{:}\, m$
        Use Algorithm 2.6 to evaluate $F_{ii} = f(T_{ii})$.
        for $j = i - 1{:} -1{:}\, 1$
            Solve the Sylvester equation in (3.1) for $F_{ij}$.
        end for
    end for
$(*)$   $F = QFQ^*$
```

The cost of Algorithm 5.1 depends greatly on the eigenvalue distribution of A, and is roughly between $28n^3$ flops and $n^4/3$ flops. Note that Q, and hence F, can be kept in factored form, with a significant computational saving. This is appropriate if F needs just to be applied to a few vectors, for example.

Note that we have set the blocking parameter $\delta = 0.1$, which our experiments indicate is as good a default choice as any. This optimal choice of δ in terms of cost or accuracy is problem-dependent.

Algorithm 5.1 has a property noted as being desirable by Parlett and Ng [34]: it acts simply on simple cases. Specifically, if A is normal, so that the Schur decomposition is $A = QDQ^*$ with D diagonal, the algorithm simply evaluates $f(A) = Qf(D)Q^*$. At another extreme, if A has just one eigenvalue of multiplicity n, then the algorithm works with a single block, $T_{11} \equiv T$, and evaluates $f(T_{11})$ via its Taylor series expanded about the eigenvalue.

If we specialize to the matrix logarithm and use the inverse scaling and squaring method in place of Algorithm 2.6, as described in section 2, Algorithm 5.1 is similar to a Schur method for the matrix logarithm proposed by Dieci, Morini, and Papini

[10]. The main difference is that in the latter paper the eigenvalues are ordered in the Schur form by increasing modulus and then the Schur form is blocked, without any further reordering, so that (4.1) holds; this tends to lead to larger blocks than Algorithm 4.1. (Consider, for example, the case where $\delta = 0.1$ and the diagonal of T is 1, i, $-i$, 1.1, for which the ordering of [10] produces one 4×4 block, whereas Algorithm 2.6 produces one 2×2 block and two 1×1 blocks.)

6. Preprocessing.

In an attempt to improve the accuracy of Algorithm 5.1 we might try to preprocess the data before applying a particular stage of the algorithm. Two techniques that have been used in the past, notably in Ward's implementation of the scaling and squaring algorithm for computing the matrix exponential [39], are translation and diagonal scaling, and in [39] their purpose is to reduce the norm of the matrix.

Translation has no effect on our algorithm. Algorithm 2.6 for evaluating the Taylor series already translates the diagonal blocks, and further translations before applying the Parlett recurrence are easily seen to have no effect, because (3.1) is invariant under translations $T \to T - \alpha I$ and $F \to F - \beta I$.

A diagonal similarity transformation could be applied at any stage of the algorithm and then undone later. For example, such a transformation could be used in conjunction with Parlett's recurrence in order to make $U := D^{-1}TD$ less nonnormal than T and to increase the separations between diagonal blocks. In fact, by choosing D of the form $D = \operatorname{diag}(\theta^{n-1}, \dots, 1)$ we can make U arbitrarily close to diagonal form. Unfortunately, no practical benefit is gained: Parlett's recurrence involves solving triangular systems and the substitution algorithm is invariant under diagonal scalings (at least, as long as they involve only powers of the machine base). Similar comments apply to the evaluation of the Taylor series in Algorithm 2.6.

A diagonal similarity transformation may be beneficial at the outset, prior to computing the Schur decomposition. One can balance A with the aid of the standard balancing algorithm used in conjunction with the QR algorithm (function `balance` in MATLAB); this algorithm computes $B = D^{-1}AD$, where D is chosen so that the norm of the ith row and ith column are of similar magnitude for all i. Ward's algorithm [39] uses an initial balancing. Balancing is a heuristic that is not guaranteed to lead to a more accurate result. We omit balancing from Algorithm 5.1, while recognizing that it is potentially useful when we are dealing with badly scaled matrices.

7. Numerical experiments.

Our experiments were carried out in MATLAB 6.5 (R13) on a Pentium IV, for which the unit roundoff $u \approx 1.1 \times 10^{-16}$. Our implementation of Algorithm 5.1 comprises several M-files and a MEX file that calls the LAPACK routine `ZTREXC` (we call the LAPACK binary supplied with MATLAB). Unless otherwise stated, $\delta = 0.1$ in Algorithm 4.1.

In computing errors we take for the "exact" $f(A)$ an approximation X computed at high precision using MATLAB's Symbolic Math Toolbox (which invokes the Maple kernel). The (relative or forward) error in $\widehat{X}$ is defined to be

$$\|X - \widehat{X}\|_\infty / \|X\|_\infty.$$

In certain applications the componentwise relative error $\max_{i,j}(|x_{ij} - \widehat{x}_{ij}|/|x_{ij}|)$ might be of interest. However, while componentwise accuracy is potentially achievable in evaluating $f(T)$, the subsequent similarity transformation by Q will, in general, destroy any special structure in the error and lead at best to a small normwise error.

Table 7.1
Errors for Experiment 1*:* $A =$ `gallery('triw',8)`.

	Algorithm 5.1	`funm`
A	4.5e-16	7.0e-1
$A+$ `rand(n)*1e-8`	6.4e-15	1.2e-10
$A+$ `triu(rand(n))*1e-8`	3.4e-16	2.2e44

We also quote the (relative) condition number

$$\mathrm{cond}(A,f)=\lim_{\epsilon\to 0}\max_{\|E\|_2\le\epsilon\|A\|_2}\frac{\|f(A+E)-f(A)\|_2}{\epsilon\|f(A)\|_2},$$

which we estimate using the finite-difference power method proposed by Kenney and Laub [27].

We present ten experiments that give insight into the many facets of the $f(A)$ problem and our particular algorithm.

Experiment 1. Our first experiment shows the importance of using a block form of the Parlett recurrence. We take A to be the 8×8 triangular matrix with $a_{ii}\equiv 1$ and $a_{ij}\equiv -1$ for $j>i$, which is MATLAB's `gallery('triw',8)`. With f the exponential, Table 7.1 shows the errors for A and two small perturbations of A, one full and one triangular. The condition number of $f(A)$ is about 2 in each case, so we would expect to be able to compute $f(A)$ accurately. Algorithm 5.1 provides very good accuracy. MATLAB 6.5's `funm`, which employs the point version of the Parlett recurrence, performs badly, as expected in view of the repeated or close eigenvalues. This is an extreme example, in that Algorithm 5.1 takes just one block, the whole Schur factor T, and so reduces to evaluating the Taylor series of T.

Experiment 2. It is easy to show numerically the need for the safeguard in the test in Algorithm 2.6 for terminating the Taylor series. For the matrix

$$T=\begin{bmatrix}0.5 & 10^{12}\\ 0 & -0.5\end{bmatrix}$$

Algorithm 5.1 evaluates the exponential with error less than u, treating the matrix as one block and taking 10 terms of the Taylor series. If the Taylor series evaluation is terminated solely based on comparison of successive terms, thus omitting the derivative test in Algorithm 2.6, then only 4 terms are taken and the error is 5×10^{-8}.

Experiment 3. We give an example to show that for the exponential function Algorithm 5.1 can be much more accurate than the scaling and squaring method implemented in MATLAB's `expm`. We take the upper triangular matrix

```
T = gallery('triw',4,2^(60)) - diag([17 17 2 2]),
```

which has diagonal elements $-16,-16,-1,-1$ and off-diagonal elements $2^{60}\approx 11\times 10^{18}$. This badly scaled matrix causes great difficulty for `expm`, which yields a relative error of order 100. Algorithm 5.1 chooses the blocking (1: 2), (3: 4) (with no reordering) and produces a result correct to machine precision. We note that the more sophisticated scaling strategy proposed in [11] would improve the accuracy of the scaling and squaring method. The significance of this experiment is that it shows that our general purpose method can be significantly more accurate than one of the best available e^A implementations.

Experiment 4. The next experiment shows how Algorithm 5.1 can behave in an unstable manner. We compute e^T, where the upper triangular T is generated by the MATLAB code

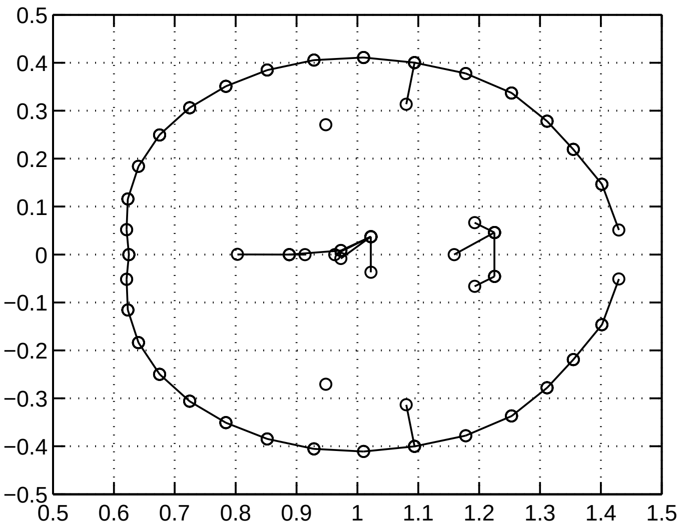

Fig. 7.1. *Eigenvalue distribution for Experiment* 4. *Circles denote eigenvalues and eigenvalues in the same block are joined by lines.*

```
n = 50; randn('state',1)
B = triu(randn(n),1) + eye(n);
Q = gallery('orthog',n);
B = Q*B*Q'; T = schur(B,'complex')
```

Although T has n eigenvalues 1 if formed in exact arithmetic, the computed T has eigenvalues mainly lying on and in an approximate circle of radius 0.4 centered on $(1,0)$. Algorithm 5.1 requires just 2 swaps to produce the block pattern

$$(1\colon 35) : 25 \text{ terms}, \quad (36\colon 36), \quad (37\colon 37), \quad (38\colon 42) : 11 \text{ terms}, \quad (43\colon 50) : 13 \text{ terms},$$

where the number of terms required in the Taylor series evaluation of each nontrivial diagonal block is shown. Figure 7.1 shows the eigenvalues and the blocking: the eigenvalues are represented by circles and a path is drawn between two eigenvalues if they belong to the same block. The condition number is $\mathrm{cond}(T,f) \approx 293$, but the error is 7×10^{-4}. Some insight is provided by Tables 7.2 and 7.3, which show, with T now the *reordered* Schur form, the blockwise errors $\|X_{ij} - \widehat{X}_{ij}\|_\infty / \|X_{ij}\|_\infty$ and the separations $\mathrm{sep}(T_{ii}, T_{jj})$ for $i \neq j$. The blocks with largest errors lie off the diagonal in the first block row and correspond to very small values of sep. This is not surprising in view of the bound (3.4).

An interesting feature of this example is that if we increase δ to 0.2, then Algorithm 5.1 chooses just one block and so calculates the exponential by a Taylor series of the whole of T, giving a result with error $1.4 \times 10^{-14} < \mathrm{cond}(T,f)u$. Figure 7.2 gives further insight by showing δ plotted against the error. The data for this plot was generated in such a way that all values of δ at which the blocking changes are included. The error is of order 10^{-4} for all δ until the first δ for which only one block is chosen. It seems that for this example any attempt to split eigenvalues into different blocks has a disastrous effect on the error.

Experiment 5. The previous experiment might suggest that it is better to overestimate δ. However, the graph of δ versus error can be U-shaped. Consider the exponential of minus the upper triangular Schur factor of the 50×50 Frank ma-

Table 7.2
Errors in blocks X_{ij} computed by Algorithm 5.1 *in Experiment* 4.

	j				
i	1	2	3	4	5
1	1.6e-14	1.9e-6	2.9e-6	2.3e-5	2.0e-3
2		1.1e-14	5.4e-15	6.6e-15	2.2e-12
3			2.1e-14	1.1e-14	5.4e-13
4				1.0e-14	4.8e-13
5					4.5e-14

Table 7.3
Values of $\text{sep}(T_{ii}, T_{jj})$ *for Experiment* 4.

	j				
i	1	2	3	4	5
1		2.2e-12	2.2e-12	3.4e-12	2.0e-13
2			5.4e-1	4.4e-2	8.3e-3
3				4.4e-2	8.3e-3
4					3.4e-5

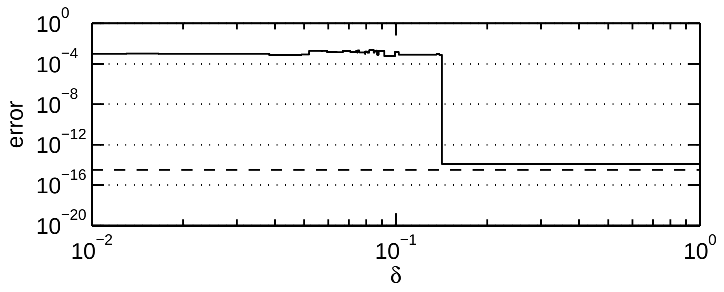

Fig. 7.2. *Blocking parameter δ versus error for Experiment* 4. *Dotted line denotes level of* $\text{cond}\, u$.

trix (MATLAB's `gallery('frank',50)`), for which $\text{cond}(A, f) \approx 2 \times 10^3$. Figure 7.3 shows the corresponding δ versus error plot; the error is near minimal for $2.3 \lesssim \delta \lesssim 5.2$ and increases rapidly outside this range.

Experiment 6. The next experiment shows that Algorithm 5.1 can fail to behave in a stable way for *all* choices of δ. The matrix is a 65×65 upper triangular matrix T constructed in MATLAB by

```
    A = -schur(gallery('frank',125),'complex')/2;
    i = [26:60 96:125]; T = A(i,i)
```

Figure 7.4 plots δ versus the error for the exponential function; the error is always at least 10^{-10}, which is three orders of magnitude greater than $\text{cond}(T, f)u$. Note, however, that varying δ does not generate all possible blockings, so we cannot rule out the possibility that the Schur–Parlett method is stable on this example for some other blocking. The following experiment provides further insight.

Experiment 7. For any particular matrix, it is of interest to know which blocking produces the most accurate computed result. We can answer this question experimentally by testing all possible blockings. The number S_n of blocking patterns for an $n \times n$ matrix can be shown to be

$$S_n = \sum_{k=1}^{n} S_n^{(k)},$$

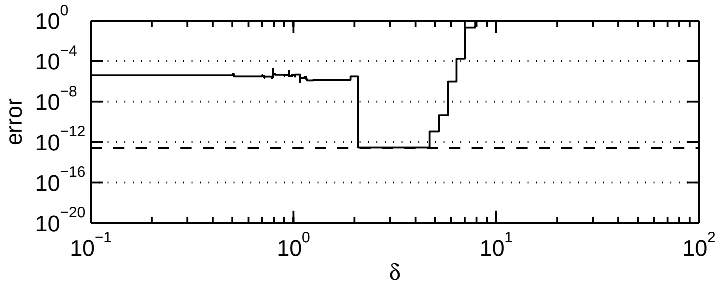

FIG. 7.3. *Blocking parameter δ versus error for Experiment* 5. *Dotted line denotes level of* cond u.

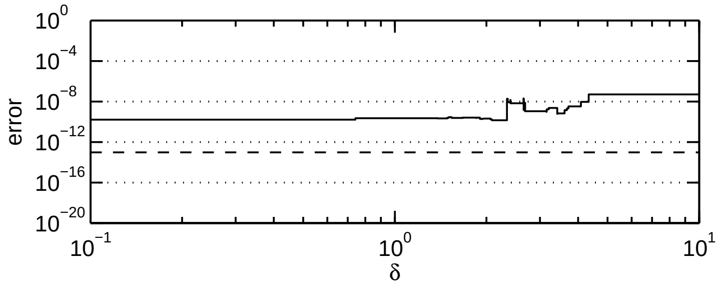

FIG. 7.4. *Blocking parameter δ versus error for Experiment* 6. *Dotted line denotes level of* cond u.

where $S_n^{(k)}$ is the number of ways a set of n elements can be partitioned into k disjoint, nonempty subsets. The numbers S_n and $S_n^{(k)}$ are known as Bell numbers and Stirling numbers of the second kind, respectively. The S_n grow very quickly, so it is feasible to try all orderings only for small n. We describe an example with $n = 10$, for which there are $S_{10} = 115,975$ different blockings. We generate an upper triangular T with the MATLAB code

```
n = 10;
mu = 0.2; phi = 5;
randn('state',0)
B = phi*triu(randn(n),1) + eye(n);
Q = gallery('orthog',n); B = Q*B*Q';
[U,T] = schur(B,'complex');
d = diag(T - eye(n)); delta = abs(d(1)-d(2));
T(1:n+1:n^2) = mu/delta*d + ones(n,1).
```

The computed eigenvalues of T lie approximately equally spaced on a circle center 1, radius 0.3.

Again, the function is the exponential, for which the condition number for this problem is 1.1×10^2. The results can be summarized as follows.

- Algorithm 5.1 chooses all 1×1 blocks and produces an error 2.8×10^{-10}.
- For the trivial blocking $\{1\colon 10\}$, the error is 3.9×10^{-16}. This blocking is produced by Algorithm 5.1 when δ is increased to 0.2.
- The other 115,974 nontrivial blockings produce errors ranging from 8.7×10^{-12} (for the blocking (1: 5), (6: 10)) to 3.0×10^{-9} (for the blocking (1: 2), (3: 4), (5: 6), 7, (8: 10)).

- For comparison, MATLAB's `expm` produces an error 1.4×10^{-14}.

Thus only the trivial blocking produces a computed result with error bounded by a small multiple of cond$u$. This example shows that the block Parlett recurrence can fail to behave in a forward stable way for *all* nontrivial blockings.

Experiment 8. Now we consider the matrix cosine function. A method specialized to this function is proposed by Serbin and Blalock [35] (see also [13, sect. 11.2.3]). The idea is to compute $\cos(A)$ by scaling by a power of 2 to produce a matrix with norm of order 1, approximate the cosine of the scaled matrix, then use the double-angle formula to recover the cosine of the original matrix:

```
    $C_0 =$ Taylor series approximation to $\cos(A/2^k)$
    for $j = 1\colon k$,
        $C_j = 2C_{j-1}^2 - I$
    end.
```

Here, we have specified a Taylor series approximation, though alternatives such as Padé approximants could also be used. Although some analysis of the method is given in [35], how to choose k and the degree of the Taylor approximation to strike a balance between minimizing the truncation error, rounding errors, and the computational effort is not understood. We have therefore implemented the following approach: we run the method with $k = 0\colon \lceil 2\log_2 \|A\|_1 \rceil$ and with the Taylor series evaluated with convergence tolerance u and record the smallest error observed. In other words, we find the most accurate solution that the method can provide for a wide range of k.

For the 6×6 Pascal matrix (MATLAB's `pascal(6)`), which has ∞-norm 462 Algorithm 5.1 produces a computed solution with error 9.0×10^{-15}; since this matrix is symmetric Algorithm 5.1 simply evaluates the cosine function on the diagonal matrix of eigenvalues. The double-angle method produces minimum error 8.5×10^{-13}, which is achieved for $k = 6$ and using 35 terms of the Taylor series.

For the MATLAB matrix $A =$ `gallery('invol',8)*pi`, which has ∞-norm of order 10^6 and eigenvalues $\pm\pi$, so that $\cos(A) = I$, the relative error for Algorithm 5.1 is 4.73×10^{-11}, resulting from the blocking $(1\colon 4)$, $(5\colon 8)$ with 4 Taylor series terms for each block (with no reordering). If just one block is taken, then 35 Taylor series terms are required and the error is about 6 times larger. The minimum error from the double-angle method is 8.6×10^{-14}, achieved for $k = 15$ and using 3 terms of the Taylor series. Interestingly, the error for $k = 0$, which evaluates $\cos(A)$ directly from the Taylor series, is 9.0×10^{-14}, while $k = 20 \approx \log_2(\|A\|_\infty)$ (which is suggested in [13]) produces a much larger error 2.0×10^{-11}. The condition number $\operatorname{cond}(A, f)$ is of order 10^8.

Our conclusion from this experiment is that Algorithm 5.1 is competitive in accuracy with the double-angle method, even when the optimal k is chosen for the latter method.

Experiment 9. Next we consider the matrix logarithm. In Algorithm 5.1 we use the inverse scaling and squaring method in place of Algorithm 2.6, as described in section 2; we take $\theta = 0.25$ and $m = 8$ and evaluate the Padé approximant by a partial fraction expansion, as recommended in [19]. We take the matrix $A = ZJZ^{-1}$ from [4], where

$$J = \operatorname{diag}(1, J_3(1), 0.3, 0.4, 0.5, 0.6, 0.7, 0.8),$$

with $J_m(\lambda)$ an $m \times m$ Jordan block with eigenvalue λ, and Z is a random matrix with condition number 10^8. The reordered Schur triangular factor, denoted by T, is blocked $(1\colon 1)$, $(2\colon 2)$, $(3\colon 3)$, $(4\colon 4)$, $(5\colon 5)$, $(6\colon 7)$, $(8\colon 10)$. The error in the computed

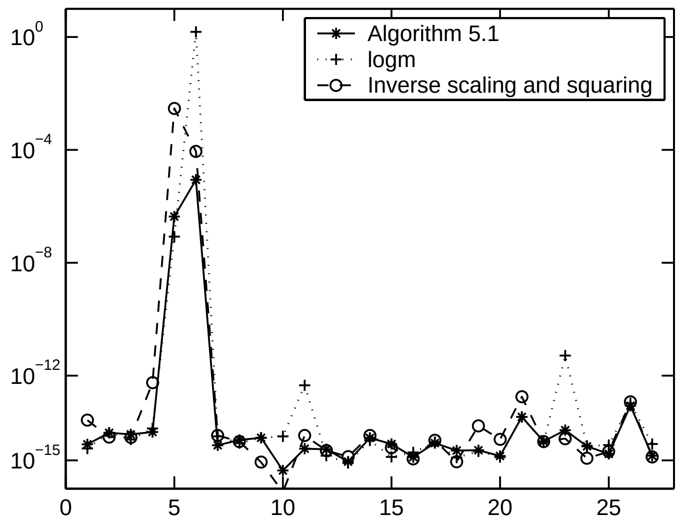

FIG. 7.5. *Error measure* (7.1) *for* 26 13×13 *test matrices.*

$X = \log A$ is $8 \times 10^{-4} \approx \operatorname{cond}(A, f)u$. However the error in the computed $\log T$ is only 1×10^{-14}, which is consistent with the fact that $\min_{i\neq j} \operatorname{sep}(T_{ii}, T_{jj}) = 1 \times 10^{-4}$. In this example, then, the error is dominated by the error introduced by the unitary transformations, and the error in the evaluation of the $\log T_{ii}$ and in the block Parlett recurrence is negligible, by comparison. Even if we evaluate $\log T$ to full working accuracy, the unitary back-transformations increase the error to the level 10^{-3} once again. This illustrates that although unitary transformations are perfectly backward stable, they can be the dominant source of forward error in Algorithm 5.1.

Experiment 10. In the final experiment we use the quantity

$$\beta = \frac{\|A - e^{\widehat{X}}\|_\infty}{\|A\|_\infty}, \tag{7.1}$$

where $\widehat{X}$ is the computed logarithm of A, to test the quality of three matrix logarithm methods: Algorithm 5.1 specialized to the logarithm as in the previous experiment, MATLAB 6.5's `logm` (which is essentially `funm` applied to the log function), and an implementation of the inverse scaling and squaring method that computes a Schur decomposition, takes square roots as necessary of the full triangular factor, and then computes a degree 8 diagonal Padé approximation. We use 27 13×13 matrices obtained from the function `matrix` in the Matrix Computation Toolbox [16]; these matrices include test matrices from MATLAB itself. The results, in Figure 7.5, show that Algorithm 5.1 performs at least as well as the other two logarithm methods for these test matrices.

8. Conclusions.

Algorithm 5.1 is applicable to a wide range of functions and imposes no restrictions on the matrix. It requires $O(n^3)$ flops unless close or repeated eigenvalues force large blocks to be chosen when the Schur form is blocked, in which case the operation count can be up to $n^4/3$ flops. The algorithm needs to evaluate derivatives of the function when there are blocks of dimension greater than 1. This is a price to be paid for catering for general functions and nonnormal matrices with possibly repeated eigenvalues.

The algorithm has a parameter δ that is used to determine the reordering and blocking of the Schur form. This parameter serves to balance the conflicting requirements of producing small diagonal blocks and keeping the separations of the blocks large. It is unclear how to choose δ to (nearly) maximize the accuracy of the computed $f(A)$. Indeed it is an open problem to understand fully the conditioning of general matrix functions, and a good choice of δ is likely to require knowledge of the conditioning. Our default choice of $\delta = 0.1$ performs well much of the time. The most difficult cases for our algorithm are when a substantial subset of the computed eigenvalues are approximately equally spaced on a circle in the complex plane, in which case the default δ may yield an unnecessarily inaccurate result. The option of running the algorithm with several different δ is not usually helpful in practice, because for most f we have no way to judge the quality of a computed $f(A)$ without comparing it with the exact answer. Moreover, it is possible that for all choices of δ the error is greater than the condition of the problem warrants (see Experiment 6). Nevertheless, as our numerical experiments make clear, even specialized methods, such as the scaling and squaring method for the matrix exponential, can behave unstably on certain examples, and Algorithm 5.1 is competitive with all the specialized algorithms to which we have compared it experimentally.

Our MATLAB implementation of Algorithm 5.1 is more robust and numerically reliable than MATLAB 6.5's `funm`, which ignores the dangers of close or repeated eigenvalues and always uses the point version of the Parlett recurrence. We hope that this implementation will serve as a benchmark with which to compare both specific $f(A)$ routines and other general purpose routines.

9. Acknowledgments. We thank Françoise Tisseur and Alessandra Papini for their comments on a draft manuscript. The comments of the referees were also very helpful.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, PA, 1999.

[2] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, 3rd ed., Prentice–Hall, Englewood Cliffs, NJ, 1997.

[3] Z. Bai and J. W. Demmel, *On swapping diagonal blocks in real Schur form*, Linear Algebra Appl., 186 (1993), pp. 73–95.

[4] C. A. Bavely and G. W. Stewart, *An algorithm for computing reducing subspaces by block diagonalization*, SIAM J. Numer. Anal., 16 (1979), pp. 359–367.

[5] Å. Björck and S. Hammarling, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.

[6] B. A. Borgias, M. Gochin, D. J. Kerwood, and T. L. James, *Relaxation matrix analysis of 2D NMR data*, Progress in NMR Spectroscopy, 22 (1990), pp. 83–100.

[7] R. Byers, *A LINPACK-style condition estimator for the equation $AX - XB^T = C$*, IEEE Trans. Automat. Control, 29 (1984), pp. 926–928.

[8] S. H. Cheng, N. J. Higham, C. S. Kenney, and A. J. Laub, *Approximating the logarithm of a matrix to specified accuracy*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1112–1125.

[9] S. M. Cox and P. C. Matthews, *Exponential time differencing for stiff systems*, J. Comput. Phys., 176 (2002), pp. 430–455.

[10] L. Dieci, B. Morini, and A. Papini, *Computational techniques for real logarithms of matrices*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 570–593.

[11] L. Dieci and A. Papini, *Padé approximation for the exponential of a block triangular matrix*, Linear Algebra Appl., 308 (2000), pp. 183–202.

[12] F. R. Gantmacher, *The Theory of Matrices*, Vol. 1, Chelsea, New York, 1959.

[13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[14] M. Gu, *Finding well-conditioned similarities to block-diagonalize nonsymmetric matrices is NP-hard*, J. Complexity, 11 (1995), pp. 377–391.

[15] T. F. Havel, I. Najfeld, and J. Yang, *Matrix decompositions of two-dimensional nuclear magnetic resonance spectra*, Proc. Natl. Acad. Sci. USA, 91 (1994), pp. 7962–7966.

[16] N. J. Higham, *The Matrix Computation Toolbox*, http://www.ma.man.ac.uk/~higham/mctoolbox (5 September 2002).

[17] N. J. Higham, *Computing real square roots of a real matrix*, Linear Algebra Appl., 88/89 (1987), pp. 405–430.

[18] N. J. Higham, *FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation (Algorithm* 674*)*, ACM Trans. Math. Software, 14 (1988), pp. 381–396.

[19] N. J. Higham, *Evaluating Padé approximants of the matrix logarithm*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 1126–1135.

[20] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, PA, 2002.

[21] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991.

[22] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna, *Lie-group methods*, Acta Numer., 9 (2000), pp. 215–365.

[23] B. Kågström, *Bounds and perturbation bounds for the matrix exponential*, BIT, 17 (1977), pp. 39–57.

[24] B. Kågström, *Numerical computation of matrix functions*, Report UMINF-58.77, Department of Information Processing, University of Umeå, Sweden, 1977.

[25] B. Kågström and P. Poromaa, *Distributed and shared memory block algorithms for the triangular Sylvester equation with* sep^{-1} *estimators*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 90–101.

[26] B. Kågström and A. Ruhe, *An algorithm for numerical computation of the Jordan normal form of a complex matrix*, ACM Trans. Math. Software, 6 (1980), pp. 398–419.

[27] C. Kenney and A. J. Laub, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209.

[28] P.-F. Lavallée, A. Malyshev, and M. Sadkane, *Spectral portrait of matrices by block diagonalization*, in Numerical Analysis and Its Applications, L. Vulkov, J. Waśniewski, and P. Yalamov, eds., Lecture Notes in Comput. Sci. 1196, Springer-Verlag, Berlin, 1997, pp. 266–273.

[29] R. Mathias, *Approximation of matrix-valued functions*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 1061–1063.

[30] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.

[31] K. C. Ng, *Contributions to the Computation of the Matrix Exponential*, Ph.D. thesis, Technical report PAM-212, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, 1984.

[32] B. N. Parlett, *Computation of functions of triangular matrices*, Memorandum ERL-M481, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, 1974.

[33] B. N. Parlett, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra Appl., 14 (1976), pp. 117–121.

[34] B. N. Parlett and K. C. Ng, *Development of an accurate algorithm for* $\exp(Bt)$, Technical report PAM-294, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, 1985.

[35] S. M. Serbin and S. A. Blalock, *An algorithm for computing the matrix cosine*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 198–204.

[36] M. I. Smith, *A Schur algorithm for computing matrix pth roots*, SIAM J. Matrix Anal. Appl., 24 (2003), pp. 971–989.

[37] C. Van Loan, *The sensitivity of the matrix exponential*, SIAM J. Numer. Anal., 14 (1977), pp. 971–981.

[38] J. M. Varah, *On the separation of two matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 216–222.

[39] R. C. Ward, *Numerical computation of the matrix exponential with accuracy estimate*, SIAM J. Numer. Anal., 14 (1977), pp. 600–610.