Estimating the Largest Elements of a Matrix

Higham, Nicholas J. and Relton, Samuel D.

2016

MIMS EPrint: **2015.116**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

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

ISSN 1749-9097

SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. C584–C601

ESTIMATING THE LARGEST ELEMENTS OF A MATRIX*

NICHOLAS J. HIGHAM† AND SAMUEL D. RELTON†

Abstract. We derive an algorithm for estimating the largest $p \geq 1$ values a_{ij} or $|a_{ij}|$ for an $m \times n$ matrix A, along with their locations in the matrix. The matrix is accessed using only matrix–vector or matrix–matrix products. For $p = 1$ the algorithm estimates the norm $\|A\|_M := \max_{i,j} |a_{ij}|$ or $\max_{i,j} a_{ij}$. The algorithm is based on a power method for mixed subordinate matrix norms and iterates on $n \times t$ matrices, where $t \geq p$ is a parameter. For $p = t = 1$ we show that the algorithm is essentially equivalent to rook pivoting in Gaussian elimination; we also obtain a bound for the expected number of matrix–vector products for random matrices and give a class of counterexamples. Our numerical experiments show that for $p = 1$ the algorithm usually converges in just two iterations, requiring the equivalent of $4t$ matrix–vector products, and for $t = 2$ the algorithm already provides excellent estimates that are usually within a factor 2 of the largest element and frequently exact. For $p > 1$ we incorporate deflation to improve the performance of the algorithm. Experiments on real-life datasets show that the algorithm is highly effective in practice.

Key words. matrix norm estimation, largest elements, power method, mixed subordinate norm, condition number estimation

AMS subject classification. 65F35

DOI. 10.1137/15M1053645

1. Introduction.

We are interested in estimating the matrix norm

$$\|A\|_M := \max_{i,j} |a_{ij}| \tag{1.1}$$

for $A \in \mathbb{R}^{m\times n}$. Note that the M-norm is not a consistent norm, that is, $\|AB\|_M \leq \|A\|_M\|B\|_M$ does not hold for all A and B for which the product is defined. However, $\|A\| = n\|A\|_M$ is consistent.

Calculating $\|A\|_M$ from its definition has $O(mn)$ cost, since each element of A must be inspected once. In the applications we have in mind the matrix A is known only implicitly. For example, the matrix could be given as a product $A = BC$, an inverse $A = B^{-1}$, or more generally as a function $A = f(B)$, possibly with sparse B and C, and forming A explicitly can be prohibitively expensive. We wish to estimate $\|A\|_M$ at the cost of a few matrix–vector products, each of which we assume can be computed cheaply. A particular application is estimation of condition numbers of various types [21], [22].

While $\|A\|_M$ measures the overall size of A, there are many situations in which we may wish to find the $p > 1$ largest entries of A. For instance, let A be the adjacency matrix of a graph representing interactions between entities in a system. The (i, j) entry of the matrix exponential e^A is called the communicability between nodes i and j [8], [9], and the larger the communicability is, the stronger the link between nodes i and j. To determine the $p > 1$ strongest links in the graph we wish to compute

*Submitted to the journal's Software and High-Performance Computing section December 21, 2015; accepted for publication (in revised form) August 11, 2016; published electronically October 19, 2016.

http://www.siam.org/journals/sisc/38-5/M105364.html

Funding: This work was supported by European Research Council Advanced Grant MATFUN (267526) and Engineering and Physical Sciences Research Council grant EP/I01912X/1.

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

the p largest entries of e^A without computing the entire matrix exponential, which is typically dense when A is sparse. Here we can exploit available methods for computing $e^A v$ using only matrix–vector products with A (and possibly A^*), such as those of Al-Mohy and Higham [1], Caliari et al. [5], or Frommer, Güttel, and Schweitzer [13].

The problem of finding the largest entries of a matrix product arises in a variety of data mining and information retrieval tasks. For example, the use of factor models in recommender systems leads to matrix products $B^T C$ with $B, C \in \mathbb{R}^{m\times n}$, $m \ll n$, and n very large [29]. Another application is link prediction in graphs [26].

Recently, Ballard et al. [3] have developed a graph-based sampling method for a problem called the MAD (Maximum All-pairs Dot-product) search. In this application one must find (or approximate) the largest inner product $|b_i^T c_j|$ between sets of vectors $\{\, b_i : i = 1\colon m \,\}$ and $\{\, c_j : j = 1\colon p \,\}$ where $b_i, c_j \in \mathbb{R}^n$. This is equivalent to approximating $\|B^T C\|_M$ where b_i and c_j form the columns of B and C, respectively.

Another application is to measure the sensitivity of matrix functions. The sensitivity of an element of $f(B)$ to perturbations in an element of $B \in \mathbb{R}^{n\times n}$ is given by the absolute value of a particular element of the Kronecker matrix $K \in \mathbb{R}^{n^2\times n^2}$ associated with the Fréchet derivative of f [22, sect. 3.2]. Therefore to determine the p greatest sensitivities we need to find the p largest elements in modulus of K. While it is not practical to form K for large n, we can form matrix–vector products, which are equivalent to evaluations of the Fréchet derivative of f.

The goal of this work is to develop an algorithm to estimate the p largest elements in absolute value of A using only a few matrix–vector products with A and its conjugate transpose. We derive the basic algorithm ($p = 1$), which is a special case of the power method for mixed subordinate norms, and analyze its convergence properties. We extend it to a block algorithm that iterates with $n \times t$ matrices instead of vectors, and then extend it further so that it estimates the p elements of largest absolute value for any $p \geq 1$. It turns out that with minor modifications the algorithms estimate the largest elements a_{ij} instead of their absolute values.

The basic algorithm ($p = t = 1$) has been stated previously by Gu and Miranian [15, Alg. 4] for a specific A of the form $A = B^{-T} C^T$, where it is derived as an adaptation of an algorithm of Hager [16]. It is used in [15] in the computation of a strong rank-revealing Cholesky decomposition. However, the behavior of the algorithm is not analyzed.

We organize our work as follows. In section 2 we derive the basic algorithm for computing $\|\cdot\|_M$, and in section 3 we analyze its convergence. We show that the algorithm is essentially equivalent to rook pivoting in Gaussian elimination, obtain a bound for the expected number of matrix–vector products for random matrices, give a class of counterexamples, and identify a class of matrices for which the algorithm requires $n - 1$ iterations to converge. In section 4 we design a practical, block implementation of the algorithm. In section 5 we extend the block algorithm to find the $p \geq 1$ largest elements in modulus and introduce a deflation strategy to improve the performance. In section 6 we perform a battery of numerical experiments to test the speed and reliability of our algorithm before giving some concluding remarks in section 7.

2. Algorithm derivation. The derivation of our algorithm exploits the fact that the norm $\|\cdot\|_M$ can be expressed as a mixed subordinate norm

$$\|A\|_{\alpha,\beta} = \max_{x\neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}, \tag{2.1}$$

where $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ are vector norms. In fact, it is easy to see that

$$\|A\|_M = \|A\|_{1,\infty} = \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_1}. \tag{2.2}$$

Recall the Hölder inequality on $\mathbb{R}^n$,

$$|z^T y| \le \|z\|_q \|y\|_p, \qquad p^{-1} + q^{-1} = 1, \tag{2.3}$$

where $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$. We say the vector z is dual to y in the p-norm if it has unit q-norm and

$$z^T y = \|y\|_p; \tag{2.4}$$

thus z is a unit q-norm vector that achieves equality in (2.3). We write $z \in \mathrm{dual}_p(y)$ to denote such a vector.

To estimate $\|A\|_M$ we will use an iteration for $\|A\|_{\alpha,\beta}$. The iteration was originally proposed in 1974 by Boyd [4] for $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ (possibly different) Hölder p-norms with $p \in (1,\infty)$ and, independently, by Tao in 1975 for $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ arbitrary vector norms; see [28] and the references therein. We will derive the iteration for $\|A\|_{1,\infty}$ from first principles, as the derivation provides some insight. Our problem is

$$\max F(x) := \|Ax\|_\infty \quad \text{subject to} \quad x \in S := \{\, x : \|x\|_1 \le 1 \,\}.$$

Since F is a convex function and S is a convex set, for any $u \in S$ there is at least one vector g for which

$$F(v) \ge F(u) + g^T(v-u) \quad \text{for all} \quad v \in S. \tag{2.5}$$

Such vectors g are called subgradients of F, the set of which is denoted by ∂F; see, for example, [10, p. 364], [27, sect. 3.3]. In view of (2.5), we will choose a subgradient g and move from u to a point $v_* \in S$ that maximizes $g^T(v-u)$, that is, a vector that maximizes $g^T v$ subject to $\|v\|_1 \le 1$. Clearly all such vectors v can be written as $v \in \mathrm{dual}_\infty(g)$. When $\|g\|_\infty = |g_j|$ is the unique largest element of g then $v = \mathrm{sign}(g_j)e_j$, where e_j is the jth unit vector and

$$\mathrm{sign}(x) = \begin{cases} 1, & x \ge 0, \\ -1, & x < 0. \end{cases}$$

However, if $\|g\|_\infty = |g_i| = |g_j|$ for $i \neq j$ then $\mathrm{dual}_\infty(g)$ contains other vectors too. For example, if $g = [1,\ 1]^T$ then we also have $v = [0.5,\ 0.5]^T \in \mathrm{dual}_\infty(g)$. We therefore know that

$$\mathrm{dual}_\infty(g) \supseteq \{\, \mathrm{sign}(g_j)e_j : |g_j| = \|g\|_\infty \,\}.$$

It is straightforward to show that

$$\partial F = \partial \|Ax\|_\infty \supseteq A^T \mathrm{dual}_\infty(Ax). \tag{2.6}$$

Indeed, let $w \in A^T\mathrm{dual}_\infty(Ax)$ so that $w = A^T d$ for $d \in \mathrm{dual}_\infty(Ax)$ with $d^TAx = \|Ax\|_\infty$ and $\|d\|_1 = 1$, by definition. Then for all y we have

$$\begin{aligned} \|Ax\|_\infty + w^T(y-x) &= \|Ax\|_\infty + d^TAy - d^TAx \\ &= d^TAy \\ &\le \|d\|_1 \|Ay\|_\infty \\ &= \|Ay\|_\infty. \end{aligned}$$

Hence we have shown that $w \in \partial\|Ax\|_\infty$. This means that in order to maximize $F(x)$ we can iteratively select a subgradient $g \in A^T \operatorname{dual}_\infty(Ax)$ and a point $x \in \operatorname{dual}_\infty(g)$.

We therefore obtain the following algorithm, which is a special case of the algorithm of Tao, but with added convergence tests. Let $e = [1, 1, \ldots, 1]^T$.

ALGORITHM 2.1 (power method). *Given $A \in \mathbb{R}^{m\times n}$, this algorithm computes γ and x such that $\gamma \le \|A\|_M$ and $\|Ax\|_\infty = \gamma\|x\|_1$.*

```
1  x = n^{-1} e
2  for k = 1, 2, ...
3        y = Ax
4        if k > 1
5           if ||y||_∞ ≤ ||g||_∞, γ = ||g||_∞, quit, end
6        end
7        g = A^T e_i, where |y_i| = ||y||_∞ (smallest such i)
8        if ||g||_∞ ≤ ||y||_∞, γ = ||y||_∞, quit, end
9        x = e_j, where |g_j| = ||g||_∞ (smallest such j)
10 end
```

The cost of the algorithm is $2k - 1$ or $2k$ matrix–vector products, where k is the number of iterations.

We mention two important extensions of the algorithm that apply throughout the rest of this work. First, in some applications it is the maximum of the a_{ij} that is required rather than the maximum of the absolute values (of course these are equivalent if A is nonnegative, as in the graph application mentioned in section 1). To estimate $\max_{i,j} a_{ij}$ all we need to do is to replace all occurrences of $\|\cdot\|_\infty \equiv \max|\cdot|$ by $\max(\cdot)$ in Algorithm 2.1.

Second, we have not yet mentioned how to deal with complex matrices $A \in \mathbb{C}^{m\times n}$. Just as for the 1-norm estimators in [19, 23], we simply replace $g = A^T e_i$ in line 7 with $g = A^* e_i$, where A^* denotes the conjugate transpose.

3. Convergence properties. Now we analyze the convergence properties of Algorithm 2.1. The following lemma provides insight into the behavior of the algorithm. We denote by x^k, y^k, and g^k the vectors on the kth iteration in lines 1–8, with x^{k+1} assigned on line 9.

THEOREM 3.1. *In Algorithm 2.1, the vectors from the kth iteration satisfy*

(a) $|{g^k}^T x^k| = \|y^k\|_\infty$, *and*

(b) $\|y^k\|_\infty \le \|g^k\|_\infty \le \|y^{k+1}\|_\infty \le \|A\|_M$.

Proof. We have, for some $i = i_k$, ${g^k}^T x^k = e_i^T A x^k = e_i^T y^k = \pm\|y^k\|_\infty$. It therefore follows that

$$\begin{aligned}
\|y^k\|_\infty &= |{g^k}^T x^k| \\
&\le \|g^k\|_\infty \|x^k\|_1 = \|g^k\|_\infty = \left|{g^k}^T x^{k+1}\right| = \left|e_i^T A x^{k+1}\right| \\
&\le \|Ax^{k+1}\|_\infty = \|y^{k+1}\|_\infty \\
&\le \|A\|_M.
\end{aligned}$$

∎

Theorem 3.1 shows that the algorithm generates a nondecreasing sequence of lower bounds $\|y^1\|_\infty, \|g^1\|_\infty, \|y^2\|_\infty, \ldots$ for $\|A\|_M$, which implies that the algorithm terminates as soon as the sequence fails to increase. The algorithm converges in at most $\min(m, n) + 1$ iterations, because it samples a different column on line 3 (for $k > 1$) and row on line 7 on each iteration, so if iteration number $\min(m, n) + 1$ is

reached then after line 3 the whole matrix has been sampled columnwise (if $m \geq n$) or row-wise (if $m \leq n$). The same conclusions hold for the variants of Algorithm 2.1 mentioned in the previous section for complex A and for estimating $\max_{i,j} a_{ij}$, for which analogues of Theorem 3.1 are easily obtained.

The kth iteration of Algorithm 2.1 can be rewritten as

$$\left.\begin{array}{l} i_k = \min\{\, i\colon |a_{i,j_{k-1}}| = \max_{1\leq p\leq m} |a_{p,j_{k-1}}|\,\} \\ \text{if } |a_{i_k,j_{k-1}}| = |a_{i_{k-1},j_{k-1}}|,\ \gamma = |a_{i_k,j_{k-1}}|, \text{ quit, end} \end{array}\right\} \text{for } k \geq 2$$

$$\begin{array}{l} j_k = \min\{\, j\colon |a_{i_k,j}| = \max_{1\leq p\leq n} |a_{i_k,p}|\,\} \\ \text{if } |a_{i_k,j_k}| = |a_{i_k,j_{k-1}}|,\ \gamma = |a_{i_k,j_k}|, \text{ quit, end} \end{array}$$

This representation of the algorithm makes it clear that from the first invocation of line 7 onwards it alternately searches rows and columns until an element is found that is the largest in modulus in both its row and its column. These are precisely the steps that rook pivoting takes in Gaussian elimination! There are three differences, however, one practical and two conceptual.

1. Algorithm 2.1 starts with $x = n^{-1}e$, whereas rook pivoting always begins by searching the first column of the matrix for a largest element in modulus.
2. In rook pivoting it is assumed that the elements a_{ij} are directly addressable. We are assuming only that matrix–vector products with A and A^* can be computed, so we need to express the algorithm as in Algorithm 2.1.
3. The measure of success of rook pivoting is the growth factor for Gaussian elimination and not how good an approximation to $\|A\|_M$ is produced.

In view of the connection with rook pivoting we can apply a result of Foster [11], [12] in order to determine the expected number of matrix–vector products for a class of random matrices.

THEOREM 3.2. *Suppose the elements of $A \in \mathbb{R}^{m\times n}$ are independently and identically distributed random variables from any continuous probability distribution. Then the expected number of matrix–vector products in Algorithm* 2.1 *is less than* $1 + \mathrm{e}$.

Proof. When $m = n$ the result is a direct application of [11, Thm. 5] (see the penultimate sentence of the proof) to Algorithm 2.1 from line 7 of the first iteration onwards. For $m \neq n$ we can simply embed A in a $p \times p$ matrix where $p = \max(m, n)$, setting the extra rows/columns to zero, since the expected number of matrix–vector products is independent of the matrix dimension. ∎

The important message of Theorem 3.2 is that for matrices with independent random elements the expected number of matrix–vector products is bounded by a small constant independent of the matrix dimensions.

In one special case more can be said about convergence. If $A = bc^T$ is a rank-1 matrix then convergence is obtained on the first iteration if $c = \alpha e$ for some α and otherwise on the second iteration at line 5.

A counterexample to a matrix norm estimator is a parametrized matrix for which the estimate differs from the true norm by a factor that can be made arbitrarily large. Counterexamples to Algorithm 2.1 can be expected to exist for two reasons: the derivation allows for convergence to a local rather than a global maximum and the algorithm samples only a subset of the elements of A. We now identify a class of counterexamples, using a construction similar to that in [19].

LEMMA 3.3. *For the matrix $A(\theta) = I + \theta C \in \mathbb{R}^{n\times n}$, where $\theta \in \mathbb{R}$ and C is such that $Ce = C^Te = 0$ and $Ce_1 = C^Te_1 = 0$, Algorithm* 2.1 *returns $\gamma = 1$.*

Proof. Stepping through the first iteration, on line 3 of Algorithm 2.1 we have $y = x$ since $Ce = 0$ and therefore $g = A^T e_1 = e_1$ on line 7. This means that we obtain $x = e_1$ on line 9. On the second iteration we obtain $y = e_1$ on line 3 and the algorithm terminates with $\gamma = \|g\|_\infty = \|y\|_\infty = 1$ on line 5. ∎

Since $\|A(\theta)\|_M \approx \theta$ for large θ, assuming $\|C\|_M \approx 1$, the estimate $\gamma = 1$ returned by Algorithm 2.1 is about a factor $|\theta|$ too small.

A class of examples can be also constructed for which Algorithm 2.1 requires exactly $n - 1$ iterations to converge. For the 5×5 matrix

$$T_5(6) = \begin{bmatrix} 1 & -4 & 1 & 1 & 1 \\ 3 & -6 & 1 & 1 & 1 \\ 1 & 9 & -12 & 1 & 1 \\ 1 & 1 & 15 & -18 & 1 \\ 1 & 1 & 1 & 21 & -24 \end{bmatrix} \tag{3.1}$$

it is easy to see that the algorithm searches the rows/columns sequentially from first to last and the corresponding values of $\|y\|_\infty$ and $\|g\|_\infty$ increase monotonically until the maximum of 24 is reached. This matrix is $T_5(6)$ belonging to the class given in the next result. This class is analogous to, though completely different from, the class given by Higham [20] for which the basic 1-norm power method that underlies the LAPACK norm estimator requires n iterations.

THEOREM 3.4. *Let $T_n(\alpha) \in \mathbb{R}^{n\times n}$, where $n \geq 3$ and $\alpha \in \mathbb{R}$ is nonzero, be defined by*

$$\begin{aligned} t_{ii} &= -\alpha(i-1), \quad i > 1, \\ t_{i+1,i} &= \alpha(i-1) + \alpha/2, \quad i \geq 1, \\ t_{12} &= -\alpha(n-1)/(2n-4), \\ t_{ij} &= \alpha/(2n-4) \quad \textit{elsewhere.} \end{aligned}$$

Then Algorithm 2.1 *converges to $\|T_n(\alpha)\|_M$ in exactly $n - 1$ steps.*

Proof. One can easily check that the sum of each row of $T_n(\alpha)$ is zero, meaning that in the first iteration of the algorithm we have $y = 0$, and hence $i = 1$. Therefore $g = A^T e_1$, the first row of the matrix, from which we can clearly see that the largest element is the second, meaning that $j = 2$ and $x = e_2$ as we start the second iteration. This time the largest element of $y = Ae_2$ is the third, and so $i = 3$ and $g = A^T e_3$ selects the third row. The largest element of this row is also the third, and so $j = 3$ and $x = e_3$ as we start the third iteration. This pattern continues until on the $(n-1)$st iteration $y = Ae_1$ and $g = A^T e_n$, at which point $\|g\|_\infty = \|y\|_\infty$ and the iteration terminates, with $\gamma = \|y\|_\infty = \|A\|_M$. ∎

Now that we have a better understanding of the theoretical properties of this algorithm we can design a practical implementation.

4. A block algorithm. We can increase the accuracy and practical efficiency of Algorithm 2.1 by developing a block version, where the iterates are now $n \times t$ matrices and each iteration updates the t column vectors concurrently. This improves in two ways upon naively running the algorithm t times with different starting vectors. First, it permits the use of level 3 BLAS operations when A is dense (or the corresponding level 3 routines in the Intel Math Kernel Library Sparse BLAS or the NVIDIA CuBLAS when A is sparse), which utilize computational resources more

efficiently. Second, the t different starting vectors can "communicate" within each iteration, which should lead to better estimates.

To obtain the increased performance there are a number of issues to consider regarding the communication between the t vectors and the careful tracking of their states. For instance, a situation may occur when one column is assigned a vector that has already been assigned to another column in a previous iteration, resulting in redundant computation. We can avoid this, and increase the chance of finding the global maximum in (2.2), by replacing all such repeated vectors with randomly generated ones. Since the maximum in (2.2) is attained for some unit vector e_k, we will replace repeated columns with random unit vectors.

Since our starting point x in Algorithm 2.1 is now an $n \times t$ matrix, we need to choose t initial columns. The first will be $n^{-1}e$, as in Algorithm 2.1. For the second column we take $b \in \mathbb{R}^n$ given by

$$b_i = (-1)^{i+1}\left(1 + \frac{i-1}{n-1}\right), \tag{4.1}$$

rescaled to have unit 1-norm ($\|b\|_1 = 3n/2$). This vector is suggested by Higham [19] as being likely to avoid cancellation in the product Ab when large elements of A are hidden by cancellation in the product Ae. The remaining columns are chosen as random unit vectors, as above. This is in contrast to Higham and Tisseur [23], who, in a matrix 1-norm estimator, use random vectors with elements selected from $\{-1, 1\}$ with equal probability for all but the first column, which is $n^{-1}e$. Experiments suggest that our approach is better for the M-norm, though the difference is small.

We note that when the columns traverse distinct vertices throughout the algorithm (so that none is replaced with a random column) our implementation is deterministic for $t = 1$ and $t = 2$. Whether or not this happens in practice depends on the matrix in question.

The final consideration is the stopping criterion for the algorithm. Since we are only interested in the single largest element, here we can (as for the unblocked algorithm) terminate the algorithm once the new estimate of $\|A\|_M$ does not exceed the previous one. We also choose to terminate if all the t columns intended to start the next iteration have been encountered in a previous iteration.

These considerations lead to the following algorithm. We include a limit itmax on the number of iterations since, although no more than $\min(m, n)/t + 1$ iterations are required, in practice we want to perform no more than a small, constant number of iterations.

ALGORITHM 4.1. *Given $A \in \mathbb{R}^{m\times n}$ and integers $t > 0$ and* itmax > 0*, this algorithm computes $\gamma \in \mathbb{R}$ and indices i and j such that $\gamma = |a_{ij}| \le \|A\|_M$. The algorithm uses the function* argmax *defined by*

$[\mu, \text{ind}] = \text{argmax}(w)$
$\mu = \|w\|_\infty$
ind = smallest i for which $|w_i| = \|w\|_\infty$

1 $\gamma = 0$
2 Let $X = [n^{-1}e,\ (2/(3n))b,\ \widetilde{X}] \in \mathbb{R}^{n\times t}$, where b is given by (4.1) and the $t - 2$ columns of $\widetilde{X}$ are distinct random unit vectors.
3 Set prev_ind to contain the locations of the nonzero rows in $\widetilde{X}$.
4 for it = 1: itmax
5 $\quad Y = AX$

```
6       [μ_k, indy_k] = argmax(Y(:,k)), k = 1:t
7       [y_max, i] = argmax(μ)
8       if it = 1 and t > 2
          % Find the largest entry of AX̃.
9         [γ, i] = argmax(μ(3:t))
10        j = indy_{2+i}
11      elseif it > 1 and y_max > γ
12            γ = y_max
13            Set j to the position of the nonzero in the ith column of X.
14      else
15            quit
16      end
17      Construct W ∈ R^{m×t} such that the (indy_k, k) element equals 1,
        for k = 1:t, and all other elements are 0.
18      Z = A^T W
19      [ψ_k, indz_k] = argmax(Z(:,k)), k = 1:t
20      z_max = ‖ψ‖_∞
21      if it > 1
22        if z_max ≤ y_max, quit, end
23        if indz is a subset of prev_ind, quit, end
24        Replace any entries of indz that are present in prev_ind by
          random integers from {1,...,n} \ (indz ∪ prev_ind).
25        Add the t indices in indz to prev_ind.
26      end
27      Construct X ∈ R^{n×t} such that the (indz_k, k) elements equal 1,
        for k = 1:t, and all other elements are 0.
28  end
```

Some further comments on this algorithm are in order. First, it is clear that γ^k, the estimate of $\|A\|_M$ on the kth iteration, is a nondecreasing sequence of real numbers bounded above by $\|A\|_M$. However, due to the replacement of repeated vectors by random ones, part (b) of Theorem 3.1 no longer holds. Second, if we set $t = 1$ and ignore the random replacement of vectors, and the test for whether it > 1 on line 21, then Algorithm 2.1 is recovered.

5. Algorithm for p largest elements. In this section we describe how, with some minor modifications, we can extend Algorithm 4.1 to tackle the problem of estimating the largest $p \geq 1$ entries of A.

To estimate the largest p entries of A we must maintain a list of the largest p entries discovered and keep in mind that more than one of them may be found in any one column. Just as it is helpful to use $t > 1$ columns to find the largest entry of A, it may be beneficial to use $t > p$ columns to find the largest p entries.

Therefore, we introduce a scaling factor α such that we will work concurrently on $t = \text{ceil}(\alpha p)$ columns within the algorithm. How to choose α is an important question. Our numerical experiments in section 6 test a variety of choices for α.

Making these changes we arrive at the following algorithm.

ALGORITHM 5.1. *Given $A \in \mathbb{R}^{m\times n}$, integers $p \geq 1$ and* itmax > 0*, and a real number $\alpha \geq 1$, this algorithm computes $\gamma \in \mathbb{R}^p$ and the distinct pairs of indices (i_k, j_k) such that $\gamma_p \leq \gamma_{p-1} \leq \cdots \leq \gamma_1 \leq \|A\|_M$ and $|a_{i_k,j_k}| = \gamma_k$ for $k = 1\colon p$. This algorithm uses the function* argmax *defined by*

$[\mu, \mathrm{ind}, \mathrm{colnum}] = \mathrm{argmax}(W, t)$
Let the t largest elements of W be
$\|W(:)\|_\infty = |w_{\mathrm{ind}_1,\mathrm{colnum}_1}| = \mu_1 \geq \cdots \geq |w_{\mathrm{ind}_t,\mathrm{colnum}_t}| = \mu_t$,
where in the case of ties elements with smaller column indices
than row indices are taken first.

1 $\gamma = 0 \in \mathbb{R}^p$, $i = 0 \in \mathbb{R}^p$, $j = 0 \in \mathbb{R}^p$
2 $t = \mathrm{ceil}(\alpha p)$
3 Let $X = [n^{-1}e,\ (2/(3n))b,\ \widetilde{X}] \in \mathbb{R}^{n\times t}$, where b is given by (4.1) and the
$t-2$ columns of $\widetilde{X}$ are distinct random unit vectors.
4 Set prev_ind to contain the locations of the nonzero rows in $\widetilde{X}$.
5 for it $= 1\colon$ itmax
6 $Y = AX$
7 $[\,\mathrm{ynorms}, \mathrm{indy}, \mathrm{colnum}\,] = \mathrm{argmax}(Y, t)$
8 if it $= 1$ and $t > 2$
9 % Find the largest entries of $A\widetilde{X}$.
10 $[\gamma, i, j] = \mathrm{argmax}(Y(:, 3\colon t), p)$
11 elseif it > 1
12 if any element of ynorms is larger than γ_p and the corresponding
indices in indy and colnum are not in i, j
% ynorms contains at least one previously unseen large element.
13 Set γ equal to the largest p elements from $\gamma \cup$ ynorms
(so that γ is sorted in descending order) and update the
indices (i_k, j_k) to match for $k = 1\colon p$.
14 else
15 quit
16 end
17 end
18 Construct $W \in \mathbb{R}^{m\times t}$ such that the (indy_k, k) element equals 1,
for $k = 1\colon t$, and all other elements are 0.
19 $Z = A^T W$
20 $[\,\mathrm{znorms}, \mathrm{indz}, \mathrm{colnum}\,] = \mathrm{argmax}(Z, t)$
21 if it > 1
22 if $\mathrm{znorms}_k \leq \mathrm{ynorms}_k$ for all $k = 1\colon t$, quit, end
23 if indz is a subset of prev_ind, quit, end
24 Replace any entries of indz that are present in prev_ind by
random integers from $\{1, \ldots, n\} \setminus (\mathrm{indz} \cup \mathrm{prev\_ind})$.
25 Add the t indices in indz to prev_ind.
26 end
27 Construct $X \in \mathbb{R}^{n\times t}$ such that the (indz_k, k) elements equal 1,
for $k = 1\colon t$, and all other elements are 0.
28 end

As with Algorithm 4.1, it is clear that each element of γ forms a nondecreasing sequence of real numbers bounded above by $\|A\|_M$ as the iteration progresses. Also, if we set $p = 1$ then Algorithm 4.1 is recovered, whilst setting $p = 1$ and $\alpha = 1$ gives Algorithm 2.1 when we ignore the random replacement of vectors and the test for whether it > 1 on line 21.

A weakness of this algorithm is that all t vectors will be attracted to the entry (or entries) a_{ij} for which $|a_{ij}| = \|A\|_M$. This means that the algorithm might fail to

find the next largest entries.

To derive an improved algorithm, suppose that $|a_{ij}|$ is the largest entry found so far. We can deflate this entry and look at the matrix $A - a_{ij}e_ie_j^T$, which is a rank-1 update of A. More generally, if (i_r, j_r), $r = 1\colon p$, are the positions of the p largest entries found so far we can work with the matrix

$$A_p := A - \sum_{r=1}^{p} a_{i_r,j_r} e_{i_r} e_{j_r}^T. \tag{5.1}$$

To perform matrix–vector products with A_k we need the a_{i_r,j_r}, but these are available from the previously computed matrix–vector products with unit vectors.

Our idea is to deflate the top p elements the algorithm has encountered so far within each matrix–vector product that the algorithm performs, which means that the matrix can change with each iteration. This deflation strategy is intended to allow the algorithm to focus on finding the remaining large entries. The modified algorithm is as follows.

ALGORITHM 5.2. *Given $A \in \mathbb{R}^{m\times n}$, integers $p \geq 1$ and* itmax > 0*, and a real number $\alpha \geq 1$, this algorithm computes $\gamma \in \mathbb{R}^p$ and distinct pairs of indices (i_k, j_k) such that $\gamma_p \leq \gamma_{p-1} \leq \cdots \leq \gamma_1 \leq \|A\|_M$ and $|a_{i_kj_k}| = \gamma_k$ for $k = 1\colon p$.*

1 Run Algorithm 5.1, storing the values of a_{ij} along with $|a_{ij}|$
and replacing all matrix–vector products and transpose matrix–vector
products after line 10 in the first iteration of Algorithm 5.1 with
lines 2–5 and 6–9, respectively.
... To compute the deflated matrix–vector product $b = A_px$
2 $b = Ax$
3 for $r = 1\colon p$
4 $b_{i_r} = b_{i_r} - a_{i_rj_r}x_{j_r}$
5 end
... To compute the deflated transpose matrix–vector product $z = A_p^Ty$
6 $z = A^Ty$
7 for $r = 1\colon p$
8 $z_{j_r} = z_{j_r} - a_{i_rj_r}y_{i_r}$
9 end

6. Numerical experiments. We now describe a number of computational experiments designed to identify the strengths and weaknesses of the algorithms. All the experiments were performed in MATLAB 2014b on a dual-core laptop with an Intel i7-2620M 2.7GHz processor and 8GB of RAM, running Ubuntu 13.10.

Throughout this section we set the iteration limit itmax = 20, in order to show that in the vast majority of cases only a handful of iterations are required.

Our first set of experiments focuses on approximating $\|A\|_M$. We examine the distribution of the underestimation ratio,

$$\psi = \frac{\gamma}{\|A\|_M} \in [0, 1], \tag{6.1}$$

and the number of iterations π, which satisfies $\pi \geq 2$, over various types of random matrices. We present the following statistics for these two quantities:

- $\psi_{\min}$: the minimal value of ψ encountered.
- ψ_{avg}: the mean value of ψ encountered.
- % exact: the percentage of cases where $\psi = 1$.

- π_{avg}: the mean value of π encountered.
- $\pi_{\max}$: the maximal value of π encountered.
- % improve: the percentage of cases where the estimate for t improved from that for $t-1$.

Our second set of experiments attempts to approximate the $p > 1$ largest entries of the test matrices. To compare the ranking of the $p > 1$ elements estimated by Algorithm 5.1 to the true values we need to use some techniques from statistics. In particular we need to compare the list of the exact top p elements against the list of the estimated top p elements, both of which are partial lists since we do not rank all entries of the matrix. There are numerous ways to do this, but we have chosen to use a weighted footrule measure [24, pp. 207–209]. This gives us a similarity function ϕ such that $\phi = 1$ means that the lists are identical and $\phi = 0$ means that the lists are completely disjoint (that is, none of the top p elements was found).[1] The similarity function is weighted so that disagreements between the two lists in the first few elements are more important than disagreements further down.

We will also measure the mean underestimation ratio between the two lists. Denoting the top p elements by $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_p$ and our estimates by $\widehat{\ell}_1 \geq \widehat{\ell}_2 \geq \cdots \geq \widehat{\ell}_p$, the mean underestimation ratio is

$$\Psi = \frac{1}{p} \sum_{i=1}^{p} \frac{\widehat{\ell}(i)}{\ell(i)}. \tag{6.2}$$

It is is easy to see that $\Psi \leq 1$ and that a value of $\Psi = 1$ implies that all the top p elements were estimated correctly. Including both measures is important because we will see that in some cases, particularly those where the largest elements of the matrix have very similar magnitude, although $\phi(\ell, \widehat{\ell})$ is small (meaning that not all of the top p elements were found) we can have high values of Ψ and the estimates are still close to optimal. Finally, we will also measure the number of top p elements that are estimated exactly, denoted by η.

We test our algorithms on various types of random matrix, similar to those used to test the block 1-norm estimator [23]. Let $N(0,1)$ and $U(0,1)$ denote random variables distributed normally (with zero mean and unit variance) and uniformly (between 0 and 1 inclusive), respectively. We use four types of random matrices.

1. `randn`: the matrix with $N(0,1)$ distributed elements.
2. `inv(randn)`: the inverse of a matrix with $N(0,1)$ distributed elements.
3. `inv(randc)`: the inverse of a matrix with $N(0,1) + \mathrm{i}U(0,1)$ distributed elements, where i is the imaginary unit. For these complex matrices, transpose is replaced by conjugate transpose in the algorithms, as described in section 2.
4. `randmult`: the product of a matrix with $N(0,1)$ elements with a matrix with $U(0,1)$ elements.

One type of matrix used by Higham and Tisseur [23] to test the block 1-norm estimator has been omitted: the matrix with elements chosen randomly from the set $\{-1, 0, 1\}$. This is because our algorithms would find one of the true maximal elements as soon as a column containing ± 1 is encountered. The `randn` and `randmult` matrices both satisfy the criteria of Theorem 3.2 and should therefore require less than $1 + \mathrm{e} \approx 3.72$ iterations to converge on average. Indeed, in section 6.1 we will see that the average number of iterations is between 2.0 and 2.2 when $p = 1$.

What are considered acceptable values of ψ and Ψ? For norm estimates used in

[1] The algorithm in [24, pp. 207–209] actually computes $1 - \phi$.

TABLE 6.1
Statistics for the underestimation ratio ψ and the number of iterations π for Algorithm 4.1, for 1000 matrices of type `randn` *with dimension 100.*

t	$\psi_{\min}$	ψ_{avg}	% exact	π_{avg}	$\pi_{\max}$	% improve
1	0.4767	0.7708	3.1	2.145	4	0
2	0.5529	0.8218	6.0	2.162	5	50.0
3	0.5574	0.8561	9.9	2.149	5	51.7
4	0.5761	0.8884	12.4	2.128	4	51.4
5	0.5833	0.8884	15.9	2.133	4	51.4
6	0.5833	0.9021	18.6	2.122	5	50.8
7	0.5967	0.9102	21.4	2.125	4	53.5
8	0.6711	0.9162	23.7	2.110	4	52.3
9	0.5810	0.9257	27.3	2.135	5	53.0
10	0.5989	0.9301	29.3	2.119	5	55.9

TABLE 6.2
Statistics for the underestimation ratio ψ and the number of iterations π for Algorithm 4.1, for 1000 matrices of type `inv(randn)` *with dimension 100.*

t	$\psi_{\min}$	ψ_{avg}	% exact	π_{avg}	$\pi_{\max}$	% improve
1	0.1646	0.9625	82.0	2.187	5	0
2	0.5219	0.9902	92.0	2.103	4	92.3
3	0.5288	0.9961	95.4	2.078	4	95.4
4	0.7424	0.9982	97.8	2.043	4	97.8
5	0.8260	0.9992	98.7	2.032	3	98.7
6	0.7265	0.9986	98.2	2.028	3	98.2
7	0.8819	0.9998	99.4	2.021	3	99.4
8	0.5498	0.9992	99.3	2.013	3	99.3
9	0.8928	0.9997	99.4	2.017	4	99.4
10	0.9724	1.0000	99.9	2.008	3	99.9

condition estimation and error bounds an estimate correct to within a factor 3 (say) is adequate, since only the order of magnitude is required, so $\psi \geq 1/3$ and $\Psi \geq 1/3$ represent good performance in this context. However, we will see that the algorithms typically perform much better than this, especially on the real-life problems that we consider.

Our third, and final, set of experiments involves applying Algorithms 4.1 and 5.2 to matrices taken from some of the applications we discussed in section 1. We show how effectively our algorithms perform when used to estimate $\|A^T B\|_M$ and $\|e^A\|_M$ for large, sparse matrices. The applications and the specific matrices used are described in more detail in section 6.3.

6.1. Approximating the single largest element. We begin by investigating the performance of Algorithm 4.1 for `randn` matrices, the results for which are summarized in Table 6.1. As expected, we see that the accuracy and reliability of the algorithm steadily increase with t, as shown by the columns $\psi_{\min}$, ψ_{avg}, and % exact. Meanwhile the number of iterations averages close to 2 for all t (cf. Theorem 3.2) and does not exceed 5.

In Table 6.2 we look at the behavior of the algorithm for matrices of type `inv(randn)`. The algorithm performs extremely well on this type of matrix, with over 90% of the estimates exact for $t \geq 2$. Indeed in the majority of cases the algorithm finds the exact value of $\|A\|_M$ in only 2 iterations. Again, the reliability improves as t increases. We see that ψ rarely deviates much from the optimal value of 1 for this type of matrix.

Tables 6.3 and 6.4 show the summary statistics for complex matrices of type

TABLE 6.3
Statistics for the underestimation ratio ψ and the number of iterations π for Algorithm 4.1, for 1000 complex matrices of type `inv(randc)` *with dimension 100.*

t	$\psi_{\min}$	ψ_{avg}	% exact	π_{avg}	$\pi_{\max}$	% improve
1	0.5039	0.9709	78.2	2.203	5	0
2	0.6728	0.9911	90.1	2.118	4	90.4
3	0.7073	0.9953	93.9	2.079	4	93.9
4	0.7688	0.9977	97.0	2.055	4	97.0
5	0.7688	0.9981	97.0	2.036	3	97.1
6	0.8305	0.9989	98.0	2.020	3	98
7	0.8185	0.9992	98.8	2.020	3	98.8
8	0.8352	0.9995	99.2	2.018	3	99.2
9	0.9248	0.9996	98.8	2.014	3	98.8
10	0.8305	0.9996	99.3	2.012	3	99.3

TABLE 6.4
Statistics for the underestimation ratio ψ and the number of iterations π for Algorithm 4.1, for 1000 matrices of type `randmult` *with dimension 500.*

t	$\psi_{\min}$	ψ_{avg}	% exact	π_{avg}	$\pi_{\max}$	% improve
1	0.8043	0.9826	64.0	2.001	3	0
2	0.8249	0.9861	69.0	2.081	4	73.6
3	0.8608	0.9892	74.8	2.096	4	78.3
4	0.8249	0.9912	77.6	2.083	4	79.8
5	0.8249	0.9920	79.8	2.080	4	82.2
6	0.8793	0.9946	84.4	2.080	4	85.6
7	0.8533	0.9943	85.6	2.083	4	86.7
8	0.8822	0.9954	86.9	2.081	6	87.7
9	0.8822	0.9965	90.2	2.084	4	90.4
10	0.8822	0.9967	91.1	2.064	4	91.3

TABLE 6.5
Statistics for the mean underestimation ratio Ψ, the number of correctly estimated entries η, the weighted footrule measure ϕ, and the number of iterations π, for 1000 `randn` *matrices of dimension 500, for Algorithm 5.1 with $p = 5$. In each case $\phi_{\min} = 0$.*

α	$\Psi_{\min}$	Ψ_{avg}	η_{avg}	ϕ_{avg}	$\phi_{\max}$	π_{avg}	$\pi_{\max}$
1	0.6960	0.8497	0.3050	0.0827	0.7854	4.4950	14
2	0.7670	0.9009	0.7630	0.1959	0.9397	5.8170	15
3	0.8007	0.9254	1.1890	0.2966	1.0000	6.1810	16
4	0.8147	0.9406	1.6080	0.3874	1.0000	6.3440	14
5	0.8197	0.9502	1.9420	0.4517	1.0000	6.3360	17
6	0.8456	0.9569	2.1510	0.5020	1.0000	6.3430	15
7	0.8103	0.9646	2.5060	0.5763	1.0000	6.1820	15
8	0.8584	0.9683	2.6830	0.6034	1.0000	6.1630	15
9	0.8457	0.9720	2.9050	0.6447	1.0000	6.0830	13
10	0.8772	0.9753	3.1220	0.6797	1.0000	6.0300	14

`inv(randc)` and matrices of type `randmult`, respectively. The behavior is very similar to that of Table 6.2 for the `inv(randn)` matrices in that ψ is close to 1 in all cases and typically only 2 iterations are required, regardless of t.

6.2. Approximating the $p = 5$ largest elements. In this subsection we investigate the behavior of Algorithms 5.1 (no deflation) and 5.2 (with deflation) for estimating the $p = 5$ largest matrix entries as α varies between 1 and 10.

Table 6.5 shows the behavior of Algorithm 5.1 on `randn` matrices. We see that the mean underestimation ratio Ψ and the weighted footrule measure ϕ both increase with α. The values Ψ_{avg} are much better than ψ_{avg} in Table 6.1. The number of iterations

TABLE 6.6

Statistics for the mean underestimation ratio Ψ, the number of correctly estimated entries η, the weighted footrule measure ϕ, and the number of iterations π, for 1000 `randmult` *matrices of dimension* 500*, for Algorithm* 5.1 *with* $p = 5$*. In each case* $\phi_{\max} = 1$*.*

α	$\Psi_{\min}$	Ψ_{avg}	η_{avg}	$\phi_{\min}$	ϕ_{avg}	π_{avg}	$\pi_{\max}$
1	0.7008	0.9565	2.9420	0	0.7450	2.2260	6
2	0.8072	0.9805	3.7940	0	0.8893	2.3540	6
3	0.8972	0.9862	4.0260	0.1606	0.9230	2.3120	6
4	0.7922	0.9885	4.1220	0	0.9383	2.2820	6
5	0.8735	0.9916	4.2550	0	0.9503	2.2840	5
6	0.8671	0.9918	4.2370	0	0.9506	2.2350	6
7	0.9221	0.9933	4.3120	0.3762	0.9568	2.2760	5
8	0.9201	0.9934	4.2920	0.6017	0.9571	2.2340	5
9	0.8823	0.9941	4.3350	0	0.9601	2.2280	5
10	0.9026	0.9949	4.3690	0	0.9616	2.1950	5

TABLE 6.7

Statistics for the mean underestimation ratio Ψ, the number of correctly estimated entries η, the weighted footrule measure ϕ, and the number of iterations π, for 1000 `randmult` *matrices of dimension* 500*, for Algorithm* 5.2 *with* $p = 5$*. In each case* $\phi_{\max} = 1$*.*

α	$\Psi_{\min}$	Ψ_{avg}	η_{avg}	$\phi_{\min}$	ϕ_{avg}	π_{avg}	$\pi_{\max}$
1	0.8076	0.9877	3.8380	0	0.8095	3.2700	7
2	0.8425	0.9964	4.5240	0	0.9319	3.5060	6
3	0.7757	0.9982	4.7540	0	0.9652	3.4720	6
4	0.8505	0.9988	4.8360	0	0.9765	3.4520	5
5	0.8794	0.9992	4.8860	0	0.9839	3.4140	5
6	0.9436	0.9995	4.9230	0.5333	0.9883	3.3910	6
7	0.8486	0.9991	4.9030	0	0.9857	3.3590	5
8	0.9724	0.9998	4.9470	0.6017	0.9944	3.3620	6
9	0.8777	0.9994	4.9330	0	0.9903	3.3190	5
10	0.9019	0.9997	4.9560	0.5664	0.9949	3.3330	5

is also larger than when searching for the $p = 1$ largest entry of A, presumably because many of the elements of A have very similar magnitude.

Table 6.6 shows the summary statistics for finding the $p = 5$ largest elements of `randmult` matrices. In this case the algorithm performs much better than for the `randn` matrices: the values of Ψ, ϕ, and η are significantly higher, whilst π is much lower and the algorithm typically needs only 2 iterations to converge. The behavior of Algorithm 5.1 on the other types of matrices is similar.

Next we consider Algorithm 5.2, which incorporates deflation. Table 6.7 shows the results of applying the algorithm to `randmult` matrices. We see a significant improvement in the values of η and ϕ between these results and those in Table 6.6, where deflation was not used, though π is slightly larger on average. For the other types of test matrices (especially `inv(randn)` and `inv(randc)`) Algorithm 5.2 performed very well with $\Psi \approx 1$, $\eta \approx 5$, $\phi \approx 1$, and $\pi_{\text{avg}} \approx 2.5$ for all $\alpha \geq 2$.

For `randn` matrices, Algorithm 5.2 performed slightly worse than Algorithm 5.1 in terms of accuracy, though it converged in fewer iterations. It seems that the relatively slow convergence of Algorithm 5.1 on this class of matrices brings a benefit by allowing more time to explore the search space. In all other cases, Algorithm 5.2 was observed to be at least as accurate as Algorithm 5.1, with fewer iterations required, so we recommend this as the default choice for estimating the largest $p > 1$ entries of a matrix.

TABLE 6.8
The time to estimate the largest entry of A^TB using Algorithm 4.1 with parameter $t = 10$ and the underestimation ratio ψ (estimate divided by largest element).

Matrix	ψ	Time (secs) Exact	Time (secs) Algorithm 4.1
MovieLens	1	88.8	0.4
Sandia/ASIC_680k	1	1.6e4	1.0
SNAP/as-Skitter	0.96	2.4e4	3.6

6.3. Experiments on real datasets. Finally, we consider the application of Algorithms 4.1 and 5.2 on datasets taken from real applications.

Our first experiment performs a MAD search, an application we described in section 1. For this experiment we apply Algorithm 4.1 to find the largest entry of a matrix product. In particular, we test our algorithm on the matrix product A^TB where A and B, as detailed below, are selected from the University of Florida Sparse Matrix Collection [6], [7] and the MovieLens 10M dataset [17]. These matrices were used by Ballard et al. [3] to test their algorithm for the MAD problem.

- MovieLens: the MovieLens 10M dataset consists of a matrix $R \in \mathbb{R}^{m\times n}$ with entries equal to the ratings given to $m = 65,133$ movies by $n = 71,567$ users. As in [3], we compute a low-rank SVD approximation $U\Sigma V^T$ of this matrix containing the 150 largest singular values and corresponding singular vectors, before forming $A = (U\Sigma)^T$ and $B = V^T$, using the MATLAB `svds` function. We aim to find the largest element of A^TB.
- Sandia/ASIC_680k: matrix arising from a circuit simulation problem. The matrix is nonsymmetric of order $n = 682,862$ with $2,638,997$ nonzero entries. We aim to find the largest element of A^TA.
- SNAP/as-Skitter: an internet topology graph resulting in a binary, symmetric matrix of order $n = 1,696,415$ with $11,095,298$ nonzero entries. We aim to find the largest element of A^TA.

For comparison, we compute the exact maximal element of each matrix product A^TB by finding A^TBe_i for each unit vector e_i and keeping track of the largest element found so far. To make use of level-3 BLAS operations, we block together 100 such unit vectors into a matrix E and find A^TBE.

The results of this experiment can be seen in Table 6.8. Our new algorithm performs extremely well on these problems: it finds the exact largest element in the first and second cases, while finding the second largest entry in the third case and underestimating the maximum by only 4%. In [3] the MovieLens problem is found to be the most difficult of the datasets used there, and the algorithm of [3] was unable to find the largest element faster than if the entire matrix product were to be computed (though it did find the largest elements of Sandia/ASIC_680k and SNAP/as-Skitter faster than exact computation of the matrix product). Overall we obtain speedups of around 200, 16,000, and 6,600, respectively, compared with forming the entire matrix products explicitly. For the SNAP/as-Skitter matrix we found that searching for the $p = 5$ largest elements using Algorithm 5.2 returns all of them correctly. This suggests that when it is imperative that the exact largest element is found it may be advantageous to estimate the top p elements, for some $p > 1$, and take the largest of them.

Our second experiment is based on network communicability. Recall from the introduction that, given an adjacency matrix A associated with a graph, the (i, j) element of e^A is the communicability between nodes i and j. In this experiment we

TABLE 6.9
The time to estimate the $p = 10$ largest entries of e^A using Algorithm 5.2 with parameter $\alpha = 3$ and the number of correctly estimated largest entries η.

Matrix	η	Time (secs) Exact	Algorithm 5.2
SNAP/ca-AstroPh	10	237.3	1.7
SNAP/ca-CondMat	10	202.1	1.2
MUFC Twitter	10	5501.7	3.2

show how our algorithm can be used to find the $p = 10$ pairs of nodes (i, j) with the largest communicability without computing the entire matrix exponential. To compute matrix–vector products $e^A v$ we use the algorithm of Al-Mohy and Higham [1], as implemented in the MATLAB function `expmv` at http://www.mathworks.com/matlabcentral/fileexchange/29576-matrix-exponential-times-a-vector.

We test our algorithm on three matrices.

- SNAP/ca-AstroPh: records research collaborations in the field of astrophysics on Arxiv until April 2003. It is a symmetric, binary matrix of order $n = 18,772$ with $396,160$ nonzero entries [25].
- SNAP/ca-CondMat: records research collaborations in the field of condensed matter physics on Arxiv until April 2003. It is a symmetric, binary matrix of order $n = 23,133$ with $186,936$ nonzero entries [25].
- MUFC Twitter: records the interaction between members of the social network Twitter on the 9th of May 2013 between 8am and 8pm when Sir Alex Ferguson resigned as the manager of Manchester United Football Club. The resulting matrix is unsymmetric of order $n = 148,918$ with $193,032$ nonzero entries [18].[2]

Table 6.9 shows the results for Algorithm 5.2 with $\alpha = 3$. Since the full matrix exponential will not always fit in RAM, and in order to make use of level-3 BLAS operations, we find the exact answer by repeatedly using `expmv` to compute $e^A E$ with 100 unit vectors at a time in the matrix E. We see that Algorithm 5.2 computes all of the 10 largest elements correctly in each case. For the two smaller matrices from the SNAP collection we see a speedup of around 150 compared with exact computation. For the MUFC Twitter matrix our new algorithm achieves a speedup of 1,720.

7. Conclusions. There is a long history of matrix condition number estimation, which is summarized in [21, Chap. 15]. Much of the early work was concerned with estimating $\|A^{-1}\|_1$ given a factorization of the square, nonsingular matrix A. It later became apparent that there are benefits to treating the more general problem of estimating $\|B\|_1$ given only the ability to form matrix–vector products with B and B^*, not least because this enables various componentwise condition numbers to be estimated. The LAPACK library [2] takes this approach. The very general norm estimation algorithms of Boyd [4] and Tao [28] have this more general form, but the special case of the M-norm ($\|A\|_M = \max_{i,j} |a_{ij}|$) has received almost no attention prior to this work. However, the applications discussed in section 1 demonstrate a clear need to estimate the M-norm via matrix–vector products.

We have derived algorithms for estimating $\|A\|_M$ and the largest p elements in absolute value of A. For the basic algorithm, in section 3 we gave convergence results, a bound on the expected number of iterations for random matrices, and examples where

[2]Data provided under a Creative Commons licence by The University of Strathclyde and Bloom Agency; http://www.mathstat.strath.ac.uk/outreach/twitter/mufc

the estimator can be arbitrarily poor or take the maximum number of iterations. We derived a block version of the $p = 1$ algorithm that, by iterating on matrices with t columns, provides more accurate estimates and allows for the use of level-3 BLAS for dense A, or the corresponding operations in sparse BLAS libraries. For general p, we introduced a deflation technique that alleviates the tendency of the estimates of the p largest elements all to approximate the largest element.

The numerical experiments on random matrices and on matrices from applications show that the algorithms produce, within just a few iterations, estimates that have mean underestimation ratio Ψ almost always greater than 0.5, are frequently exact, and increase in quality with t. In particular, the algorithms can identify the largest elements of matrices defined implicitly as A^TA or e^A, for large, sparse matrices A, thousands of times faster than if these matrices were explicitly formed. Our experiments show that Algorithm 5.2, incorporating blocking and deflation, performs better than the corresponding algorithms without these features in the majority of the test problems, so this is our recommended algorithm.

Our algorithms incorporate randomness, but not as a fundamental part of their design. Gu [14, sect. 7] has recently proposed a randomized algorithm for estimating the matrix 1-norm. It would be interesting to see whether Gu's ideas can be adapted for the M-norm.

An important feature of our algorithms is that they can be treated as black boxes that can be applied to many different problems, in contrast to the more specialized algorithms designed for products of two matrices, such as those in [3], [29]. Since our algorithms only require the computation of matrix–vector products, they are relatively simple to implement and can serve as a benchmark for testing more specialized algorithms in multiple application areas.

MATLAB implementations of Algorithms 4.1 and 5.2 are available on GitHub at https://github.com/sdrelton/matrix-est-maxelts.

REFERENCES

[1] A. H. Al-Mohy and N. J. Higham, *Computing the action of the matrix exponential, with an application to exponential integrators*, SIAM J. Sci. Comput., 33 (2011), pp. 488–511, https://doi.org/10.1137/100788860.

[2] E. Anderson, Z. Bai, C. Bischof, L. 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, 1999, https://doi.org/10.1137/1.9780898719604.

[3] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, *Diamond sampling for approximate maximum all-pairs dot-product (MAD) search*, in Proceedings of the IEEE International Conference on Data Mining, C. Aggarwal, Z.-H. Zhou, A. Tuzhilin, H. Xiong, and X. Wu, eds., 2015, pp. 11–20, https://doi.org/10.1109/ICDM.2015.46.

[4] D. W. Boyd, *The power method for ℓ^p norms*, Linear Algebra Appl., 9 (1974), pp. 95–101, https://doi.org/10.1016/0024-3795(74)90029-9.

[5] M. Caliari, P. Kandolf, A. Ostermann, and S. Rainer, *Comparison of software for computing the action of the matrix exponential*, BIT, 54 (2014), pp. 113–128, https://doi.org/10.1007/s10543-013-0446-0.

[6] T. A. Davis, *University of Florida Sparse Matrix Collection*, http://www.cise.ufl.edu/research/sparse/matrices.

[7] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25, https://doi.org/10.1145/2049662.2049663.

[8] E. Estrada and N. Hatano, *Communicability in complex networks*, Phys. Rev. E, 77 (2008), 036111, https://doi.org/10.1103/PhysRevE.77.036111.

[9] E. Estrada and D. J. Higham, *Network properties revealed through matrix functions*, SIAM Rev., 52 (2010), pp. 696–714, https://doi.org/10.1137/090761070.

[10] R. Fletcher, *Practical Methods of Optimization*, 2nd ed., Wiley, Chichester, UK, 1987.

[11] L. V. Foster, *The growth factor and efficiency of Gaussian elimination with rook piv-*

oting, J. Comput. Appl. Math., 86 (1997), pp. 177–194, https://doi.org/10.1016/S0377-0427(97)00154-4.

[12] L. V. Foster, *Corrigendum. The growth factor and efficiency of Gaussian elimination with rook pivoting*, J. Comput. Appl. Math., 98 (1998), p. 177, https://doi.org/10.1016/S0377-0427(98)00093-4.

[13] A. Frommer, S. Güttel, and M. Schweitzer, *Efficient and stable Arnoldi restarts for matrix functions based on quadrature*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 661–683, https://doi.org/10.1137/13093491X.

[14] M. Gu, *Subspace iteration randomization and singular value problems*, SIAM J. Sci. Comput., 37 (2015), pp. A1139–A1173, https://doi.org/10.1137/130938700.

[15] M. Gu and L. Miranian, *Strong rank revealing Cholesky factorization*, Electron. Trans. Numer. Anal., 17 (2004), pp. 76–92, http://eudml.org/doc/125035.

[16] W. W. Hager, *Condition estimates*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 311–316, https://doi.org/10.1137/0905023.

[17] F. M. Harper and J. A. Konstan, *The MovieLens datasets: History and context*, ACM Trans. Interactive Intelligent Systems, 5 (2015), pp. 19:1–19:19, https://doi.org/10.1145/2827872.

[18] D. J. Higham, P. Grindrod, A. V. Mantzaris, A. Otley, and P. Laflin, *Anticipating activity in social media spikes*, in Workshop on Modeling and Mining Temporal Interactions within the 9th International AAAI Conference on the Web and Social Media, B. Gonçalves, M. Karsai, and N. Perra, eds., The AAAI Press, Palo Alto, CA, 2015, pp. 1–7.

[19] 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, https://doi.org/10.1145/63522.214391.

[20] N. J. Higham, *Experience with a matrix norm estimator*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 804–809, https://doi.org/10.1137/0911047.

[21] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002, https://doi.org/10.1137/1.9780898718027.

[22] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008, https://doi.org/10.1137/1.9780898717778.

[23] N. J. Higham and F. Tisseur, *A block algorithm for matrix* 1*-norm estimation, with an application to* 1*-norm pseudospectra*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1185–1201, https://doi.org/10.1137/S0895479899356080.

[24] A. N. Langville and C. D. Meyer, *Who's #1? The Science of Rating and Ranking*, Princeton University Press, Princeton, NJ, 2012.

[25] J. Leskovec, J. Kleinberg, and C. Faloutsos, *Graph evolution: Densification and shrinking diameters*, ACM Trans. Knowl. Discov. Data, 1 (2007), pp. 2:1–2:41.

[26] D. Liben-Nowell and J. Kleinberg, *The link prediction problem for social networks*, J. Amer. Soc. Inf. Sci. Tech., 58 (2007), pp. 1019–1031, https://doi.org/10.1002/asi.20591.

[27] J. Peypouquet, *Convex Optimization in Normed Spaces: Theory, Methods and Examples*, Springer-Verlag, New York, 2015, https://doi.org/10.1007/978-3-319-13710-0.

[28] 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), https://doi.org/10.1016/0024-3795(84)90093-4.

[29] C. Teflioudi, R. Gemulla, and O. Mykytiuk, *LEMP: Fast retrieval of large entries in a matrix product*, in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15), 2015, pp. 107–122, https://doi.org/10.1145/2723372.2747647.