Numer Algor (2015) 70:287–308
DOI 10.1007/s11075-014-9947-4

ORIGINAL PAPER

Estimating the condition number of $f(A)b$

Edvin Deadman

Received: 3 July 2014 / Accepted: 2 December 2014 / Published online: 14 December 2014

Abstract New algorithms are developed for estimating the condition number of $f(A)b$, where A is a matrix and b is a vector. The condition number estimation algorithms for $f(A)$ already available in the literature require the explicit computation of matrix functions and their Fréchet derivatives and are therefore unsuitable for the large, sparse A typically encountered in $f(A)b$ problems. The algorithms we propose here use only matrix-vector multiplications. They are based on a modified version of the power iteration for estimating the norm of the Fréchet derivative of a matrix function, and work in conjunction with any existing algorithm for computing $f(A)b$. The number of matrix-vector multiplications required to estimate the condition number is proportional to the square of the number of matrix-vector multiplications required by the underlying $f(A)b$ algorithm. We develop a specific version of our algorithm for estimating the condition number of $e^A b$, based on the algorithm of Al-Mohy and Higham (SIAM J. Matrix Anal. Appl. **30**(4), 1639–1657, 2009). Numerical experiments demonstrate that our condition estimates are reliable and of reasonable cost.

Keywords Matrix function · Matrix exponential · Condition number estimation · Fréchet derivative · Power iteration · Block 1-norm estimator · Python

Mathematics Subject Classifications (2010) 15A60 · 65F35 · 65F60

This work was supported by European Research Council Advanced Grant MATFUN (267526).

E. Deadman (✉)
School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK
e-mail: edvin.deadman@manchester.ac.uk
URL: http://www.maths.manchester.ac.uk/~edeadman

1 Introduction

The computation of $f(A)b$ arises in a wide variety of applications in science and engineering. Here $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$ and $f(A)$ is a matrix function. A particularly important example is the action of the matrix exponential $e^A b$, which is central to the solution of differential equations using exponential integrators [23]. Further examples include the application of $\text{sign}(A)b$ in problems in control theory and lattice quantum chromodynamics [14], and the solution of fractional differential equations using $A^\alpha b$, $\alpha \in \mathbb{R}$ [6].

In applications, typically $A \in \mathbb{C}^{n\times n}$ is large and sparse, but $f(A)$ and b may be dense. This imposes constraints on how $f(A)b$ can be computed. Storing $O(n^2)$ data is not feasible and operating on A using dense matrix techniques requiring $O(n^3)$ flops (such as Schur decomposition) is impractical. In particular, it is undesirable to explicitly compute $f(A)$. In practice, this means that a successful algorithm for computing $f(A)b$ should only require products of the form Av or A^*v where v is a vector.

Much of the early work on the numerical computation of $f(A)b$ was based on Krylov subspace projection methods [15, 27, 30, 34]. The EXPOKIT software package [32] for computing $e^A b$ is based on these methods. More recently, algorithms have been developed using *rational* Krylov methods (see [16] and the references therein).

An alternative approach, suggested in [9] and extended with the use of conformal mappings in [17], is to apply quadrature to a contour integral representation of $f(A)b$.

The final class of methods we mention here are polynomial expansions of the form $f(A) \approx p_k(A)b$ where $p_k(A)$ is a polynomial obtained by truncating an expansion for f in terms of a complete system of polynomials. For example, Sheehan, Saad and Sidje [31] use Chebyshev and Laguerre polynomial expansions. The most widely used example of this type of method is that of Al-Mohy and Higham [2], which uses scaling and a truncated Taylor series to compute $e^A b$. An error analysis in exact arithmetic is used to obtain optimal scaling and truncation parameters. Fischer [13] has recently performed a rounding error analysis for $e^A b$ algorithms based on the scaling method. Although a full backward error analysis is not obtained, some classes of (A, b) for which the algorithms are backward stable are identified.

Implementations of $f(A)b$ algorithms are available in many programming languages. A comprehensive list is given in [20].

The three classes of methods described above each have their own weaknesses. For example rational Krylov methods require large, sparse linear systems to be solved at each iteration. Quadrature and Chebyshev expansion methods are inefficient without some knowledge of the eigensystem of A. The Taylor series method for $e^A b$ can become very expensive when $\|A\|$ is large.

Condition number estimation for $f(A)$ is now a well-developed field. Kenney and Laub [25, 26] developed some general algorithms for condition estimation. More recently, specific algorithms have been developed for estimating condition numbers for the matrix exponential [1], the matrix logarithm [4] and matrix powers [21], all in the 1-norm. The approach taken by these algorithms is to estimate the norm of the Fréchet derivative of $f(A)$, which is equal to the absolute condition number of $f(A)$ [29]. Despite this, the condition number of $f(A)b$, in which perturbations in b must also be considered, has received little attention in the literature. Some bounds are available. For example Al-Mohy and Higham [2] show that the relative condition number of $f(A)b$, $\mathrm{cond}(f, A, b)$, is bounded above in the Frobenius norm by a term involving the relative condition number of $f(A)$, $\mathrm{cond}(f, A)$:

$$\mathrm{cond}(f, A, b) \le \frac{\|f(A)\|_F \|b\|_F}{\|f(A)b\|_F}(1 + \mathrm{cond}(f, A)).$$

However, the available algorithms for estimating $\mathrm{cond}(f, A)$ require the computation of $f(A)$ and its Fréchet derivatives. This approach is not feasible for the large, sparse A typically of interest in $f(A)b$ problems. We know of no algorithms designed for estimating the condition number of $f(A)b$.

The contribution in this work is the development of a general algorithm for estimating the condition number of $f(A)b$. The algorithm works in conjunction with any method for computing $f(A)b$ that relies only on matrix-vector products or the solution of linear systems. Hence the algorithm itself requires only matrix-vector products and linear system solves, and is suitable for large, sparse A. We have developed a specific version of our algorithm for the action of the matrix exponential, based on Al-Mohy and Higham's $e^A b$ algorithm [2].

In many applications, it is the quantity $f(tA)b$ that is of interest, rather than $f(A)b$, where t is a scalar. For example, in applications of exponential integrators, t typically denotes a time step. In Section 2 we define a condition number that takes into account the effect of perturbations in A, b and t. We obtain some useful bounds on which to base our estimates, and show that the effect of perturbing t increases the condition number by at most a factor 2. Thus for the purposes of condition estimation, perturbations in t can be neglected. In Section 3 we develop algorithms for estimating the condition number, based on computing the bounds in Section 2 via the norm of the Fréchet derivative of $f(A)$. We consider the specific case $e^{tA}b$ in Section 4. Numerical experiments are given in Section 5, using a combination of dense matrices of size $n \le 100$ and a selection of large, sparse test matrices taken from the $f(A)b$ literature.

2 Defining the condition number

The standard notation in the matrix function literature uses $\mathrm{cond}_{\mathrm{rel}}(f, A)$ and $\mathrm{cond}_{\mathrm{abs}}(f, A)$ to denote the relative and absolute condition numbers of $f(A)$. In this paper we will only consider relative condition numbers, so we drop the subscript rel and use $\operatorname{cond}(f, A, b, t)$ to denote the relative condition number of $f(tA)b$:

$$\operatorname{cond}(f, A, b, t) := \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon\|A\| \\ \|\Delta b\|\le\epsilon\|b\| \\ |\Delta t|\le\epsilon|t|}} \frac{\|f((t+\Delta t)(A+\Delta A))(b+\Delta b) - f(tA)b\|}{\epsilon\|f(tA)b\|} \tag{2.1}$$

$$= \lim_{\epsilon\to 0} \sup_{\substack{\|\Delta A\|\le\epsilon\|A\| \\ \|\Delta b\|\le\epsilon\|b\| \\ |\Delta t|\le\epsilon|t|}} \frac{1}{\epsilon\|f(tA)b\|}\Big(\|L_f(tA, t\Delta A)b + f(tA)\Delta b + L_f(tA, \Delta tA)b + o(\|\Delta A\|) + o(\|\Delta b\|) + o(|\Delta t|)\|\Big), \tag{2.2}$$

where $L_f(A, \Delta A)$ is the Fréchet derivative of f in the direction of the matrix ΔA and $\|\cdot\|$ denotes any choice of vector norm and the corresponding induced matrix norm.

Note that if $f(A) = A^{-1}$ and t is ignored, then (2.1) is precisely equivalent to the standard normwise condition number for the linear system $Ax = b$, which, in the notation of [18], is given by

$$\kappa_{A,b}(A, x) := \lim_{\epsilon\to 0} \sup\left\{\frac{\|\Delta x\|}{\epsilon\|x\|} : (A+\Delta A)(x+\Delta x) = b+\Delta b, \quad \|\Delta A\| \le \epsilon\|A\|, \quad \|\Delta b\| \le \epsilon\|b\|\right\}.$$

It can be shown (see, for example, [18, Section 7.1]) that

$$\kappa_{A,b}(A, x) = \frac{\|A^{-1}\|\|b\|}{\|x\|} + \|A^{-1}\|\|A\|.$$

Thus the condition number of $A^{-1}b$ can easily be computed. For general f however, it is not possible to express $\operatorname{cond}(f, A, b, t)$ in such a simple form, so a different approach is needed.

An upper bound for $\operatorname{cond}(f, A, b, t)$ can be obtained by separating out the terms in the numerator of (2.2), ignoring the $o(\|\Delta A\|)$, $o(\|\Delta b\|)$ and $o(|\Delta t|)$ terms and using the linearity of the Fréchet derivative in its second argument. This gives

$$\begin{aligned}
\operatorname{cond}(f, A, b, t) \le\ & \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\le\epsilon\|A\|} \frac{\|L_f(tA, t\Delta A)b\|}{\epsilon\|f(tA)b\|} + \lim_{\epsilon\to 0} \sup_{\|\Delta b\|\le\epsilon\|b\|} \frac{\|f(tA)\Delta b\|}{\epsilon\|f(tA)b\|} \\
& + \lim_{\epsilon\to 0} \sup_{|\Delta t|\le\epsilon|t|} \frac{\|\Delta t L_f(tA, A)b\|}{\epsilon\|f(tA)b\|} \\
=\ & \lim_{\epsilon\to 0} \sup_{\|\Delta A\|\le\epsilon} \frac{\|L_f(tA, \Delta A/\epsilon)b\|\|tA\|}{\|f(tA)b\|} \\
& + \lim_{\epsilon\to 0} \sup_{\|\Delta b\|\le\epsilon} \frac{\|f(tA)\Delta b/\epsilon\|\|b\|}{\|f(tA)b\|} + \frac{\|tL_f(tA, A)b\|}{\|f(tA)b\|} \\
=\ & \sup_{\|\Delta A\|\le 1} \frac{\|L_f(tA, \Delta A)b\|\|tA\|}{\|f(tA)b\|} + \sup_{\|\Delta b\|\le 1} \frac{\|f(tA)\Delta b\|\|b\|}{\|f(tA)b\|} \qquad (2.3) \\
& + \frac{\|L_f(tA, tA)b\|}{\|f(tA)b\|} \\
=\ & \sup_{\|\Delta A\|=1} \frac{\|L_f(tA, \Delta A)b\|\|tA\|}{\|f(tA)b\|} + \frac{\|f(tA)\|\|b\|}{\|f(tA)b\|} + \frac{\|L_f(tA, tA)b\|}{\|f(tA)b\|}.
\end{aligned}$$

To obtain a lower bound, we take two of the quantities ΔA, Δb, Δt to be zero. Thus

$$\begin{aligned}
& \frac{1}{\|f(tA)b\|} \max\left(\|tA\| \max_{\|\Delta A\|=1} \|L_f(tA, \Delta A)b\|, \|f(tA)\|\|b\|, \|L_f(tA, tA)b\| \right) \\
& \le \operatorname{cond}(f, A, b, t) \\
& \le \frac{1}{\|f(tA)b\|} \left(\|tA\| \max_{\|\Delta A\|=1} \|L_f(tA, \Delta A)b\| + \|f(tA)\|\|b\| + \|L_f(tA, tA)b\| \right). \qquad (2.4)
\end{aligned}$$

The upper and lower bounds differ by a factor of at most 3 so estimating either will provide a suitable estimate for $\operatorname{cond}(f, A, b, t)$. If $t = 1$ is fixed, the problem reduces to finding the condition number of $f(A)b$, which we denote by $\operatorname{cond}(f, A, b)$. In this case (2.4) reduces to

$$\begin{aligned}
& \frac{1}{\|f(A)b\|} \max\left(\|A\| \max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|, \|f(A)\|\|b\| \right) \le \operatorname{cond}(f, A, b) \\
& \qquad \le \frac{1}{\|f(A)b\|} \left(\|A\| \max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\| + \|f(A)\|\|b\| \right),
\end{aligned}$$

with the upper and lower bounds now differing by a factor of at most 2. Essentially the same result was originally obtained by Al-Mohy and Higham [2, Lemma 4.2].

By choosing $\Delta A = A/\|A\|$, we find that

$$\|tA\| \max_{\|\Delta A\|=1} \|L_f(tA, \Delta A)b\| \geq \|L_f(tA, tA)b\|,$$

so, as one might expect, the terms in (2.4) caused by the perturbation of t are smaller than those caused by perturbing A. It follows that

$$\mathrm{cond}(f, tA, b) \leq \mathrm{cond}(f, A, b, t) \leq 2\,\mathrm{cond}(f, tA, b), \tag{2.5}$$

and that

$$\begin{aligned} &\frac{1}{\|f(tA)b\|} \max\left(\|tA\| \max_{\|\Delta A\|=1} \|L_f(tA, \Delta A)b\|, \|f(tA)\|\|b\|\right) \\ &\quad \leq \mathrm{cond}(f, tA, b) \leq \mathrm{cond}(f, A, b, t) \\ &\quad \leq \frac{1}{\|f(tA)b\|}\left(2\|tA\| \max_{\|\Delta A\|=1} \|L_f(tA, \Delta A)b\| + \|f(tA)\|\|b\|\right). \end{aligned} \tag{2.6}$$

In practice, only order of magnitude estimates of condition numbers are ever needed. The upper bound in (2.6) is an overestimate by at most a factor of 6 and it is this quantity that we will use as our condition estimate. Furthermore, (2.5) suggests that for simplicity we can ignore t and consider just $f(A)b$ and $\mathrm{cond}(f, A, b)$. A suitable estimate for $\mathrm{cond}(f, A, b, t)$ can be found by simply replacing A with tA.

3 Estimating the condition number of $f(A)b$

Computing an estimate of the upper bound in (2.6) depends on our ability to estimate $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$, $\|A\|$ and $\|f(A)\|$ using only products of the form Av and A^*v. The main aim of this section is to develop methods for estimating $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$, but it will be instructive to briefly discuss $\|A\|$ and $\|f(A)\|$ first.

The quantity $\|A\|_1$ can be estimated using the 1-norm estimation algorithm of Higham and Tisseur [22], hereafter referred to as `normest1`. The algorithm returns an estimate using fewer than 10 products of the form Av and A^*v. Turning to the 2-norm, the power method (see for example [19, Algorithm 3.19]) also uses only matrix-vector products. Convergence depends on the singular values of A, but in practice very few iterations are required to obtain an order of magnitude estimate of $\|A\|_2$.

The power method and `normest1` can both also be used to estimate $\|f(A)\|$. Products of the form $f(A)v$ or $f(A)^*v$ are now required. These products can be computed provided that an algorithm for $f(A)b$ is available.

One way of estimating $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$ is to simply try several random choices of ΔA. Kenney and Laub [26] made this approach rigorous for real matrices and showed how, in practice, few tries are required in order to achieve a high probability of being within an order of magnitude of the true maximum.

However, in the context of the $f(A)b$ problem for large, sparse A, it is undesirable to store dense $O(n^2)$ quantities such as a randomly chosen ΔA. This approach may be of interest when only perturbations in the nonzero elements of A are of relevance to the condition number, but we will not study this further here. Instead we will investigate another class of methods for estimating $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$, based on the Kronecker form of the Fréchet derivative.

By linearity in ΔA, the vector $L_f(A, \Delta A)b$ can be written in the Kronecker form $L_f(A, \Delta A)b = K_f(A, b)\,\mathrm{vec}(\Delta A)$, where $K_f(A, b) \in \mathbb{C}^{n\times n^2}$ and the vec operator stacks the columns of a matrix on top of each other. The Hermitian conjugate of the Kronecker form, $K_f(A, b)^*$, can be related to the adjoint of the Fréchet derivative as follows. For $P, Q \in \mathbb{C}^{p\times q}$ let $\langle P, Q\rangle = \mathrm{trace}(Q^*P)$. Then for $y \in \mathbb{C}^n$ and $E \in \mathbb{C}^{n\times n}$ we have

$$\langle K_f(A, b)\,\mathrm{vec}(E), y\rangle = \langle L_f(A, E)b, y\rangle.$$

The adjoint of the Fréchet derivative, $L_f^\star : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$, is then defined via

$$\langle L_f(A, E), F\rangle = \langle E, L_f^\star(A, F)\rangle.$$

The Hermitian conjugate $K_f(A, b)^*$ can now be expressed in terms of $L_f^\star$.

$$\begin{aligned}\langle \mathrm{vec}(E), K_f(A, b)^*y\rangle &= \langle K_f(A, b)\,\mathrm{vec}(E), y\rangle\\ &= \langle L_f(A, E)b, y\rangle\\ &= \langle L_f(A, E), yb^*\rangle\\ &= \langle E, L_f^\star(A, yb^*)\rangle.\end{aligned}$$

Hence

$$K_f(A, b)^*y = L_f^\star(A, yb^*). \tag{3.7}$$

Various candidate algorithms to compute or estimate the quantity $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$ via the Kronecker form are now available, depending on our choice of norm. We consider first the Frobenius norm, and note that

$$\max_{\|\Delta A\|_F=1} \|L_f(A, \Delta A)b\|_F = \max_{\|\mathrm{vec}(\Delta A)\|_2=1} \|K_f(A, b)\,\mathrm{vec}(\Delta A)\|_2 = \|K_f(A, b)\|_2.$$

By explicitly forming $K_f(A, b)$ and computing its 2-norm, we can obtain $\max_{\|\Delta A\|_F=1} \|L_f(A, \Delta A)b\|_F$. The Frobenius norm is not induced by any vector norm so it is not an ideal choice for computing the upper bound in (2.6). However, since

$$\frac{1}{\sqrt{n}} \max_{\|\Delta A\|_1=1} \|L_f(A, \Delta A)b\|_1 \le \max_{\|\Delta A\|_F=1} \|L_f(A, \Delta A)b\|_F$$
$$\le \sqrt{n} \max_{\|\Delta A\|_1=1} \|L_f(A, \Delta A)b\|_1, \tag{3.8}$$

we can use $\sqrt{n}\|K_f(A, b)\|_2$ as an upper bound for $\max_{\|\Delta A\|_1=1} \|L_f(A, \Delta A)b\|_1$. Assuming that we are able to compute quantities of the form $L_f(A, E)b$, the following algorithm, analogous to [19, Algorithm 3.17], estimates $\operatorname{cond}(f, A, b)$.

Algorithm 3.1 For a function f, a matrix $A \in \mathbb{C}^{n\times n}$ and a vector $b \in \mathbb{C}^n$, this algorithm computes an upper bound κ for $\operatorname{cond}(f, A, b)$ in the 1-norm, with $\kappa \le 6\sqrt{n}\operatorname{cond}(f, A, b)$.

```
for $j = 1\colon n$
    for $i = 1\colon n$
        Compute $y = L_f(A, e_i e_j^T)b$
        $K_f(:, (j-1)n + i) = y$
    end
end
return $\kappa = (2\sqrt{n}\|K_f(A, b)\|_2\|A\|_1 + \|f(A)\|_1\|b\|_1)/\|f(A)b\|_1$
```

Cost: $O(n^5)$ flops assuming evaluation of matrix functions and their derivatives cost $O(n^3)$ flops.

Due to its cost, and the need to store the $n \times n^2$ quantity K_f, Algorithm 3.1 will be of practical use only for small problems, but it will enable us to test the more efficient algorithms we develop later.

An alternative approach to estimating $\operatorname{cond}(f, A, b)$ is to adapt the standard 1-norm condition estimation method for matrix functions [19, Algorithm 3.22], which is based on estimating the 1-norm of the Kronecker form of the Fréchet derivative using `normest1`. Using [11, Lemma 2.1] we deduce that

$$\frac{1}{n} \max_{\|\Delta A\|_1=1} \|L_f(A, \Delta A)b\|_1 \le \|K_f(A, b)\|_1 \le \max_{\|\Delta A\|_1=1} \|L_f(A, \Delta A)b\|_1.$$

The quantity $\|K_f(A, b)\|_1$ can then be estimated using [11, Algorithm 2.2], which applies `normest1` to the rectangular matrix $K_f(A, b)$. This requires computation of quantities of the form $K_f(A, b)v$ or $K_f(A, b)^*w$, where $v \in \mathbb{C}^{n^2}$ and $w \in \mathbb{C}^n$. Although these can be computed without explicitly forming $K_f(A, b)$, the approach still depends on the storage and use of dense vectors of size n^2. This will not be practical if A is large and sparse.

Consider instead [19, Algorithm 3.20]. The algorithm is an application of the power method to $K_f(A)^*K_f(A)$, where $K_f(A) \in \mathbb{C}^{n^2\times n^2}$ is the Kronecker form of the Fréchet derivative of $f(A)$. The result is an estimate of $\|K_f(A)\|_2 = \|L_f(A)\|_F$.

The power method can instead be applied to $K_f(A,b)^* K_f(A,b)$ to provide an estimate for $\|K_f(A,b)\|_2$. This results in the following algorithm.

Algorithm 3.2 For the function f, $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$ this algorithm computes an estimate $\gamma \le \|K_f(A,b)\|_2$.

```
1  Choose a nonzero starting matrix $Z_0 \in \mathbb{C}^{n\times n}$ with $\|Z_0\|_F = 1$
2  for $k = 0{:}\,\infty$
3      $w_{k+1} = L_f(A, Z_k)b$
4      $Z_{k+1} = L_f^\star(A, w_{k+1}b^*)$
5      $\gamma_{k+1} = \|Z_{k+1}\|_F / \|w_{k+1}\|_F$
6      if converged, $\gamma = \gamma_{k+1}$, quit, end
7      $Z_{k+1} = Z_{k+1}/\|Z_{k+1}\|_F$
8  end
```

A typical convergence test for γ is $|\gamma_{k+1} - \gamma_k| \le \text{tol}\gamma_{k+1}$ or $k > \text{it\_max}$, where $\text{tol} = 0.1$ is sufficient for an order of magnitude estimate. In practice very few power iterations are required and it is reasonable to take $\text{it\_max} = 10$.

Algorithm 3.2 suffers from the same issues that were encountered when applying [11, Algorithm 2.2] to $K_f(A,b)$: Z_k has size $n \times n$ and will in general be dense. However, a few alterations to Algorithm 3.2 will enable us to estimate $\|K_f(A,b)\|_2$ whilst only storing vectors of size n.

First we swap the order of L_f and $L_f^\star$ so that we are estimating the 2-norm of the $n \times n$ matrix $K_f(A,b)K_f(A,b)^*$ instead of the $n^2 \times n^2$ matrix $K_f(A,b)^* K_f(A,b)$. In particular we can now use a random vector as our starting point rather than a random $n \times n$ matrix.

Second, in Algorithm 3.2 there are two iterated quantities, $w_k \in \mathbb{C}^n$ and $Z_k \in \mathbb{C}^{n\times n}$. We can merge lines 3 and 4 into single iterated quantity, $y_k \in \mathbb{C}^n$. Then $y_{k+1} = L_f(A, L_f^\star(A, y_k b^*))b$.

Using y_k as our iterated quantity prevents us from using the ratio $\gamma_{k+1} = \|Z_{k+1}\|_F / \|w_{k+1}\|_F$ in line 5. Instead we must use $\sqrt{\|y_{k+1}\|_F / \|y_k\|_F}$. This is a weaker lower bound to $\|K_f(A,b)K_f(A,b)\|_2$ than $\|Z_{k+1}\|_F / \|w_{k+1}\|_F$ but this is a necessary evil, and numerical tests will show that the number of iterations required for convergence is still very low.

Applying these changes to Algorithm 3.2, we obtain the following algorithm.

Algorithm 3.3 For the function f, $A \in \mathbb{C}^{n\times n}$ and $b \in \mathbb{C}^n$ this algorithm computes an estimate $\gamma \le \|K_f(A,b)\|_2$.

```
Choose a unit nonzero starting vector $y_0 \in \mathbb{C}^n$
for $k = 0{:}\,\infty$
    $y_{k+1} = L_f(A, L_f^\star(A, y_k b^*))b$
    $\gamma_{k+1} = \sqrt{\|y_{k+1}\|_2}$
    if $|\gamma_{k+1} - \gamma_k| < 0.1\gamma_{k+1}$ or $k > \text{it\_max}$, $\gamma = \gamma_{k+1}$, quit, end
    $y_{k+1} = y_{k+1}/\|y_{k+1}\|_2$
end
```

The ease with which we can compute $L_f(A, L_f^\star(A, y_k b^*))b$ will determine how useful Algorithm 3.3 is. To compute $L_f(A, L_f^\star(A, y_k b^*))b$ for general f, we first note that the Fréchet derivative of a matrix function can be evaluated by computing the function of a larger matrix:

$$f\left(\begin{bmatrix} A & E \\ 0 & A \end{bmatrix}\right) = \begin{bmatrix} f(A) & L_f(A, E) \\ 0 & f(A) \end{bmatrix}.$$

Under suitable conditions on f [21, Lemma 6.2], which are satisfied for 'standard' functions such as the exponential, the adjoint of the Fréchet derivative is given by

$$L_f^\star(A, E) = L_{\bar{f}}(A, E^*)^*,$$

where $\bar{f}(z) = \overline{f(\bar{z})}$. This implies

$$\bar{f}\left(\begin{bmatrix} A^* & E \\ 0 & A^* \end{bmatrix}\right) = \begin{bmatrix} \bar{f}(A^*) & L_f^\star(A^*, E) \\ 0 & \bar{f}(A^*) \end{bmatrix}.$$

We can exploit the above relations to develop a very general method of computing $L_f(A, L_f^\star(A, y_k b^*))b$ that depends only on the manner in which $f(A)b$ is computed. We note that $L_f(A, L_f^\star(A, y_k b^*))b$ is given by the top n elements of

$$f\left(\begin{bmatrix} A & L_f^\star(A, y_k b^*) \\ 0 & A \end{bmatrix}\right)\begin{bmatrix} 0 \\ b \end{bmatrix}. \tag{3.9}$$

Suppose we have an algorithm for computing $f(A)b$ which requires Π matrix-vector multiplications. Then computing (3.9) will require Π multiplications of the form

$$\begin{bmatrix} A & L_f^\star(A, y_k b^*) \\ 0 & A \end{bmatrix}\begin{bmatrix} p \\ q \end{bmatrix}$$

for some $p, q \in \mathbb{C}^n$. Each such multiplication requires the computation of $L_f^\star(A, y_k b^*)q$ in addition to the two multiplications Ap and Aq. We can evaluate $L_f^\star(A, y_k b^*)q$ by computing

$$\bar{f}\left(\begin{bmatrix} A^* & y_k b^* \\ 0 & A^* \end{bmatrix}\right)\begin{bmatrix} 0 \\ q \end{bmatrix},$$

and storing the top n entries. This itself requires Π multiplications of the form

$$\begin{bmatrix} A^* & y_k b^* \\ 0 & A^* \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A^* u + y_k b^* v \\ A^* v \end{bmatrix}.$$

Thus the total number of matrix-vector multiplications (of size n) required to compute $L_f(A, L_f^\star(A, y_k b^*))b$ is $2\Pi(\Pi+1)$. Since only an order of magnitude condition estimate is required, $L_f(A, L_f^\star(A, y_k b^*))b$ need not be evaluated to double precision accuracy, so Π need not be as large as when $f(A)b$ itself is computed in double precision.

Rational Krylov methods also require linear systems of the form $Ar = s$ or $A^* r = s$ to be solved to evaluate $f(A)b$. This can easily be accounted for in our approach.

Evaluating (3.9) using a Krylov method may require solutions of systems of the form

$$\begin{bmatrix} A & L_f^{\star}(A, y_k b^*) \\ 0 & A \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}.$$

The solution can be obtained by solving the smaller systems $Ay = q$ and $Ax = p - L_f^{\star}(A, y_k b^*)y$. The quantity $L_f^{\star}(A, y_k b^*)y$ is computed by finding

$$\bar{f}\left(\begin{bmatrix} A^* & y_k b^* \\ 0 & A^* \end{bmatrix}\right)\begin{bmatrix} 0 \\ y \end{bmatrix},$$

with any linear systems of size $2n$ split into 2 systems of size n in a similar manner.

We collate the ideas above into the following algorithm for computing $L_f(A, L_f^{\star}(A, y_k b^*))b$.

Algorithm 3.4 Given a method for computing $f(A)b$ that requires only matrix-vector multiplications and the solution of linear systems, this algorithm computes $L_f(A, L_f^{\star}(A, y_k b^*))b$. The user-supplied subroutine $y = \mathrm{fAb}(x)$, where x is a vector, must compute the quantity $y = f(X)x$ for some matrix X by requesting products of the form Xv or solutions to linear systems of the form $Xv = w$.

```
$[l_1, l_2]^T = \mathrm{fAb}([0, b]^T)$:
    for each product required with a vector $[p, q]^T$:
        compute and $Ap$ and $Aq$
        $[r_1, r_2]^T = \mathrm{fAb}([0, \bar{q}]^T)$:
            for each product required with a vector $[u, v]^T$:
                provide $[\bar{A}^* u + \overline{y_k b^*} v, \bar{A}^* v]^T$
            end
            for each linear system with right-hand side $[u, v]^T$:
                solve $\bar{A}^* y = v$, $\bar{A}^* x = u - \overline{y_k b^*} y$, provide $[x, y]^T$
            end
        provide $[Ap + \bar{r}_1, Aq]^T$
    end
    for each linear system with right-hand side $[p, q]^T$:
        solve $Ay = q$
        $[r_1, r_2]^T = \mathrm{fAb}([0, \bar{y}]^T)$:
            for each product required with a vector $[u, v]^T$:
                provide $[\bar{A}^* u + \overline{y_k b^*} v, \bar{A}^* v]^T$:
            end
            for each linear system with right-hand side $[u, v]^T$:
                solve $\bar{A}^* t = v$, $\bar{A}^* s = u - \overline{y_k b^*} t$, provide $[s, t]^T$
            end
        solve $Ax = p - \bar{r}_1$
        provide $[x, y]^T$
    end
return $l_1$
```

Cost: If $f(A)b$ can be evaluated using Π matrix-vector products and Γ linear system solves (of size n), then $L_f(A, L_f^{\star}(A, y_k b^*))b$ is evaluated using $2\Pi(\Pi + \Gamma + 1)$ products and $2\Gamma(\Gamma + \Pi + 1)$ solves (also of size n).

If f has a Taylor series with real coefficients, so that $f \equiv \bar{f}$, then the complex conjugation can be removed whenever it appears in Algorithm 3.4.

We are now in a position to state our overall algorithm for estimating cond(f, A, b). Algorithms 3.3 and 3.4 are used to estimate the quantity $\max_{\|\Delta A\|_F=1} \|L_f(A, \Delta A)b\|_F$. This is combined with (3.8) to give an estimate of the upper bound in (2.6) in the 1-norm.

Algorithm 3.5 Given a method for computing $f(A)b$ that requires only matrix-vector multiplications or linear system solves, this algorithm computes an estimate of a quantity $\kappa \geq \operatorname{cond}(f, A, b)$ with $\kappa \leq 6\sqrt{n}\operatorname{cond}(f, A, b)$ in the 1-norm. The user-supplied subroutine $y = \mathrm{fAb}(x)$, where x is a vector, must compute the quantity $y = f(X)x$ for some matrix X by requesting products of the form Xv or solutions to linear systems of the form $Xv = w$.

```
Compute $\gamma$ using Algorithm 3.3, with Algorithm 3.4 used to evaluate
    $L_f(A, L_f^\star(A, y_k b^*))b$
Estimate $\|A\|_1$ using normest1
Estimate $\|f(A)\|_1$ using normest1,with $y = \mathrm{fAb}(x)$ used to provide
    $f(A)v$ or $f(A)^* v$
$\kappa = (2\sqrt{n}\gamma\|A\|_1 + \|f(A)\|_1\|b\|_1)/\|f(A)b\|_1$
```

Some $f(A)b$ algorithms are actually designed to compute $f(A)B$, where $B \in \mathbb{C}^{n\times m}$ (for example [2]). Our analysis and algorithms readily generalize to this case. In (2.3) the term

$$\sup_{\|\Delta B\|\leq 1} \|f(A)\Delta B\|$$

now arises as a result of perturbing B. In the 1-norm this term is simply $\|f(A)\|_1$. The iterated quantity in Algorithm 3.3 becomes

$$Y_{k+1} = L_f(A, L_f^\star(A, Y_k B^*))B \in \mathbb{C}^{n\times m},$$

and vector 2-norms in Algorithm 3.3 are replaced with the Frobenius norm for $n \times m$ matrices. No other changes are required to the algorithms in this section.

Algorithm 3.5 is designed to work for general f but it can be made more efficient by adapting it to specific $f(A)b$ algorithms. In Section 4 we consider the specific case of the action of the matrix exponential $e^A b$.

4 Application to the action of the matrix exponential

Al-Mohy and Higham's algorithm [2] uses a scaling scheme based on the identity $e^A = (e^{A/s})^s$, $s \in \mathbb{N}$ to reduce the norm of A and improve the convergence properties of a truncated Taylor series. Error analysis is available to obtain an optimal choice of scaling parameter s and truncation parameter m for a given precision. Approximately sm matrix-vector multiplications are required to estimate $e^A b$.

It is natural to ask whether a similar approach might be preferable to Algorithm 3.4 for computing the quantity $L_f(A, L_f^\star(A, y_k b^*))b$. Substituting the truncated Taylor series for $e^{A/s}$ into $(e^{A/s})^s$ and differentiating, we obtain

$$L_{\exp}(A, L^{\star}_{\exp}(A, yb^*))b = \sum_{i=1}^{m}\sum_{j=1}^{i}\sum_{k=1}^{m}\sum_{l=1}^{k}\sum_{p=1}^{s}\sum_{q=1}^{s}\left\{\frac{a_i a_k}{s^{k+i}}(e^{A/s})^{q-1}A^{j-1}(e^{A^*/s})^{p-1}(A^*)^{l-1}\right.$$
$$\left.(yb^*)(A^*)^{k-l}(e^{A^*/s})^{s-p}A^{i-j}(e^{A/s})^{s-q}b\right\},$$

where $a_i = 1/i!$. Computing $L_f(A, L^{\star}_f(A, y_k b^*))b$ in this manner requires $O(s^3m^7)$ matrix-vector multiplications. In comparison, using Algorithm 3.4 requires $O(s^2m^2)$ multiplications. We therefore focus on adapting Algorithm 3.4.

Al-Mohy and Higham devised a method of choosing s and m to obtain e^Ab with a given precision (in exact arithmetic) in the most efficient manner possible. Quantities of the form $\|A^p\|_1^{1/p}$ are computed using `normest1` for $p = 2, \ldots, 8$. These quantities are compared with a set of pre-computed parameters θ_m, $m = 2, \ldots, 55$ whose values depend on the desired precision. For each m, θ_m is used to choose s such that the backward error bound (in exact arithmetic) does not exceed the desired precision. The choice of m which results in the smallest value of sm is then used. Since only an order of magnitude estimate for the condition number is required, the θ_m for half precision can be used. Numerical experiments in Section 5 show that this results in a significant computational saving. Table 1 shows selected values of θ_m.

The evaluation of $L_{\exp}(A, L^{\star}_{\exp}(A, yb^*))b$ using Algorithm 3.4 effectively requires the computation of quantities of the form $e^X y$ where

$$X = \begin{bmatrix} A & L^{\star}_f(A, y_k b^*) \\ 0 & A \end{bmatrix}, \text{ or } X = \begin{bmatrix} A^* & y_k b^* \\ 0 & A^* \end{bmatrix}.$$

In theory, to guarantee the desired precision, for every different X we would need to recompute s and m to take into account the changing $(1, 2)$ block. In practice, it is more desirable to compute s and m once at the beginning of the algorithm. If the 1-norm of the $(1, 2)$ block in X does not exceed $\|A\|_1$, then the s and m which are optimal for e^Ab are also optimal for e^Xy. If the $(1, 2)$ block is larger in 1-norm than $\|A\|_1$, then the s and m which are optimal for e^Ab may not be optimal for e^Xy and a loss of accuracy could result. In practice, since only an order of magnitude estimate of the condition number is required, our numerical experiments suggest that computing s and m once at the beginning of the algorithm is nevertheless acceptable.

Table 1 Selected constants θ_m for half, single and double precision

m	5	10	15	20	25	30	35	40	45	50	55
Half	7.1e-1	2.2e0	3.7e0	5.2e0	6.6e0	8.1e0	9.5e0	1.1e1	1.2e1	1.4e1	1.5e1
Single	1.3e-1	1.0e0	2.2e0	3.6e0	4.9e0	6.3e0	7.7e0	9.1e0	1.1e1	1.2e1	1.3e1
Double	2.4e-3	1.4e-1	6.4e-1	1.4e0	2.4e0	3.5e0	4.7e0	6.0e0	7.2e0	7.5e0	7.5e0

This provides a considerable computational saving. To choose s and m we therefore use [2, Code Fragment 3.1], with the half precision θ_m from Table 1.

To fully adapt Algorithm 3.5 for $e^A b$, we must consider three further points.

First we recall that Al-Mohy and Higham's algorithm translates A by $\mu = \mathrm{trace}(A)/n$. Incorporating this into the algorithm is straightforward.

Second, in their algorithm balancing can be used if desired. This is a diagonal similarity transformation, $\tilde{A} = D^{-1}AD$, that attempts to equalize the norms of the ith row and ith column of A for all i. Balancing can lead to a smaller choice of s and m, but should not be used if $\|\tilde{A}\|_1 > \|A\|_1$. Note that

$$f(A) = Df(\tilde{A})D^{-1},$$

which can be differentiated to give

$$L_f(A, E) = DL_f(\tilde{A}, D^{-1}ED)D^{-1}.$$

We deduce that

$$L_f(A, L_f^{\star}(A, y_k b^*))b = DL_f(\tilde{A}, D^{-1}\bar{D}^{-1}L_{\bar{f}}(\tilde{A}^*, \bar{D}y_k(D^{-1}b)^*)\bar{D}D)D^{-1}b.$$

For dense A, a candidate balancing transformation can be obtained by using the LAPACK routines DGEBAL (for real A) or ZGEBAL (for complex A). For sparse A, the algorithms of Chen and Demmel [7] can be used.

Finally, we reintroduce the scalar t, so that we are computing the condition number of $e^{tA}b$. We recall from (2.5) that the condition numbers obtained with or without perturbing t agree to within a factor 2. Simply applying the framework of Algorithm 3.5 to the scaled matrix tA will provide the required upper bound from (2.6).

We can now state our algorithm for estimating $\mathrm{cond}(\exp, A, b, t)$ in full.

Algorithm 4.1 This algorithm computes an estimate of a quantity $\kappa \geq \mathrm{cond}(\exp, A, b, t)$. κ is within a factor of $6\sqrt{n}$ of $\mathrm{cond}(\exp, A, b, t)$ in the 1-norm. The subroutine `parameters`(A, tol) refers to [2, Code Fragment 3.1], which computes the parameters m and s for the precision tol.

```
estimate $\|A\|_1$, using normest1
balance = true, $\sigma = \|b\|_1$, tol $= 2^{-11}$ % unit roundoff for half precision
$\tilde{A} = D^{-1}AD$, where $D$ is the diagonal matrix computed by the chosen
    balancing routine
if $\|\tilde{A}\|_1 < \|A\|_1$, $A = \tilde{A}$, $b = D^{-1}b$, else balance = false, end
if trace$(A)$ is available, $\mu = \mathrm{trace}(A)/n$, else $\mu = 0$, end
$A = A - \mu I$
$[m, s] =$ parameters$(tA, \mathrm{tol})$ % compute $m$ and $s$ using
    [2, Code Fragment 3.1] with half precision $\theta_m$
$\eta = e^{t\mu/s}$
choose unit nonzero starting vector $y_0 \in \mathbb{C}^n$ % Start of the power
    method for $K_{\mathrm{exp}}(tA, b)$
for $k = 0\colon \infty$
```

```
11          if balance, $y_k = \bar{D}y_k$, end
12          $F_1 = 0$, $F_2 = b$, $b_1 = 0$, $b_2 = b$
13          for $i = 1\colon s$
14              $c_1 = \max\{\|b_1\|_\infty, \|b_2\|_\infty\}$
15              for $j = 1\colon m$
16                  $G_1 = 0$, $G_2 = b_2$, $d_1 = 0$, $d_2 = b_2$
17                  if balance, $d_2 = \bar{D}Dd_2$, end
18                  for $q = 1\colon s$
19                      $c_3 = \max\{\|d_1\|_\infty, \|d_2\|_\infty\}$
20                      for $r = 1\colon m$
21                        $d_1 = (\bar{t}A^* d_1 + y_k b^* d_2)/(sr)$, $d_2 = (\bar{t}A^* d_2)/(sr)$
22                        $c_4 = \max\{\|d_1\|_\infty, \|d_2\|_\infty\}$
23                        $G_1 = G_1 + d_1$, $G_2 = G_2 + d_2$
24                        if $c_3 + c_4 \leq$ tol × $\max\{\|G_1\|_\infty, \|G_2\|_\infty\}$,
                          break, end
25                        $c_3 = c_4$
26                      end
27                      $G_1 = \bar{\eta}G_1$, $G_2 = \bar{\eta}G_2$, $d_1 = G_1$, $d_2 = G_2$
28                  end
29                  if balance, $G_1 = D^{-1}\bar{D}^{-1}G_1$, end
30                  $b_1 = (tAb_1 + G_1)/(sj)$, $b_2 = tAb_2/(sj)$
31                  $c_2 = \max\{\|b_1\|_\infty, \|b_2\|_\infty\}$
32                  $F_1 = F_1 + b_1$, $F_2 = F_2 + b_2$
33                  if $c_1 + c_2 \leq$ tol × $\max\{\|F_1\|_\infty, \|F_2\|_\infty\}$, break, end
34                  $c_1 = c_2$
35              end
36              $F_1 = \eta F_1$, $F_2 = \eta F_2$, $b_1 = F_1$, $b_2 = F_2$
37          end
38          $y_{k+1} = F_1$
39          if balance, $y_{k+1} = Dy_{k+1}$, end
40          $\gamma_{k+1} = \sqrt{\|y_{k+1}\|_2}$
41          if $|\gamma_{k+1} - \gamma_k|/|\gamma_{k+1}| < 0.1$ or $k >$ it_max, $\gamma = \gamma_{k+1}$, break, end
42          $y_{k+1} = y_{k+1}/\|y_{k+1}\|_2$
43      end
44      estimate $\beta = \|e^{tA}\|_1$ using normest1:
45          when required, compute $e^{tA}v$, $(e^{tA})^*v$ via [2, Alg. 3.2, lines 12-23],
            and $[m, s]$ from line 7
46      compute $F = e^{tA}b$ using [2, Alg. 3.2, lines 12-23] and $[m, s]$ from line 7
47      $\kappa = (2\sqrt{n}\gamma\|tA\|_1 + \beta\sigma)/\|F\|_1$
```

Cost: Each iteration (lines 10-43) requires $2m^2s^2 + 2ms$ matrix-vector multiplications. The estimation of $\|e^{tA}b\|_1$ and $\|e^{tA}\|_1$ (lines 44-46) requires at most $10ms$ matrix-vector multiplications. Approximately $10n$ of allocatable memory is required.

Algorithm 4.1 can be adapted to return $e^{tA}b$ in double precision in addition to the condition estimate. This provides a saving of ms matrix-vector multiplications over computing the quantities separately. Before line 46, s and m should be recomputed using the double precision θ_m, by calling the subroutine parameters$(tA, 2^{-53})$. Then F in line 46 contains $e^{tA}b$ in double precision.

One of the strengths of Al-Mohy and Higham's algorithm is its ability to exploit level 3 BLAS operations for $e^{tA}B$, where $B \in \mathbb{C}^{n\times m}$, rather than requiring m separate calls to the algorithm for $e^{tA}b$. This strength extends naturally to Algorithm 4.1, by simply replacing vectors $b, b_1, b_2, F_1, F_2, G_1, G_2, y_k$ with $n \times m$ matrices and replacing $\|y_k\|_2$ with $\|Y_k\|_F$.

5 Numerical tests

For our approach to condition estimation to be deemed useful, numerical experiments must demonstrate two key points. First they must show that we can obtain accurate and reliable condition estimates. Second they must show that our approach is not prohibitively expensive (this could be caused by large numbers of iterations in the power method in Algorithm 3.3, or large numbers of matrix multiplications required to evaluate $L_f(A, L_f^\star(A, y_k b^*))b$ in Algorithm 3.4).

Our implementations and test codes were written in Python, with extensive use made of NumPy and SciPy [24]. The tests were performed on a 2.8GHz Intel Core i7 MacBook Pro. Our first two numerical experiments are based on $e^{tA}b$ and Algorithm 4.1 (the $e^{tA}b$ algorithm of [2] is available in SciPy 0.13.0 and later versions via `scipy.sparse.linalg.expm_multiply`). Our final experiment was a more general test of Algorithm 3.5 for functions other than the exponential.

Experiment 1: Accuracy of Algorithm 4.1 To investigate the accuracy of Algorithm 4.1, we computed κ, the upper bound in (2.6), exactly using Algorithm 3.1, which explicitly forms the Kronecker matrix and computes its 2-norm. The iteration in Algorithm 4.1 has a relative convergence tolerance of 0.1. In theory `normest1` can produce arbitrarily inaccurate 1-norm estimates, however in practice it is nearly always correct to within a factor 2. We therefore expect Algorithm 4.1 to give an estimate of κ with a relative error approximately less than 1. Test matrices were taken from the special matrix collection in SciPy's `linalg` module. The test matrices were taken to be of size 100, with the exception of the Hadamard matrix, which was of size 64. For matrices larger than 100×100 the computation of the exact value of κ, using Algorithm 3.1, becomes too expensive. For each test matrix, a randomly chosen b with elements uniformly distributed on $[-1, 1)$, and $b = [1, 1, \dots, 1]$ were tested, with $t \in \{0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10\}$. This resulted in a total of 112 tests. We summarize the results below.

- The largest number of iterations required in the iterative phase of Algorithm 4.1 in any of the tests was 4. No tests failed to converge.
- The relative error in the computation of κ using Algorithm 4.1 was less than 0.1 in each test.
- For each test we computed the product ms and also the product $m_d s_d$, where m_d and s_d are the scaling and truncation parameters that are required to evaluate $e^{tA}b$ in double precision. Each iteration within Algorithm 4.1 requires a number of matrix-vector products approximately proportional to $m^2 s^2$. The mean value of $(ms/m_d s_d)^2$ over all the tests was 0.42. We conclude that each iteration required

approximately 42 % of the number of matrix-vector multiplications that would have been required to evaluate $L_{\exp}(A, L^{\star}_{\exp}(A, y_k b^*))b$ in double precision.

- For each test we counted the total number of matrix-vector multiplications Π_{cond} taken to estimate the condition number, and the total number of matrix-vector multiplications $\Pi_{\exp}$ required to compute $e^{tA}b$ in double precision. Figure 1 shows the approximate quadratic relationship between the two. For these tests $\Pi_{\text{cond}}/\Pi^2_{\exp}$ took a mean value of 0.65, although this ranged between 0.1 and 1.4.

The main conclusions from this set of tests are that Algorithm 4.1 provides accurate condition number estimates, and that choosing m and s for half precision accuracy provides a significant computational saving. Nevertheless, the square relationship, in terms of matrix-vector multiplications, between the cost of forming $e^{tA}b$ and estimating its condition number is evident. This is a result of the method used to compute $L_{\exp}(A, L^{\star}_{\exp}(A, y_k b^*))b$.

Experiment 2: Sparse matrices with Algorithm 4.1 The action of a matrix function $f(A)b$ is typically of interest for large, sparse A. Our second set of numerical tests involve a selection of such matrices. Now, computing κ exactly using Algorithm 3.1 is usually too expensive. Instead, the aim is to demonstrate that, for 'realistic' test matrices, the condition number cond(exp, A, b, t) can be estimated without requiring a prohibitively large number of matrix-vector multiplications.

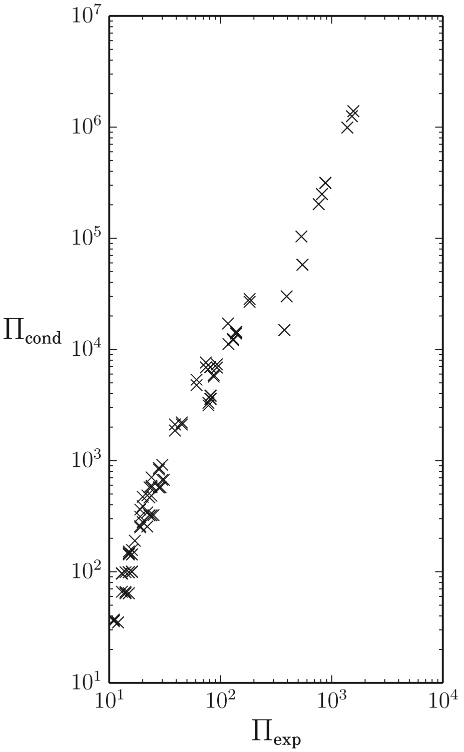

Fig. 1 The number of matrix-vector multiplications Π_{cond} required to estimate cond(exp, A, b, t) for the tests in Experiment 1 versus the number of matrix-vector multiplications $\Pi_{\exp}$ required to compute $e^{tA}b$ in double precision

We used the same tests as Al-Mohy and Higham [2]. The matrix poisson is a multiple of the finite difference discretization of the 2D Laplacian (this is most easily obtained by using the command -2500*gallery('poisson',99) in MATLAB). This was originally used as a test matrix by Trefethen, Weideman and Schmelzer [33]. The matrices orani678, bcspwr and gr_30_30 belong to the Harwell-Boeing collection and are available from the University of Florida Sparse Matrix Collection [10]. Finally, the matrices boeing767 and 3Ddiffuse were used as test matrices by Sheehan, Saad and Sidje [31]. The full problem details are:

- poisson, $n = 9801$, $t = 0.02, 1.0$, b obtained as described in [33];
- orani678, $n = 2529$, $t = 100$, $b = [1, 1, \ldots, 1]^T$;
- bcspwr10, $n = 5300$, $t = 10$, $b = [1, 0, \ldots, 0, 1]^T$;
- gr_30_30, $n = 900$, $t = 2$, $b = [1, 1, \ldots, 1]^T$;
- boeing767, $n = 55$, $t = 0.01, 0.10, 1.00$, $b = [1, 1, \ldots, 1]^T$;
- 3Ddiffuse, $n = 250, 000$, as described in [31, Section 4.3].

Table 2 shows the results of these tests. Note that the test $t = 1.0$ for the poisson matrix was not performed here. Al-Mohy and Higham found that their algorithm performed poorly for this test, with 47702 matrix-vector multiplications required. Estimating the condition number would require of the order 10^9 matrix-vector multiplications. For the tests shown, the value of $\Pi_{\text{cond}}/\Pi_{\text{exp}}^2$ varied between 0.05 and 1.19, with a mean of 0.5. For the boeing767 test, the matrix was small enough to allow us to check the condition estimates using Algorithm 3.1. For the remaining tests this was not possible due to the excessive memory requirements of Algorithm 3.1. Note that increasing t from 0.01 to 0.1 for the boeing767 matrix *decreases* the condition estimate before it then *increases* when $t = 1.0$. This behaviour is related to the well known ‘hump’ phenomenon for the matrix exponential [28].

Table 2 Estimating the condition number of $e^{tA}b$ for various large, sparse matrices detailed in Experiment 2

		Estimating cond(exp, A, b, t)						Computing $e^{tA}b$			
		κ	it	m	s	Π_{cond}	τ	m	s	Π_{exp}	τ
poisson		1232	3	52	14	1.1e6	428	54	21	1200	6.10
orani678		2.5e6	3	55	19	7.5e5	240	55	29	1248	0.41
bcspwr10		2.7e7	3	55	5	1.3e5	37.5	54	8	578	0.37
gr_30_30		668	3	30	4	2.8e4	3.25	48	4	155	0.01
	$t = 0.01$	2.2e10	3	50	3	9403	0.83	55	4	418	0.02
boeing767	$t = 0.1$	1.4e10	3	55	23	2.6e5	25.3	55	35	971	0.09
	$t = 1.0$	1.2e12	3	55	226	2.1e7	1230	55	349	5510	0.68
3Ddiffuse		31.4	2	17	1	1022	15.6	32	1	38	0.18

Π_{cond} and Π_{exp} denote the total number of matrix-vector multiplications required to compute cond(exp, A, b, t) and $e^{tA}b$ respectively; m and s denote the truncation and scaling parameters; it denotes the number of iterations of the power method, τ denotes the runtime in seconds and κ is the condition estimate

Experiment 3: Accuracy of Algorithm 3.5 for general functions The aim of this experiment was to demonstrate that the general method described in Algorithm 3.5 can be used for functions other than the exponential without requiring an excessive number of iterations to converge. In general the algorithm used to compute $f(A)b$ must be adapted to allow the quantity $L_f(A, L_f^\star(A, y_k b^*))b$ to be computed efficiently. Adapting various $f(A)b$ algorithms in this manner is the subject of future research, so for the purposes of this experiment, test matrices were limited to $n = 100$ and $L_f(A, L_f^\star(A, y_k b^*))b$ was computed using standard dense $f(A)$ algorithms.

We used the same set of test matrices as in Experiment 1 and the exact value of κ was computed using Algorithm 3.1. The following matrix functions were used:

- the principal matrix logarithm, available in SciPy as `scipy.linalg.logm`, computed using inverse scaling and squaring [3];
- the matrix sine, `scipy.linalg.sinm`, computed using the Schur-Parlett algorithm [8];
- the matrix cosine, `scipy.linalg.cosm`, also computed using the Schur-Parlett algorithm;
- the matrix square root, `scipy.linalg.sqrtm`, computed using a blocked version of the Björck-Hammarling algorithm [5, 12];
- the matrix cube root, `scipy.linalg.fractional_matrix_power`, computed using the Schur-Padé algorithm [21].

Discarding tests for which $f(A)b$ could not be computed (for example $\log(A)b$ if A is singular), this resulted in a total of 518 tests with condition numbers ranging from 1.0 to 3.9e12. Since we were using dense $f(A)$ algorithms, counting matrix-vector multiplications was not relevant to this experiment. Instead we were interested in the number of iterations required for convergence and the relative error of the condition estimate. We summarize the results below.

- 97.5 % of the tests converged within 4 iterations. The remaining tests converged within 6 iterations. No tests failed to converge.
- The relative error in the computation of κ using Algorithm 3.5 was less than 0.1 in 93.4 % of the tests. The relative error was less than 0.4 in 99.4 % of the tests. The remaining three tests had relative errors less than 0.6.

We conclude that, for general f, Algorithm 3.5 returns accurate condition estimates and does not require an excessive number of iterations to converge.

6 Conclusions and outlook

We have developed a general framework for estimating the condition number of $f(A)b$. Central to our approach is the use of the power method to estimate $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$. Our framework can be applied to algorithms that compute $f(A)b$ using combinations of matrix-vector multiplications and linear system solves. The number of matrix-vector multiplications or linear system solves required by our method is proportional to the square of the number required to compute $f(A)b$. Since only an order of magnitude condition estimate is usually required, a considerable amount of computation can be saved by relaxing the tolerances in whichever $f(A)b$ algorithm is used.

We applied our method to Al-Mohy and Higham's algorithm for $e^{tA}b$ [2]. We found that, in practice, very few iterations of the power method were required to estimate $\max_{\|\Delta A\|=1} \|L_{\exp}(A, \Delta A)b\|$, and that the total number of matrix-vector multiplications, Π_{cond}, required to estimate the condition number is typically approximately $0.6 \times \Pi_{\exp}^2$, where $\Pi_{\exp}$ is the number of matrix-vector multiplications required to compute $e^{tA}b$ in double precision. Experiments using dense $f(A)$ algorithms to estimate $\text{cond}(f, A, b, t)$ for a variety of different f confirmed that, in general, very few iterations of the power method are required to estimate $\max_{\|\Delta A\|=1} \|L_f(A, \Delta A)b\|$.

The quadratic relationship between Π_{cond} and $\Pi_{\exp}$ is due to the estimation of $L_f(A, L_f^\star(A, y_k b^*))b$. This behaviour will be present irrespective of which $f(A)b$ algorithm is used and is a result of avoiding the computation and storage of dense $O(n^2)$ quantities in the power method. It would be desirable to be able to compute $L_f(A, L_f^\star(A, y_k b^*))b$ more efficiently. In the case of $e^{tA}b$ and Algorithm 4.1 this could be done in a multicore setting by parallelizing some of the loops in the algorithm.

Further work on this topic will focus on applying our methods to other $f(A)b$ algorithms, such as Krylov subspace projection methods. This would involve developing implementations of the $f(A)b$ algorithms which allow the user to control the matrix-vector multiplications (a 'reverse communication' interface) and then modifying the algorithms to allow $f(A)b$ to be returned in lower precision so that $L_f(A, L_f^\star(A, y_k b^*))b$ can be computed efficiently by Algorithm 3.4. Alternative approaches to computing the condition number of $f(A)b$ also warrant investigation. For example, we were unable to adapt the standard 1-norm condition estimation method [19, Algorithm 3.22] to the $f(A)b$ problem in a manner which avoided the explicit use of $O(n^2)$ quantities, but this may still be possible. The approach of Kenney and Laub [26] suffers from the same problems, but it may still be of interest when only the nonzero elements of A should be perturbed.

Acknowledgments I am grateful to Nick Higham for carefully reading earlier drafts of this paper and making many valuable comments. I am also grateful to the referees for their suggestions.

References

1. Al-Mohy, A.H., Higham, N.J.: Computing the Fréchet derivative of the matrix exponential, with an application to condition number estimation. SIAM J. Matrix Anal. Appl. **30**(4), 1639–1657 (2009)
2. Al-Mohy, A.H., Higham, N.J.: Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. **33**(2), 488–511 (2011)
3. Al-Mohy, A.H., Higham, N.J.: Improved inverse scaling and squaring algorithms for the matrix logarithm. SIAM J. Sci. Comput. **34**(4), C153–C169 (2012)
4. Al-Mohy, A.H., Higham, N.J., Relton, S.D.: Computing the Fréchet derivative of the matrix logarithm and estimating the condition number. SIAM J. Sci. Comput. **35**(4), C394–C410 (2013)

5. Björck, Å., Hammarling, S.: A Schur method for the square root of a matrix. Linear Algebra Appl. **52/53**, 127–140 (1983)
6. Burrage, K., Hale, N., Kay, D.: An efficient implicit FEM scheme for fractional-in-space reaction-diffusion equations. SIAM J. Sci. Comput. **34**(4), A2145–A2172 (2012)
7. Chen, T.-Y., Demmel, J.W.: Balancing sparse matrices for computing eigenvalues. Linear Algebra Appl. **309**, 261–287 (2000)
8. Davies, P.I., Higham, N.J.: A Schur–Parlett algorithm for computing matrix functions. SIAM J. Matrix Anal. Appl. **25**(2), 464–485 (2003)
9. Davies, P.I., Higham, N.J.: Computing $f(A)b$ for matrix functions f. In: Boriçi, A., Frommer, A., Joó, B., Kennedy, A., Pendleton, B. (eds.): QCD and Numerical Analysis III, volume 47 of Lecture Notes in Computational Science and Engineering, pp. 15–24. Springer, Berlin (2005)
10. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Software **38**(1), 1 (2011)
11. Deadman, E., Higham, N.J.: Testing matrix function algorithms using identities. MIMS EPrint 2014.13, Manchester Institute for Mathematical Sciences, The University of Manchester, UK (2014)
12. Deadman, E., Higham, N.J., Ralha, R.: Blocked Schur algorithms for computing the matrix square root. In: Manninen, P., Öster, P. (eds.): Applied Parallel and Scientific Computing: 11th International Conference, PARA 2012, Helsinki, Finland, volume 7782 of Lecture Notes in Computer Science, pp. 171–182. Springer, Berlin (2013)
13. Fischer, T.M.: On the stability of some algorithms for computing the action of the matrix exponential. Linear Algebra Appl. **443**, 1–20 (2014)
14. Frommer, A., Simoncini, V.: Matrix functions. In: Schilders, W.H.A., Van der Vorst, H.A., Rommes, J. (eds.): Model Order Reduction: Theory, Research Aspects and Applications, volume 13 of Mathematics in Industry, pp. 275–303, Berlin (2008)
15. Gallopoulos, E., Saad, Y.: Efficient solution of parabolic equations by Krylov approximation methods. SIAM J. Sci. Statist. Comput. **13**(5), 1236–1264 (1992)
16. Güttel, S.: Rational Krylov approximation of matrix functions: Numerical methods and optimal pole selection. GAMM-Mitteilungen **36**(1), 8–31 (2013)
17. Hale, N., Higham, N.J., Trefethen, L.N.: Computing A^{α}, $\log(A)$, and related matrix functions by contour integrals. SIAM J. Numer. Anal. **46**(5), 2505–2523 (2008)
18. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition (2002)
19. Higham, N.J.: Functions of Matrices: Theory and Computation. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2008)
20. Higham, N.J., Edvin, D.: A catalogue of software for matrix functions. Version 1.0 MIMS EPrint 2014.8, Manchester Institute for Mathematical Sciences, The University of Manchester, UK (2014)
21. Higham, N.J., Lin, L.: An improved Schur–Padé algorithm for fractional powers of a matrix and their Fréchet derivatives. SIAM J. Matrix Anal. Appl. **34**(3), 1341–1360 (2013)
22. Higham, N.J., Tisseur, F.: A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. SIAM J. Matrix Anal. Appl. **21**(4), 1185–1201 (2000)
23. Hochbruck, M., Ostermann, A.: Exponential integrators. Acta Numerica **19**, 209–286, 5 (2010)
24. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001)
25. Kenney, C.S., Laub, A.J.: Condition estimates for matrix functions. SIAM J. Matrix Anal. Appl. **10**(2), 191–209 (1989)
26. Kenney, C.S., Laub, A.J.: Small-sample statistical condition estimates for general matrix functions. J. Sci. Comput. **15**(1), 36–61 (1994)
27. Knizhnerman, L.A.: Calculation of functions of unsymmetric matrices using Arnoldi's method. U.S.S.R. Comput. Maths. Math. Phys. **31**(1), 1–9 (1991)
28. Moler, C.B., Loan, C.V.: Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. SIAM Rev. **45**(1), 3–49 (2003)
29. Rice, J.: A theory of condition. SIAM J. Numer. Anal. **3**(2), 287–310 (1966)
30. Saad, Y.: Analysis of some Krylov subspace approximations to the matrix exponential operator. SIAM J. Numer. Anal. **29**(1), 209–228 (1992)
31. Sheehan, B.N., Saad, Y., Sidje, R.B.: Computing exp($-\tau A$)b with Laguerre polynomials. Electron. Trans. Numer. Anal. **37**, 147–165 (2010)

32. Sidje, R.B.: Expokit: A software package for computing matrix exponentials. ACM Trans. Math. Softw. **24**(1), 130–156 (1998)
33. Trefethen, L.N., Weideman, J.A.C., Schmelzer, T.: Talbot quadratures and rational approximations. BIT **46**(3), 653–670 (2006)
34. Van der Vorst, H.A.: An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A. J. Comput. Appl. Math. **18**(2), 249–263 (1987)