SIAM J. MATRIX ANAL. APPL.
Vol. 21, No. 4, pp. 1185–1201

A BLOCK ALGORITHM FOR MATRIX 1-NORM ESTIMATION, WITH AN APPLICATION TO 1-NORM PSEUDOSPECTRA*

NICHOLAS J. HIGHAM† AND FRANÇOISE TISSEUR†

Abstract. The matrix 1-norm estimation algorithm used in LAPACK and various other software libraries and packages has proved to be a valuable tool. However, it has the limitations that it offers the user no control over the accuracy and reliability of the estimate and that it is based on level 2 BLAS operations. A block generalization of the 1-norm power method underlying the estimator is derived here and developed into a practical algorithm applicable to both real and complex matrices. The algorithm works with $n \times t$ matrices, where t is a parameter. For $t = 1$ the original algorithm is recovered, but with two improvements (one for real matrices and one for complex matrices). The accuracy and reliability of the estimates generally increase with t and the computational kernels are level 3 BLAS operations for $t > 1$. The last $t-1$ columns of the starting matrix are randomly chosen, giving the algorithm a statistical flavor. As a by-product of our investigations we identify a matrix for which the 1-norm power method takes the maximum number of iterations. As an application of the new estimator we show how it can be used to efficiently approximate 1-norm pseudospectra.

Key words. matrix 1-norm, matrix norm estimation, matrix condition number, condition number estimation, p-norm power method, 1-norm pseudospectrum, LAPACK, level 3 BLAS

AMS subject classification. 65F35

PII. S0895479899356080

1. Introduction.

Research in matrix condition number estimation began in the 1970s with the problem of cheaply estimating the condition number $\kappa(A) = \|A\|\|A^{-1}\|$ and an approximate null vector of a square matrix A given some factorization of it. The earliest algorithm is one of Gragg and Stewart [10]. It was improved by Cline, Moler, Stewart, and Wilkinson [4], leading to the 1-norm condition estimation algorithm used in LINPACK [8] and later included in MATLAB (function `rcond`).

During the 1980s, attention was drawn to various componentwise condition numbers and it was recognized that most condition estimation problems can be reduced to the estimation of $\|A\|$ when matrix-vector products Ax and A^Tx can be cheaply computed [2], [16, Sec. 14.1]. Hager [12] derived an algorithm for the 1-norm that is a special case of the more general p-norm power method proposed by Boyd [3] and later investigated by Tao [18]. Hager's algorithm was modified by Higham [14] and incorporated in LAPACK (routine `xLACON`) [1] and MATLAB (function `condest`).

The LINPACK and LAPACK estimators both produce estimates that in practice are almost always within a factor 10 and 3, respectively, of the quantities they are estimating [13], [14], [15]. This has been entirely adequate for applications where only an order of magnitude estimate is required, such as the evaluation of error bounds. However, in some applications an estimate with one or more correct digits is required (see, for example, the pseudospectra application described in section 4), and for these the LINPACK and LAPACK estimators have the drawback that they offer the user no way to control or improve the accuracy of the estimate. Here, "accuracy" refers to

*Received by the editors May 10, 1999; accepted for publication by A. Edelman July 21, 1999; published electronically March 30, 2000. This work was supported by Engineering and Physical Sciences Research Council grants GR/L76532 and GR/L94314.

http://www.siam.org/journals/simax/21-4/35608.html

†Department of Mathematics, University of Manchester, Manchester, M13 9PL, England (higham@ma.man.ac.uk, ftisseur@ma.man.ac.uk).

TABLE 1
Empirical probabilities that $\min\{\widetilde{\phi}_s/\|A\|_1, \|A\|_1/\widetilde{\phi}_s\} \geq \alpha$ *for one* A *of the form* `inv(randn(100))` *and* $N(0,1)$ *vectors* x_j.

α	$s=1$	4	8	12	16	20
0.99	0.01	0.02	0.04	0.03	0.03	0.04
0.9	0.10	0.23	0.31	0.31	0.35	0.35
0.5	0.62	0.92	0.98	0.99	1.00	1.00
0.3	0.88	0.99	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00	1.00	1.00

average case behavior. Also of interest for a norm estimator is its worst case behavior, that is, its "reliability."

Since estimating $\|A\|$ cheaply appears inevitably to admit the possibility of arbitrarily poor estimates (although proving so is an open problem [6]), one might look for an approach for which probabilistic statements can be made about the accuracy of the estimate. The definition $\|A\| = \max_{x\neq 0} \|Ax\|/\|x\|$ of a subordinate matrix norm suggests the estimate

$$\phi_s = \max\left\{ \frac{\|Ax_j\|}{\|x_j\|} : j = 1\colon s \right\} \leq \|A\|, \tag{1.1}$$

where s is a parameter and the x_j are independently chosen random vectors. In the case of the 2-norm ($\|x\|_2 = (x^T x)^{1/2}$) and an appropriate distribution of the x_j, explicit bounds are available on the probability of such estimates being within a given factor of $\|A\|$ [7]. As is done in [11] for certain estimates of the Frobenius norm we can scale our estimates $\widetilde{\phi}_s \leftarrow \theta_s \phi_s$, where the constant θ_s is chosen so that the expected value of $\widetilde{\phi}_s$ is $\|A\|$ (note that $\widetilde{\phi}_s$ can therefore be greater or less than $\|A\|$). For the 1-norm ($\|x\|_1 = \sum_i |x_i|$), which is our interest here, we investigate this approach empirically. For a fixed matrix A of the form, in MATLAB notation, `inv(randn(100))`, Table 1 shows the observed probabilities that $\min\{\widetilde{\phi}_s/\|A\|_1, \|A\|_1/\widetilde{\phi}_s\} \geq \alpha$ for various α and s, based on 1000 separate evaluations of the ϕ_s with vectors x_j from the normal $N(0,1)$ distribution, and where θ_s is determined empirically so that the mean of the $\widetilde{\phi}_s$ is $\|A\|_1$.

The table shows that even with $s = n/5 = 20$ samples only 35% of the estimates were within a factor 0.9 of the true norm. The statistical sampling technique is clearly too crude to be useful for obtaining estimates with correct digits. One way to exploit the information contained in the vectors Ax_j is to regard them as first iterates from the 1-norm power method with starting vectors x_j and to continue to iterate. These considerations motivate the block generalization of the 1-norm power method that we present in this paper. Our block power method works with a matrix with t columns instead of a vector. To give a feel for how our new estimator compares with the sampling technique, we applied the estimator (Algorithm 2.4) to 1000 random matrices of the form `inv(randn(100))`. The results are shown in Table 2; the estimates for $t = 1\colon 5$ are obtained at approximately the same cost as the estimates $\widetilde{\phi}_s$ for $s = 4, 8, 12, 16, 20$, respectively. The superiority of the new estimator is clear.

In section 2 we derive the block 1-norm power method and develop it into a practical algorithm for both real and complex matrices. In section 3 we present numerical experiments that give insight into the behavior of the algorithm. An application involving complex matrices is given in section 4, where we describe how the algorithm

TABLE 2
Empirical probabilities that est $\geq \alpha\|A\|_1$, *for* est *from Algorithm* 2.4 *and* A *of the form* `inv(randn(100))`.

α	$t=1$	2	3	4	5
0.99	0.86	0.93	0.98	0.98	0.99
0.9	0.94	0.98	1.00	1.00	1.00
0.5	1.00	1.00	1.00	1.00	1.00

can be used to approximate 1-norm pseudospectra. Conclusions are presented in section 5.

Finally, we note that although our work is specific to the 1-norm, the ∞-norm can be estimated by applying our algorithm to A^*, since $\|A\|_\infty = \|A^*\|_1$.

2. Block 1-norm power method. The 1-norm power method is a special case of Boyd's p-norm power method [3] and was derived independently by Hager [12]. For a real matrix A we denote by $\mathrm{sign}(A)$ the matrix with (i,j) element 1 or -1 according as $a_{ij} \geq 0$ or $a_{ij} < 0$. The jth column of the identity matrix is denoted by e_j.

ALGORITHM 2.1 (1-norm power method). *Given* $A \in \mathbb{R}^{n\times n}$ *this algorithm computes* γ *and* x *such that* $\gamma \leq \|A\|_1$ *and* $\|Ax\|_1 = \gamma\|x\|_1$.

$x = \mathrm{ones}(n,1)/n$
repeat
  $y = Ax$
  $\xi = \mathrm{sign}(y)$
  $z = A^T\xi$
  if $\|z\|_\infty \leq z^T x$
    $\gamma = \|y\|_1$
    quit
  end
  $x = e_j$, where $|z_j| = \|z\|_\infty$ (smallest such j)
end

Algorithm 2.1 was modified by Higham [14, Alg. 4.1] (see also [15], [16, Alg. 14.4]) to improve its reliability and efficiency. The modifications that improve the reliability are, first, to force at least two iterations and, second, to take as the final estimate the maximum of that produced by the algorithm and

$$\|Ab\|_1/\|b\|_1, \qquad b_i = (-1)^{i+1}\left(1 + \frac{i-1}{n-1}\right), \qquad i = 1\colon n. \tag{2.1}$$

The vector b is a heuristic choice intended to "pick out" any large elements of A in those cases where such elements fail to be revealed during the course of the algorithm. Efficiency is improved by terminating the algorithm after computing ξ if it is the same as the previous ξ, since it can be shown that convergence would otherwise be declared after the subsequent computation of z.

To obtain a more accurate and reliable estimate than that provided by Algorithm 2.1 we could run the algorithm t times in succession on t different starting vectors. This idea was suggested in [12], with each starting vector being the mean of the unit vectors e_j not already visited and with the algorithm being prohibited from visiting unit vectors previously visited. Note that the estimates obtained this way are nondecreasing in t. This approach has two weaknesses: it allows limited communication of information between the t different iterations and the highest level computational kernel remains matrix-vector multiplication. We therefore develop a

block algorithm that works with an $n \times t$ matrix as a whole instead of t separate n-vectors. The block approach offers the potential of better estimates, through providing more information on which to base decisions, and it allows the use of level 3 BLAS operations, thus promising greater efficiency. The following algorithm estimates not just the 1-norm of A, but, as a by-product, the 1-norms of the t columns of A having largest 1-norms.

ALGORITHM 2.2 (block 1-norm power method). *Given $A \in \mathbb{R}^{n\times n}$ and a positive integer t, this algorithm computes vectors g and* ind *with $g_j = \|A(:, \mathrm{ind}_j)\|_1$ and $g_1 \geq \cdots \geq g_t$ such that g_j is a lower bound for the 1-norm of the column of A of jth largest 1-norm.*

Choose starting matrix $X \in \mathbb{R}^{n\times t}$ with columns of unit 1-norm.
repeat
  $Y = AX$
  $g_j = \|Y(:, j)\|_1$, $j = 1\colon t$
  Sort g so that $g_1 \geq \cdots \geq g_t$.
  ind_best $= \mathrm{ind}_j$ where $g_1 = \|Y(:, j)\|_1$
  $S = \mathrm{sign}(Y)$
  $Z = A^T S$
  $h_i = \|Z(i, :)\|_\infty$, $\mathrm{ind}_i = i$, $i = 1\colon n$
  if $\max(h_i) \leq Z(:, \mathrm{ind\_best})^T X(:, \mathrm{ind\_best})$, quit, end
  Sort h so that $h_1 \geq \cdots \geq h_t$ and reorder ind correspondingly.
  $X(:, j) = e_{\mathrm{ind}_j}$, $j = 1\colon t$
end

Like the basic 1-norm power method ($t = 1$) [14], Algorithm 2.2 has the attractive property that it generates increasing sequences of estimates. Denote with a superscript "(k)" quantities from the kth iteration of the loop in Algorithm 2.2 and let a_j denote the jth column of A.

LEMMA 2.3. *The sorted vectors $g^{(k)}$ and $h^{(k)}$ satisfy*

$$g_1^{(k)} \leq h_1^{(k)} \leq g_1^{(k+1)}, \qquad k \geq 1 \tag{2.2}$$

and

$$g_j^{(k)} \leq g_j^{(k+1)}, \qquad j = 1\colon t, \quad k \geq 2. \tag{2.3}$$

Proof. First, we have

$$g_1^{(k)} = \max_{1\leq j\leq t} \|y_j^{(k)}\|_1 =: \|y_r^{(k)}\|_1.$$

But

$$\|y_r^{(k)}\|_1 = {s_r^{(k)}}^T y_r^{(k)} = {s_r^{(k)}}^T A x_r^{(k)} = {z_r^{(k)}}^T x_r^{(k)}.$$

Thus

$$g_1^{(k)} = {z_r^{(k)}}^T x_r^{(k)} \leq \|z_r^{(k)}\|_\infty \|x_r^{(k)}\|_1 = \|z_r^{(k)}\|_\infty \leq \max_{i,j} |z_{ij}^{(k)}| = h_1^{(k)}.$$

Furthermore, if $h_1^{(k)} = \|Z^{(k)}(r, :)\|_\infty$, then

$$h_1^{(k)} = \max_j |a_r^T s_j^{(k)}| \leq \|a_r\|_1 = \|y_1^{(k+1)}\|_1 \leq g_1^{(k+1)}.$$

To prove (2.3), assume without loss of generality that $x_j^{(k-1)} = e_j$, $j = 1\colon t$. Then $Y^{(k)} = [a_1 \ \dots \ a_t]$ and $Z^{(k)}$ has the form

$$Z^{(k)} = \begin{bmatrix} \|a_1\|_1 & \alpha_1 & \dots & \alpha_1 \\ \alpha_2 & \|a_2\|_1 & \dots & \alpha_2 \\ \vdots & & \ddots & \vdots \\ \alpha_t & \dots & \alpha_t & \|a_t\|_1 \\ \alpha_{t+1} & & & \alpha_{t+1} \\ \vdots & & & \vdots \\ \alpha_n & \dots & \dots & \alpha_n \end{bmatrix},$$

where each α_i in row i is, in general, different, and $\alpha_i \le \|a_i\|_1$, $i = 1\colon n$. The algorithm chooses the ind values corresponding to the t rows of $Z^{(k)}$ with largest ∞-norm, and the $g_j^{(k+1)}$ on the next stage are at least as large as these ∞-norms. Since row j of $Z^{(k)}$ has ∞-norm $\|a_j\|_1$ for $j = 1\colon t$ and $\{g_j^{(k)}\}_{j=1}^t = \{\|a_j\|_1\}_{j=1}^t$, inequality (2.3) follows. ∎

Algorithm 2.2 has three possible sources of inefficiency. First, the columns of S are vectors of ± 1s and so a pair of columns s_i and s_j may be parallel ($s_i = \pm s_j$), in which case $z_i = \pm z_j$ and in computing Z a matrix-vector multiplication is redundant. There can be up to $t-1$ redundant matrix-vector products per iteration, this maximum being achieved on the second iteration with S the matrix of ones when A has nonnegative elements. The second possible inefficiency is that a column of S may be parallel to one from the previous iteration, in which case the formation of $Z = A^T S$ again involves redundant computation. We choose to detect parallel columns and replace them by random vectors rand$\{-1, 1\}$ not already in S or the previous S; here, rand$\{-1, 1\}$ denotes a vector with elements from the uniform distribution on the set $\{-1, 1\}$. The detection is done by forming inner products between columns and looking for elements of magnitude n. The total cost of these computations is $O(nt^2)$ flops, which is negligible compared with the $2n^2t$ flops required for each matrix product in the algorithm, since $t \ll n$ in practice.

Our strategy could be extended to check for parallel columns between the current S and all previous S; we return to this possibility in section 3. If all the columns of S are parallel to columns from the previous iteration, then it is easy to see that Algorithm 2.2 is about to converge; we therefore immediately terminate the iteration without computing Z.

Finally, on step k we can have $x_j^{(k)} = e_s = x_i^{(r)}$ for some i and some $r < k$, so that the computation of $Ax_j^{(k)}$ then repeats an earlier computation. Repeated vectors e_j are easily avoided by keeping track of the indices of all the previously used e_j and selecting $\mathrm{ind}_1, \dots, \mathrm{ind}_t$ from among the indices not previously used. If all the indices are repeats, then again we prematurely terminate the iteration, saving a matrix product.

Note that the first and third inefficiencies are possible only for $t > 1$, while the second can happen for the original 1-norm power method.

Our strategy of detecting redundant computations and replacing them by computations that provide new information has three benefits. First, it can reduce the amount of computation, through premature detection of convergence. Second, it can lead to better estimates. For the columns of S this depends on the random

replacement vectors generated, but for the e_j the improvement is deterministic up to future replacements of columns of S. The third benefit is that the dimensions of the matrix multiplications remain constant at each iteration, as opposed to varying if we simply skip redundant computations; this helps us to make efficient use of the computing resources.

The next algorithm incorporates these modifications. The algorithm is forced to take at least 2 (and at most itmax) iterations so that it computes at least t columns of A; it also explicitly identifies the approximate maximizing vector that achieves the norm estimate.

ALGORITHM 2.4 (practical block 1-norm estimator). *Given $A \in \mathbb{R}^{n\times n}$ and positive integers t and* itmax ≥ 2*, this algorithm computes a scalar* est *and vectors v and w such that* est $\leq \|A\|_1$*, $w = Av$, and $\|w\|_1 = \text{est}\|v\|_1$.*

Choose starting matrix $X \in \mathbb{R}^{n\times t}$ with columns of unit 1-norm.
ind_hist = [ ] % Integer vector recording indices of used unit vectors e_j.
$\text{est}_{\text{old}} = 0$, ind = zeros$(n, 1)$, S = zeros(n, t)
for $k = 1, 2, \ldots$
  $Y = AX$
  est = max$\{\, \|Y(:, j)\|_1 : j = 1\colon t \,\}$
  if est $> \text{est}_{\text{old}}$ or $k = 2$
    ind_best $= \text{ind}_j$ where est $= \|Y(:, j)\|_1$, $w = Y(:, \text{ind\_best})$
  end
(1) if $k \geq 2$ and est $\leq \text{est}_{\text{old}}$, est $= \text{est}_{\text{old}}$, goto (6), end
  $\text{est}_{\text{old}} = \text{est}$, $S_{\text{old}} = S$
  if $k >$ itmax, goto (6), end
  $S = \text{sign}(Y)$
(2) If every column of S is parallel to a column of S_{old}, goto (6), end
  if $t > 1$
    Ensure that no column of S is parallel to another column of S
    or to a column of S_{old} by replacing columns of S by rand$\{-1, 1\}$.
  end
(3) $Z = A^T S$
  $h_i = \|Z(i, :)\|_\infty$, $\text{ind}_i = i$, $i = 1\colon n$
(4) if $k \geq 2$ and $\max(h_i) = h_{\text{ind\_best}}$, goto (6), end
  Sort h so that $h_1 \geq \cdots \geq h_n$ and re-order ind correspondingly.
  if $t > 1$
(5) If ind$(1\colon t)$ is contained in ind_hist, goto (6), end
    Replace ind$(1\colon t)$ by the first t indices in ind$(1\colon n)$ that are
    not in ind_hist.
  end
  $X(:, j) = e_{\text{ind}_j}$, $j = 1\colon t$
  ind_hist = [ind_hist ind$(1\colon t)$]
end
(6) $v = e_{\text{ind\_best}}$

Note that this algorithm does not explicitly compute lower bounds for the 1-norms of all t largest columns of A. If this information is required (and we are not aware of any applications in which it is needed), then it can be obtained by keeping track of the largest $\|y_j^{(k)}\|_1$ values encountered.

Inequality (2.2), now expressed as $\text{est}^{(k)} \leq h_1^{(k)} \leq \text{est}^{(k+1)}$, is still valid, except that $h_1^{(k)} > \text{est}^{(k+1)}$ is possible on the last iteration if the original $\text{ind}_1^{(k)}$ is a repeat

(this event is handled by the test (1)). However, (2.3) is no longer true, because of the avoidance of repeated indices.

How do Algorithms 2.4 and 2.2 compare? Ignoring the itmax test, Algorithm 2.4 terminates in at most $n/t+1$ iterations, since t vertices e_j are visited on each iteration after the first and no vertex can be visited more than once. The same cannot be said of Algorithm 2.2 because of the possibility of repeated vertices. Algorithm 2.4 can produce a smaller estimate than Algorithm 2.2: when a redundant computation is avoided, the new information computed can lead to an apparently more promising vertex (based on the relative sizes of the h_i) replacing one that actually corresponds to a larger column of A. However, it is more likely, when Algorithm 2.4 and Algorithm 2.2 produce different results, that Algorithm 2.4 produces a better estimate, as in the following example.

Example 2.5. For a certain 10×10 matrix A, with $t = 2$ and a random starting matrix X_1 we obtained the following results. For Algorithm 2.2,

```
1:  (0, 8.10e-001) (0, 6.13e-001)
2:  (10, 1.77e+000) (4, 1.76e+000)
Underestimation ratio: 1.99e-001
```

The first column denotes the iteration number. The kth row gives the sorted $g_i^{(k)}$, $j = 1{:}t$, each one preceded by the corresponding index ind_i. (Since X_1 does not have columns e_j, the ind_i for the first iteration are shown as zero.) For Algorithm 2.4,

```
1:  (0, 8.10e-001) (0, 6.13e-001)
2:  (10, 1.77e+000) (4, 1.76e+000)
    1 parallel column between S and S_old
3:  (2, 8.87e+000) (5, 4.59e+000)
Exact estimate!
```

Algorithm 2.2 converges after two iterations and produces an estimate too small by a factor 5. However, on the second iteration Algorithm 2.4 detects a column of S parallel to one of S_{old} and replaces it. The new column produces a different Z matrix and causes the convergence test (4) to be failed. The extra iteration visits the (unique) column of maximum 1-norm, so an exact estimate is obtained.

When $t = 1$, Algorithm 2.4 differs from the modified version of Algorithm 2.1 used in LAPACK 3.0 [14, Alg. 4.1], [16, Alg. 14.4] in two ways. First, Algorithm 2.4 does not use the "extra estimate" (2.1). Second, the LAPACK algorithm checks for $\xi^{(k)} = \xi^{(k-1)}$ but not for $\xi^{(k)} = -\xi^{(k-1)}$ as does Algorithm 2.4. This is an oversight. We recommend that LAPACK's `xLACON` be modified to include the extra test. This change will not affect the estimates produced but will sometimes reduce the number of iterations.

We now explain our choice of starting matrix. We take the first column of X to be the vector of 1s, which is the starting vector used in Algorithm 2.1. This has the advantage that for a matrix with nonnegative elements the algorithm converges with an exact estimate on the second iteration, and such matrices arise in applications, for example as a stochastic matrix or as the inverse of an M-matrix. The remaining columns are chosen as $\mathrm{rand}\{-1, 1\}$, with a check for and correction of parallel columns, exactly as for S in the body of the algorithm. We choose random vectors because it is difficult to argue for any particular fixed vectors and because randomness lessens the importance of counterexamples (see the comments in the next section).

Next, we consider complex matrices, which arise in the pseudospectrum application of section 4. Everything in this section remains valid for complex matrices provided that $\mathrm{sign}(A)$ is redefined as the matrix $(a_{ij}/|a_{ij}|)$ (and $\mathrm{sign}(0) = 1$) and

transposes are replaced by conjugate transposes. The matrix S is now complex with elements of unit modulus and we are much less likely to find parallel columns of S from one iteration to the next or within the current S. Therefore for complex matrices we omit the tests for parallel columns. However, we take the same, real, starting matrix. There is one further question in the complex case. In the analogue of Algorithm 2.1 for complex matrices in [14, Alg. 5.1] z is defined as $z = \mathrm{Re}(A^*\xi)$, based on subgradient considerations. In our block algorithm should we take $Z = A^*S$ or $Z = \mathrm{Re}(A^*S)$? The former can be justified from heuristic considerations and preserves more information about A. We return to this question in the next section.

The motivation for Algorithm 2.4 is to enable more accurate and reliable estimates to be obtained than are provided by the 1-norm power method. The question arises of how the accuracy and reliability of the estimates varies with t. Little can be said theoretically because, unlike the approach of [12] mentioned at the start of this section, the estimates are not monotonic in t. If we run Algorithm 2.4 for t_1 and for $t_2 > t_1$, using a common set of t_1 starting vectors, we can obtain a smaller estimate for t_2 than for t_1, because a less promising choice of unit vector e_j can turn out to be better than a more promising choice made with more available information. Nonmonotonicity is unlikely, however, and we argue that it is a price worth paying for the other advantages that accrue. In the next section we investigate the behavior of Algorithm 2.4 empirically.

3. Numerical experiments. Our aim in this section is to answer the following questions about Algorithm 2.4, bearing in mind that for $t = 1$ the algorithm is an implementation of the well-understood 1-norm power method.

1. How does the accuracy and reliability of the norm estimates vary with t?
2. How good are the norm estimates in general?
3. How does the number of iterations behave for $t > 1$?

Note that we are not searching for counterexamples, as was done for previous condition estimators [4], [5], [14]. We know that for a fixed starting matrix and any $t \ll n$ there must be families of matrices whose norm is underestimated by an arbitrarily large factor, since the algorithm samples the behavior of the $n \times n$ matrix A on fewer than n vectors. But since the algorithm uses a random starting matrix for $t > 1$, each counterexample will be valid only for particular starting matrices.

All our tests have been performed with MATLAB.

Our first group of tests deals with random real matrices. Amongst the matrices A we used were the following:

1. A from the normal $N(0,1)$ distribution (denoted `randn`) and its inverse, orthogonal QR factor, upper triangular part, and inverse of the upper triangular part.
2. A from the uniform distribution on the set $\{-1, 0, 1\}$ (denoted `rand({-1,0,1})`), A^{-1} from the uniform distribution on the interval $[0, 1]$.
3. A and A^{-1} of the form $U\Sigma V^T$ where U and V are random orthogonal matrices and $\Sigma = \mathrm{diag}(\sigma_i)$, with the singular values σ_i distributed exponentially, arithmetically, or with all except the smallest equal to unity, and with 2-norm condition number ranging from 1 to 10^{16}.

Note that we omit, for example, matrices from the uniform $[0, 1]$ and uniform $\{-1, 1\}$ distributions, because for such matrices Algorithm 2.4 is easily seen to produce the exact norm for all t.

We chose n and t in the range $25 \le n \le 250$, $1 \le t \le 10$.

For each test matrix we recorded a variety of statistics including the underestimation ratio $\text{est}/\|A\|_1 \le 1$, averaged and minimized over each type of A for fixed n and t, the relative error $|\text{est} - \|A\|_1|/\|A\|_1$, and the number of iterations. We declared an estimate exact if the relative error was no larger than 10^{-14} (the unit roundoff is of order 10^{-16}). For a given matrix A we first generated a starting matrix X_1 with $t_{\max}$ columns, where $t_{\max}$ is the largest value of t to be used, and then ran Algorithm 2.4 for $t = 1\colon t_{\max}$ using starting matrix $X_1(:, 1\colon t)$. In this way we could see the effect of increasing t. In particular, we checked what percentage of the estimates for a given t were at least as large as the estimates for all smaller t; we denote this "improve%." We set itmax = 5 in Algorithm 2.4.

First, we give some general comments on the results.

1. Increasing t usually gave larger average and minimum underestimation ratios, though there were exceptions. The quantity improve% was 100 about half the time and never less than 78.
2. The number of iterations averaged between 2 and 3 throughout, with maxima ranging from 2 to 5 depending on the type of matrix. Thus increasing t from 1 has little effect on the number of iterations—an important fact that could not be predicted from the theory. Although there are specially constructed examples for which the 1-norm power method requires many iterations (one is described below), it is rare for the limit of 5 iterations to come into effect.
3. Throughout the tests we also computed the extra estimate (2.1) used by the LAPACK norm estimator [14]. As expected, in none of our tests (with random matrices) was the extra estimate larger than the estimate provided by Algorithm 2.4 with $t = 1$.

In Tables 3 and 4 we show detailed results for two particular types of random matrix from among those described above, with $n = 100$: `inv(randn)` and `rand({-1,0,1})`. The columns headed "Products" show the average and maximum total number of matrix products AX and A^TS. In each case 5000 matrices were used. For the matrices `inv(randn)` taking $t = 2$ significantly improves the worst-case and average estimates and the proportion of exact estimates over $t = 1$, while for $t = 4$ the estimate is exact almost 98% of the time. For the matrices `rand({-1,0,1})` the improvements as t increases are less dramatic but still useful; notice that exactly four matrix products were required in every case. As well as recording the number of products, we checked how convergence was achieved. For the matrices `rand({-1,0,1})` convergence was always achieved at the test (4) in Algorithm 2.4, while for `inv(randn)` convergence was declared at tests (4) and (2) in approximately 96% and 4% of the cases, respectively (with just a few instances of convergence at (5) for $t \ge 2$). The last two columns of Table 3 show the average and maximum number of parallel columns of S that were detected. A small number of repeated e_j vectors were detected and replaced (the largest average was 0.03, occurring for $t = 2$, and the maximum number of 7 occurred for $t = 8$). No parallel columns or repeated e_j were detected for the matrices in Table 4.

The strategy of replacing parallel columns of S and repeated e_j vectors has little effect on the overall performance of Algorithm 2.4 in our tests with random matrices. Since particular examples can be found where it is beneficial (see Example 2.5) and the cost is negligible, we feel its use is worthwhile. However, we see no advantage to extending the strategy to compare the columns of S with those of all previous S matrices.

TABLE 3
Results for 5000 *matrices* `inv(randn)` *of dimension* 100.

	Underest. ratio			Products			Parallel cols.	
t	min	average	% exact	average	max	improve%	average	max
1	0.176	0.979	83.40	4.3	10		0.00	0
2	0.507	0.993	92.64	4.0	8	98.42	0.09	2
3	0.628	0.997	96.40	4.0	6	98.88	0.22	4
4	0.702	0.999	97.98	4.0	6	99.46	0.37	6
5	0.780	0.999	98.98	4.0	6	99.80	0.52	8
6	0.798	1.000	99.40	4.0	6	99.92	0.68	9
7	0.885	1.000	99.62	4.0	6	99.94	0.86	11
8	0.889	1.000	99.78	4.0	6	99.96	1.04	13
9	0.893	1.000	99.88	4.0	4	100.00	1.22	15
10	0.893	1.000	99.92	4.0	4	100.00	1.39	17

TABLE 4
Results for 5000 *matrices* `rand({-1,0,1})` *of dimension* 100.

	Underest. ratio			Products		
t	min	average	% exact	average	max	improve%
1	0.530	0.836	3.42	4	4	
2	0.588	0.883	6.80	4	4	88.62
3	0.676	0.904	10.12	4	4	87.68
4	0.708	0.917	13.00	4	4	88.08
5	0.733	0.928	16.48	4	4	88.86
6	0.743	0.935	19.24	4	4	88.70
7	0.757	0.941	22.24	4	4	89.20
8	0.761	0.946	25.60	4	4	89.88
9	0.775	0.951	28.56	4	4	90.74
10	0.775	0.956	31.64	4	4	91.44

Higham [15] gives a tridiagonal matrix for which the 1-norm power method (Algorithm 2.1) requires n iterations to converge. We have constructed a matrix for which the maximum $n+1$ iterations is required. It is the inverse of a bidiagonal matrix:

$$A_n(\alpha) = -\begin{bmatrix} 1 & \alpha & & & \\ & 1 & \alpha & & \\ & & 1 & \ddots & \\ & & & \ddots & \alpha \\ & & & & 1 \end{bmatrix}^{-1}$$
$$= -\begin{bmatrix} 1 & -\alpha & \alpha^2 & \dots & (-\alpha)^{n-1} \\ & 1 & -\alpha & & \vdots \\ & & 1 & \ddots & \vdots \\ & & & \ddots & -\alpha \\ & & & & 1 \end{bmatrix} \in \mathbb{R}^{n\times n}, \qquad 0 < \alpha < 1.$$

(The minus sign in front of the matrix is necessary!) It is straightforward to show that when Algorithm 2.1 is applied to $A_n(\alpha)$ it produces, for $k = 1\colon n$,

$$x^{(k)} = e_k, \qquad y^{(k+1)} = Ae_k, \qquad \|y^{(k+1)}\|_1 = \frac{1-\alpha^k}{1-\alpha},$$
$$\xi^{(k+1)} = [\,(-1)^k \quad (-1)^{k-1} \quad \dots \quad -1 \quad 1 \quad \dots \quad 1\,], \qquad \|z^{(k+1)}\|_\infty = |z_{k+1}^{(k+1)}|.$$

TABLE 5
Results for matrix $A_{100}(1-10^{-6})$, *with* 1000 *repetitions.*

	Underest. ratio			Products		
t	min	average	% exact	average	max	improve%
1	0.050	0.050	0.00	11.0	11*	
2	0.290	0.901	60.80	7.8	11*	100.00
3	0.510	0.975	84.90	6.5	11*	96.40
4	0.650	0.997	97.60	5.4	11*	99.60
5	0.840	0.999	99.30	4.9	11*	99.70
6	1.000	1.000	100.00	4.6	11*	100.00
7	1.000	1.000	100.00	4.3	11*	100.00
8	1.000	1.000	100.00	4.2	8	100.00
9	1.000	1.000	100.00	4.1	8	100.00
10	1.000	1.000	100.00	4.1	8	100.00

Thus every column of $A_n(\alpha)$ is computed, in order from first to last, and the exact norm is obtained. If the algorithm is terminated after p iterations, then it produces the estimate $(1-\alpha^{p-1})/(1-\alpha)$, with underestimation ratio $(1-\alpha^{p-1})/(1-\alpha^n)$. For $t = 1$, Algorithm 2.4 behaves in exactly the same way.

We applied Algorithm 2.4 1000 times to $A_{100}(1-10^{-6})$ (with itmax = 5, as in all our tests). The results are shown in Table 5; in the sixth column, "11*" denotes that convergence was declared because the iteration limit was reached (the percentage of such occurrences varied from 100% for $t = 1$ to 0.1% for $t = 7$). The underestimation ratio for $t = 1$ agrees with the theory and is unacceptably small (note that there is no randomness and hence only one estimate for $t = 1$). But for all $t \geq 2$ the average norm estimates are satisfactory. The extra estimate (2.1) has the value 0.561; thus it significantly improves the estimate for $t = 1$ but is worse than the average estimates for all greater t.

For complex matrices we have tried both $Z = A^*S$ and $Z = \mathrm{Re}(A^*S)$ at (3) in Algorithm 2.4. Tables 6 and 7 compare the two choices for 5000 100×100 random complex matrices of the form `inv(rand+i*rand)`, where `rand` is a matrix from the uniform distribution on the interval $[0, 1]$. In this test, larger underestimation ratios are obtained for $Z = A^*S$, the percentage of exact estimates is higher, and the statistics on the number of matrix products are slightly better. In other tests we have found the complex choice of Z always to perform at least as well, overall, as the real choice (see, for example, the riffle shuffle example in section 4). We therefore keep Z complex in Algorithm 2.4. In version 3.0 of LAPACK the norm estimator has been modified from that in version 2.0 to keep Z (here a vector) complex.

Finally, how does Algorithm 2.4 compare with the suggestion of Hager mentioned at the beginning of section 2 of running the 1-norm power method t times in succession? In tests with random matrices we have found Hager's approach to produce surprisingly good norm estimates, but they are generally inferior to those from Algorithm 2.4. Since Hager's approach is based entirely on level 2 BLAS operations, Algorithm 2.4 is clearly preferred.

4. Computing 1-norm pseudospectra. In this section we apply the complex version of Algorithm 2.4 to the computation of 1-norm pseudospectra. For $\epsilon \geq 0$ and any subordinate matrix norm the ϵ-pseudospectrum of $A \in \mathbb{C}^{n\times n}$ is defined by [22]

$$\Lambda_\epsilon(A) = \{\, z \in \mathbb{C} : z \text{ is an eigenvalue of } A + E \text{ for some } E \text{ with} \|E\| \leq \epsilon \,\},$$

TABLE 6
Results for 5000 *matrices* `inv(rand+i*rand)` *of dimension* 100*, with* $Z = A^*S$.

	Underest. ratio			Products		
t	min	average	% exact	average	max	improve%
1	0.456	0.980	76.04	4.2	8	
2	0.688	0.994	89.92	4.0	8	98.46
3	0.746	0.997	95.30	4.0	6	99.14
4	0.763	0.999	97.46	4.0	6	99.34
5	0.763	0.999	98.68	4.0	6	99.62
6	0.859	1.000	99.18	4.0	6	99.82
7	0.859	1.000	99.50	4.0	4	99.88
8	0.859	1.000	99.70	4.0	4	99.96
9	0.859	1.000	99.86	4.0	4	100.00
10	0.954	1.000	99.90	4.0	4	100.00

TABLE 7
Results for 5000 *matrices* `inv(rand+i*rand)` *of dimension* 100*, with* $Z = \mathrm{Re}(A^*S)$.

	Underest. ratio			Products		
t	min	average	% exact	average	max	improve%
1	0.403	0.908	36.82	4.2	8	
2	0.503	0.955	56.94	4.1	8	93.98
3	0.633	0.975	71.24	4.0	8	94.92
4	0.641	0.986	81.10	4.0	6	96.22
5	0.641	0.992	87.52	4.0	6	97.48
6	0.690	0.995	92.36	4.0	6	98.54
7	0.781	0.997	95.00	4.0	6	99.20
8	0.801	0.998	96.70	4.0	6	99.44
9	0.819	0.999	98.12	4.0	6	99.74
10	0.822	0.999	98.68	4.0	6	99.84

or, equivalently, in terms of the resolvent $(zI - A)^{-1}$, as

$$\Lambda_\epsilon(A) = \{\, z \in \mathbb{C} : \|(zI - A)^{-1}\| \geq \epsilon^{-1} \,\}.$$

Most published work on pseudospectra has dealt with the 2-norm, and the utility of 2-norm pseudospectra in revealing the effects of nonnormality is well appreciated [19], [20], [22].

The 2-norm and any other p-norm of an $n \times n$ matrix differ by a factor at most $\sqrt{n}$. For small n, pseudospectra therefore do not vary much between different p-norms. However, Jónsson and Trefethen have shown [17] that in Markov chain applications the choice of norm for pseudospectra can be crucial. In Markov chains representing a random walk on an m-dimensional hypercube and a riffle shuffle of m cards, the transition matrices are of dimension exponential in m and factorial in m, respectively. It is known that when measured in an appropriate way, these random processes converge to a steady state not gradually but suddenly after a certain number of steps. As these processes involve powers of matrices of possibly huge dimension, and as the 1-norm is the natural norm for probability,[1] 1-norm pseudospectra are an important tool for explaining the transient behavior [17].

One of the most useful graphical representations of pseudospectra is a plot of level curves of the resolvent. We therefore consider the standard approach of evaluating $\|(zI - A)^{-1}\|_1$ on an equally spaced grid of points z in some region of interest in the

[1] Because of the use of row vectors in the Markov chain literature, it is actually the ∞-norm that is relevant. By using $\|A\|_1 = \|A^T\|_\infty$ we can continue to work with the 1-norm.

complex plane and sending the results to a contour plotter. A variety of methods for carrying out these computations for the 2-norm are surveyed by Trefethen [21], but most of the ideas employed are not directly applicable to the 1-norm.

Explicitly forming $(zI - A)^{-1}$ at each grid point is computationally expensive. A more efficient approach is to factorize $P(zI - A) = LU$ at each point z by LU factorization with partial pivoting and to use Algorithm 2.4 to *estimate* $\|(zI-A)^{-1}\|_1$; the matrix multiplications in the algorithm become triangular solves with multiple right-hand sides. This approach can take advantage of sparsity in A. However, the method still requires $O(n^3)$ operations per grid point when A is full.

We consider instead a more efficient approach that is applicable when we can compute a Schur factorization [9, Chap. 7]

$$A = QTQ^*, \tag{4.1}$$

where Q is unitary and T is upper triangular. Given this factorization we have

$$zI - A = Q(zI - T)Q^*, \tag{4.2}$$

so that forming a matrix product with $(zI - A)^{-1}$ or its conjugate transpose reduces to solving a multiple right-hand side triangular system and multiplying by Q and Q^*. Given this initial decomposition the cost per grid point of estimating the resolvent norm using Algorithm 2.4 is just $O(n^2t)$ flops—a substantial saving over the first approach, and of the same order of magnitude as the cost of standard methods for computing 2-norm pseudospectra (provided t is small). In place of the Schur decomposition we could use a Hessenberg decomposition, computed either by Gauss transformations or Householder transformations [9, Sec. 7.4]; these decompositions are less expensive, but the cost per grid point is a larger multiple of n^2 flops because of the need to factorize a Hessenberg matrix.

In more detail our approach is as follows.

ALGORITHM 4.1 (1-norm pseudospectra estimation). *Given $A \in \mathbb{C}^{n\times n}$ and a positive integer t, this algorithm estimates $\|(zI - A)^{-1}\|_1$ on a specified grid of points in the complex plane, using Algorithm* 2.4 *with parameter t.*

```
Compute the Schur factorization A = QTQ*.
for each grid point z
    Apply the complex version of Algorithm 2.4 to (zI - A)^{-1}
    with parameter t, using the representation (4.2).
end
```

Our experience is that Algorithm 4.1 frequently leads to visually acceptable contour plots even for $t = 1$. As the following example shows, however, a larger value of t may be needed. We take an example from Markov chains: the Gilbert–Shannon–Reeds model of a riffle shuffle on a deck of n cards. The transition matrix P is of dimension $n!$. Remarkably, as Jónsson and Trefethen explain [17], the dimension of the matrix can be reduced to n by certain transformations that preserve the 1-norms of powers and of the resolvent. For this experiment we took $n = 52$ and, as in [17], we computed pseudospectra of the "decay matrix" $A = P - \lim_{k\to\infty} P^k$, working with the reduced form of A.

Figure 1 shows approximations to the 1-norm pseudospectra computed on a 100×100 grid, with $t = 1, 2, 3$ in Algorithm 2.4. Contours are plotted for $\epsilon = 10^{-1}, 10^{-1.5}, \ldots, 10^{-4}$, the dashed line marks the unit circle, and the eigenvalues are plotted as dots. We did not exploit the fact that, since A is real, the pseudospectra are symmetric about the real axis. The contour plot for $t = 1$ is clearly incorrect in

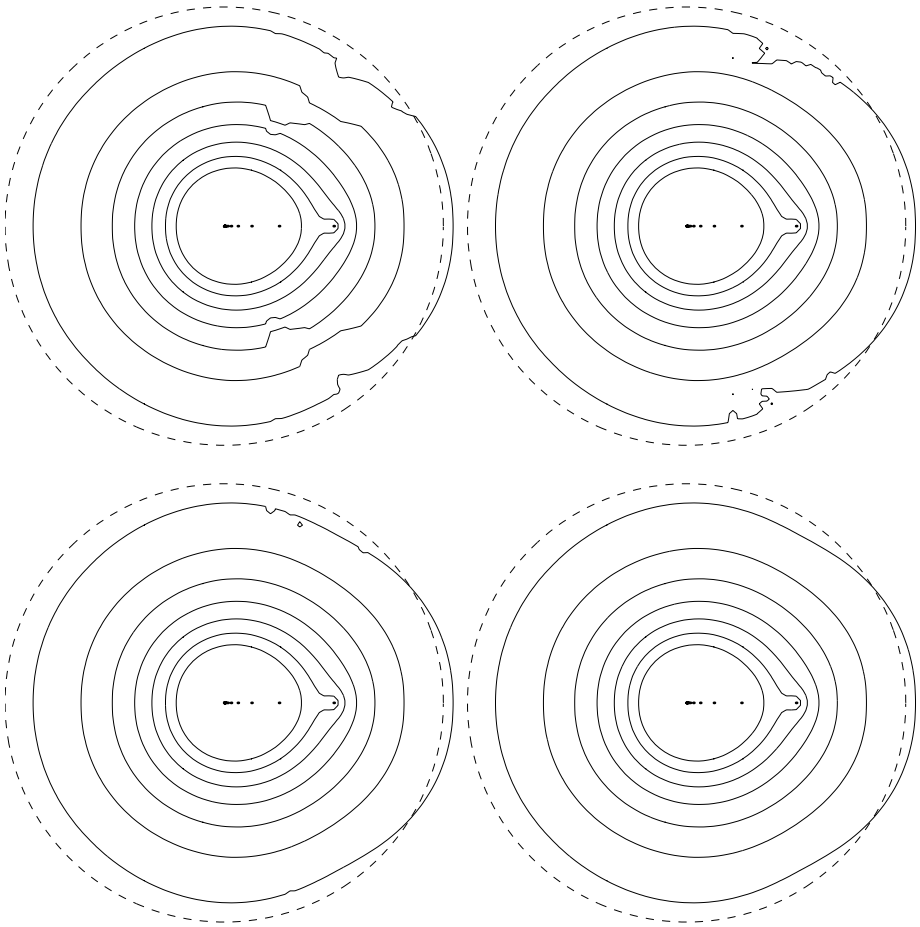

FIG. 1. 1-*norm pseudospectra for the riffle shuffle. Clockwise from top left:* $t = 1$ *and* $Z = \mathrm{Re}(A^*S)$*,* $t = 1$*,* $t = 2$*,* $t = 3$*.*

the outer contour, while $t = 2$ yields an improvement and the plot for $t = 3$ is correct to visual accuracy. Table 8 summarizes the key statistics from these computations, showing that on average the norm estimates had about t correct significant digits for $t = 1, 2, 3$. The figure and the table confirm that it is better to keep Z complex in Algorithm 2.4.

We give a further example, in which A is a spectral discretization of an integral operator of Landau [21, Sec. 21]; like the operator, A is complex and symmetric. We took dimension $n = 250$ and Fresnel number 8. As noted in [21] for the 2-norm, a fine grid is needed to resolve the details for this example; we used a 200×200 grid. Figure 2 shows the computed pseudospectra for $t = 1, 4, 8, 16$, and Table 9 summarizes the statistics for these values and for $t = 2$. Contours are plotted for $\epsilon = 10^{-1}, 10^{-1.5}, \ldots, 10^{-10}$, the dashed line marks the unit circle, and the eigenvalues are plotted as dots. In Table 9, "11^*" denotes that convergence was declared because the iteration limit was reached (the percentage of such occurrences was 0.12% for $t = 1$ and 0.025% for $t = 2$). For $t = 1$ one of the contour lines misses an eigenvalue

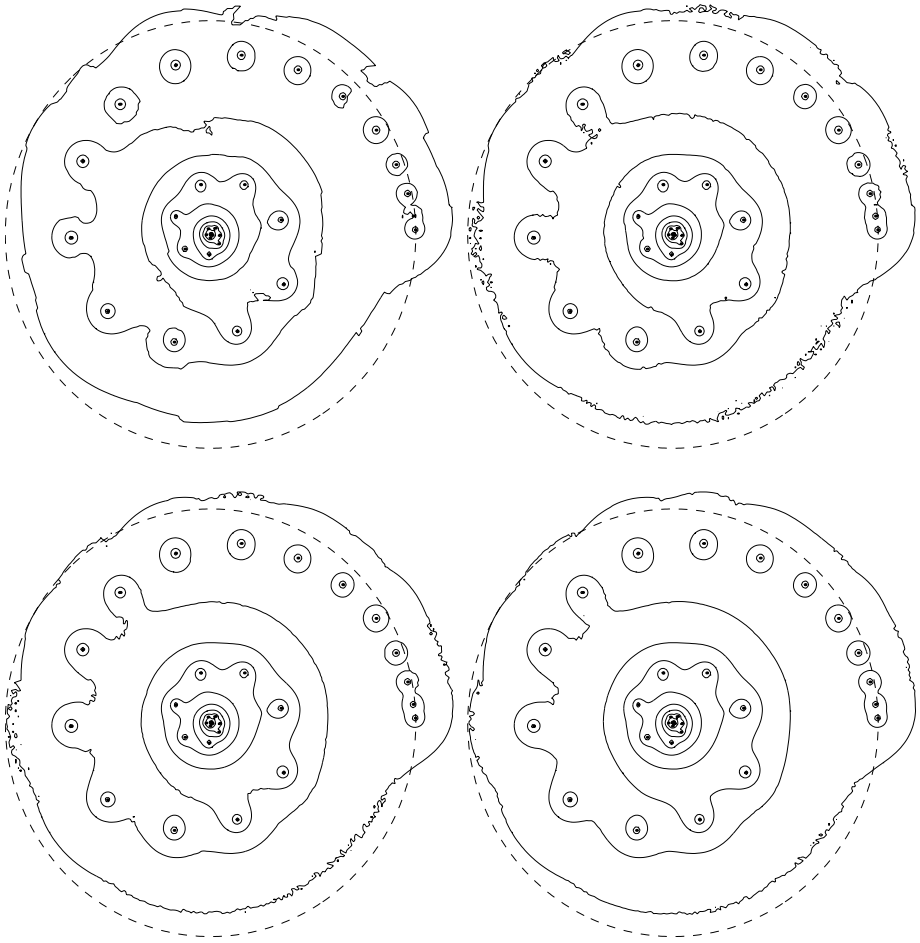

FIG. 2. 1-*norm pseudospectra for the Landau matrix. Clockwise from top left:* $t = 1$, $t = 4$, $t = 8$, $t = 16$.

in the northwest corner of the plot. For $t = 16$ the plot differs from the exact one only by some tiny oscillations in two outer contours. While Algorithm 2.4 performs well even for small t, as measured by the underestimation ratio, quite accurate values of the 1-norm of the resolvent are needed in this example in order to produce smooth contours.

5. Conclusions. We have derived a new matrix 1-norm estimation algorithm, Algorithm 2.4, with a number of key features. Most importantly, the algorithm has a parameter t that can be used to control the accuracy and reliability of the estimate (which is actually a lower bound). While there is no guarantee that increasing t increases the estimate (leaving aside the fact that the starting matrix is partly random), the estimate typically does increase with t, leading quickly to one or more correct significant digits in the estimate. A crucial property of the algorithm is that the number of iterations and matrix products required for convergence is essentially independent of t (for random matrices about two iterations are required on average,

TABLE 8
Results for riffle shuffle example. $t = 1^*$ *denotes* $t = 1$ *with* $Z = \mathrm{Re}(A^*S)$.

	Underest. ratio			Products	
t	min	average	% exact	average	max
1^*	0.464	0.963	80.06	4.5	10
1	0.486	0.978	85.90	4.1	8
2	0.621	0.989	89.88	4.1	8
3	0.686	0.996	94.30	4.0	6

TABLE 9
Results for Landau matrix example.

	Underest. ratio			Products	
t	min	average	% exact	average	max
1	0.096	0.920	13.87	4.2	11^*
2	0.511	0.954	17.95	4.1	11^*
4	0.666	0.971	29.00	4.0	10
8	0.786	0.984	43.22	4.0	8
16	0.846	0.993	61.02	4.0	6

corresponding to four products of $n \times n$ and $n \times t$ matrices). The algorithm avoids redundant computations and keeps constant the size of the matrix multiplications. In future work we intend to investigate how the choice of t affects the efficiency of the algorithm in a high-performance computing environment.

Unlike the statistically based norm estimation techniques in [7], [11] which currently apply only to real matrices, our algorithm handles both real and complex matrices.

Since our algorithm uses a partly random starting matrix for $t \geq 2$, it is natural to ask whether bounds, valid for all A, can be obtained on the probability of the estimate being within a certain factor of $\|A\|_1$. We feel that the very features of the algorithm that make it so effective make it difficult or impossible to derive useful bounds of this type.

For $t = 1$ our algorithm is very similar to the estimator in LAPACK, the differences being that our algorithm omits the extra estimate (2.1) and for real matrices we test for parallel rather than simply repeated sign vectors (which improves the efficiency). Unlike in the estimator in LAPACK 2.0, for complex matrices we do not take the real part of the z vector (which improves both the quality of the estimates and the efficiency), and this change has been incorporated into LAPACK 3.0.

The new algorithm makes an attractive replacement for the existing LAPACK estimator. The value $t = 1$ would be the natural default choice (with the extra estimate (2.1) included for extra reliability and backward compatibility) and a user willing to pay more for more accurate and reliable 1-norm estimates would have the option of choosing a larger t.

Acknowledgment. We thank Nick Trefethen for providing M-files that compute the riffle shuffle and Landau matrices used in section 4.

REFERENCES

[1] E. ANDERSON, Z. BAI, C. H. BISCHOF, S. BLACKFORD, J. W. DEMMEL, J. J. DONGARRA, J. J. DU CROZ, A. GREENBAUM, S. J. HAMMARLING, A. MCKENNEY, AND D. C. SORENSEN, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, PA, 1999.

[2] M. Arioli, J. W. Demmel, and I. S. Duff, *Solving sparse linear systems with sparse backward error*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 165–190.
[3] D. W. Boyd, *The power method for ℓ^p norms*, Linear Algebra Appl., 9 (1974), pp. 95–101.
[4] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.
[5] A. K. Cline and R. K. Rew, *A set of counter-examples to three condition number estimators*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 602–611.
[6] J. W. Demmel, *Open Problems in Numerical Linear Algebra*, IMA Preprint Series #961, Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN, 1992.
[7] J. D. Dixon, *Estimating extremal eigenvalues and condition numbers of matrices*, SIAM J. Numer. Anal., 20 (1983), pp. 812–814.
[8] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.
[9] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[10] W. B. Gragg and G. W. Stewart, *A stable variant of the secant method for solving nonlinear equations*, SIAM J. Numer. Anal., 13 (1976), pp. 889–903.
[11] T. Gudmundsson, C. S. Kenney, and A. J. Laub, *Small-sample statistical estimates for matrix norms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 776–792.
[12] W. W. Hager, *Condition estimates*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 311–316.
[13] N. J. Higham, *A survey of condition number estimation for triangular matrices*, SIAM Rev., 29 (1987), pp. 575–596.
[14] 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.
[15] N. J. Higham, *Experience with a matrix norm estimator*, SIAM J. Sci. Stat. Comput., 11 (1990), pp. 804–809.
[16] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 1996.
[17] G. F. Jónsson and L. N. Trefethen, *A numerical analyst looks at the "cutoff phenomenon" in card shuffling and other Markov chains*, in Numerical Analysis 1997, Proceedings of the 17th Dundee Biennial Conference, D. F. Griffiths, D. J. Higham, and G. A. Watson, eds., Pitman Res. Notes Math. Ser. 380, Addison Wesley Longman, Harlow, Essex, UK, 1998, pp. 150–178.
[18] P. D. Tao, *Convergence of a subgradient method for computing the bound norm of matrices*, Linear Algebra Appl., 62 (1984), pp. 163–182 (in French).
[19] L. N. Trefethen, *Pseudospectra of matrices*, in Numerical Analysis 1991, Proceedings of the 14th Dundee Conference, D. F. Griffiths and G. A. Watson, eds., Pitman Res. Notes Math. 260, Longman Scientific and Technical, Essex, UK, 1992, pp. 234–266.
[20] L. N. Trefethen, *Pseudospectra of linear operators*, SIAM Rev., 39 (1997), pp. 383–406.
[21] L. N. Trefethen, *Computation of pseudospectra*, in Acta Numerica 8, A. Iseries, ed., Cambridge University Press, Cambridge, UK, 1999, pp. 247–295.
[22] L. N. Trefethen, *Spectra and Pseudospectra: The Behavior of Non-Normal Matrices and Operators*, 1999, in preparation.