SIAM J. Matrix Anal. Appl.
Vol. 40, No. 1, pp. 210–234

STABLE COMPUTATION OF GENERALIZED MATRIX FUNCTIONS VIA POLYNOMIAL INTERPOLATION*

JARED L. AURENTZ†, ANTHONY P. AUSTIN‡, MICHELE BENZI§, AND VASSILIS KALANTZIS¶

Abstract. Generalized matrix functions (GMFs) extend the concept of a matrix function to rectangular matrices via the singular value decomposition. Several applications involving directed graphs, Hamiltonian dynamical systems, and optimization problems with low-rank constraints require the action of a GMF of a large, sparse matrix on a vector. We present a new method for applying GMFs to vectors based on Chebyshev interpolation. The method is matrix free and requires no orthogonalization and minimal additional storage. Comparisons against existing approaches based on Lanczos bidiagonalization demonstrate the competitiveness of our approach. We prove that our method is backward stable by generalizing the proof of the backward stability of Clenshaw's algorithm to the matrix case.

Key words. generalized matrix functions, Chebyshev polynomials, Clenshaw's algorithm, graph theory

AMS subject classifications. 65F60, 15A16, 05C50

DOI. 10.1137/18M1191786

1. Introduction.

First introduced in [22], generalized matrix functions (GMFs) extend the notion of matrix functions from square matrices to rectangular ones using the singular value decomposition (SVD). Although they are perhaps less well known than their "standard" counterparts, GMFs arise in a variety of applications, including communication metrics for directed graphs [3, 12], matrix exponentials of Hamiltonian systems [14, 15], including the graph wave equation [10, 27], and regularization of ill-posed problems [21]. For additional theory and applications of GMFs see, for instance, [1, 2, 28] and the references therein. In all these applications, the quantity of interest is the action of a GMF on a vector. For small matrices, one can proceed directly by computing the full SVD of the matrix. Algorithms based on Lanczos

*Received by the editors June 7, 2018; accepted for publication (in revised form) December 10, 2018; published electronically February 12, 2019.
http://www.siam.org/journals/simax/40-1/M119178.html

Funding: The work of the first author was supported by the Spanish Ministry of Economy and Competitiveness, through the Severo Ochoa Programme for Centres of Excellence in R&D (SEV-2015-0554). The work of the second author was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. The work of the third author was supported by the National Science Foundation (DMS-1719578). The work of the fourth author was supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Basic Energy Sciences programs (DE-SC0008877).

†Instituto de Ciencias Matemáticas, 28049 Madrid, Spain (jared.aurentz@icmat.es).

‡Department of Mathematics, Virginia Tech, Blacksburg, VA 24061 (apaustin@vt.edu).

§Scuola Normale Superiore, Piazza dei Cavalieri 7, 56126 Pisa, Italy (michele.benzi@sns.it).

¶IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (vkal@ibm.com).

bidiagonalization have been proposed in [4] for large and sparse matrices.

In this paper, we present a new method for applying a GMF of a large, sparse matrix to a vector. Our method, which is based on Chebyshev interpolation, is "matrix free"—it needs only a routine that computes the action of the matrix (and its transpose) on a vector—and uses only a small amount of additional memory, making it easy to parallelize and well suited to large-scale problems. Similar techniques have been used to accelerate the solution of large symmetric eigenvalue problems by using multicore and GPU processors [5, 6, 18]. We verify the efficacy of our method with numerical experiments, which show our method to be superior in terms of memory usage and, for certain problems, compute time.

We also prove that our method is backward stable by generalizing the proof of the backward stability of Clenshaw's algorithm to the matrix case. The proof we give can, with minimal modification, be used to establish backward stability for the Chebyshev interpolation methods that are so popular for computing eigenvalues and (matrix) functions of symmetric matrices. To the best of our knowledge, this is the first backward stability result for these methods to be established, and our analysis therefore fills an important gap in the literature. We verify this stability result with numerical experiments.

2. Functions of matrices. We begin by recalling some basic facts about both standard and generalized matrix functions. We then use these to establish the properties of GMFs that form the foundation for our algorithm.

2.1. Standard matrix functions. For simplicity, let A be an $n \times n$ Hermitian matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$, and write $A = Q\Lambda Q^*$ in eigendecomposed form, where Q is unitary and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Given a complex-valued function f, recall that the (standard) matrix function $f(A)$ is obtained by applying f to the eigenvalues of A: $f(A) = Qf(\Lambda)Q^*$, where $f(\Lambda) = \operatorname{diag}\big(f(\lambda_1), \ldots, f(\lambda_n)\big)$. Note that if $n = 1$, so that $A = \lambda_1$ is a scalar, then $f(A) = f(\lambda_1)$. In this sense, the definition of $f(A)$ can be viewed as an extension of f to matrix arguments.

Observe that if p is any polynomial that interpolates f in the eigenvalues $\lambda_1, \ldots \lambda_n$, then $p(A) = f(A)$. We refer to this result as the *polynomial interpolation theorem* for standard matrix functions and note that it can be used as the basis for a rigorous and more complete definition of $f(A)$ than the one given in the preceding paragraph; see [19, Chapter V], [24, p. 5].[1] Note also that $p(A) = A^j$ when $p(x) = x^j$ for any nonnegative integer j; thus, the definition of powers of A obtained via matrix functions is consistent with that obtained via repeated matrix multiplication.

This last observation means that, given p such that $p(A) = f(A)$, we can explicitly compute $f(A)$ once we have represented p in some basis. The problem with this approach is that we cannot find p without knowing all the eigenvalues of A, and this is not practical if A is large. Instead, we take p to be a polynomial that approximates f on some interval (or other set) that contains the eigenvalues of A; the better the approximation of p to f, the better the approximation of $p(A)$ to $f(A)$. If f is sufficiently smooth, finding a p that yields an accurate approximation is an easy task. This idea was pioneered by Druskin and Knizhnerman in [17]. More broadly, this idea forms the basis for Krylov subspace methods, polynomial preconditioning, and filtering techniques for large-scale eigenvalue problems.

[1] If A is diagonalizable, the same definition works, even if the eigenvector matrix is not unitary. For nondiagonalizable A, the polynomial will need to interpolate derivatives of f as well.

2.2. Generalized matrix functions. Just as standard matrix functions are defined by applying functions to eigenvalues, generalized matrix functions are defined by applying functions to singular values. Let B be an $m \times n$ matrix of rank $r \leq \min(m,n)$ with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and write $B = U\Sigma V^*$ in a (compact) SVD, where the $m \times r$ matrix U and $n \times r$ matrix V each have orthonormal columns and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$. Given a scalar function f, we define the generalized matrix function

$$f^\diamond(B) = Uf(\Sigma)V^*,$$

where $f(\Sigma) = \operatorname{diag}\big(f(\sigma_1), \ldots, f(\sigma_r)\big)$, a standard matrix function. If $B = 0$, we set $f^\diamond(B) = 0$. Note that in general $f^\diamond(B) \neq f(B)$ if B is square and indefinite; thus, GMFs are not strictly generalizations of their standard counterparts. Nevertheless, the terminology has stuck, and we continue to use it here.

In what sense is $f^\diamond$ an extension of the scalar function f to matrix arguments? If $B = \beta$ is a real scalar, then we can obtain an SVD of B by taking $U = \operatorname{sign}(\beta)$, $\Sigma = |\beta|$, and $V = 1$. We therefore have $f^\diamond(\beta) = f^\diamond(B) = \operatorname{sign}(\beta) f(|\beta|)$, and so for real scalars x,

$$f^\diamond(x) = \begin{cases} f(x), & x > 0, \\ 0, & x = 0, \\ -f(-x) & x < 0, \end{cases} \tag{1}$$

an *odd* function. More generally, $f^\diamond$ is odd as a function of matrices because if $U\Sigma V^*$ is an SVD of B, then $(-U)\Sigma V^*$ is an SVD of $-B$, and

$$f^\diamond(-B) = (-U)f(\Sigma)V^* = -\big(Uf(\Sigma)V^*\big) = -f^\diamond(B).$$

The reason this happens is that $f^\diamond(B)$ depends only on the positive singular values of B; therefore, only the values of f on the positive real axis matter. This inherent odd symmetry is a distinguishing feature of GMFs and will recur repeatedly throughout this paper.

Since our aim is to develop techniques for applying GMFs to vectors by using polynomials, we need a way to compute $p^\diamond(B)$ for polynomials p; and for it to be applicable to large matrices, it cannot rely explicitly on the SVD of B. The challenge here is that it is not generally true, for instance, that $p^\diamond(B) = B^j$ when $p(x) = x^j$, as would be the case for a standard matrix function. Indeed, B may be rectangular, so B^j may not even be defined. It is therefore not immediately obvious how to express a polynomial GMF at all, much less in a form that avoids reference to the SVD.

Nevertheless, if p is an *odd* polynomial, then $p^\diamond(B)$ can be easily computed by using just matrix multiplication by B and B^*. To see this, observe that if p is odd, then $p(x) = q(x^2)x$ for some polynomial q, and (see next sentence) we have

$$p^\diamond(B) = q(BB^*)B = Bq(B^*B),$$

where $q(BB^*)$ and $q(B^*B)$ are standard matrix functions. This can be seen by using the SVD of B:

$$q(BB^*)B = q(U\Sigma^2U^*)U\Sigma V^* = Uq(\Sigma^2)\Sigma V^* = Up(\Sigma)V^* = p^\diamond(B).$$

The equality involving $Bq(B^*B)$ may be established similarly.

The preceding discussion leads us to state the following theorem, which makes precise the correspondence between GMFs and odd scalar-valued functions.

THEOREM 2.1 (generalized polynomial interpolation theorem). *Let B be an $m \times n$ matrix, and let f be a scalar function. There is an odd polynomial p such that $p^\diamond(B) = f^\diamond(B)$.*

Proof. Take p to be any odd polynomial that interpolates the odd scalar-valued function defined in (1) at the singular values of B. ∎

As was the case with standard matrix functions, we cannot compute the polynomial p in the theorem without knowing the singular values of B. This is not practical if B is large. Instead, we will choose p to approximate f on an interval that contains the singular values. The theorem shows that we can restrict our attention to odd polynomials p, which gives us a means of computing $p^\diamond(B)$ without the SVD of B.

We close this section by observing that if p is the odd polynomial of minimal degree such that p annihilates B when applied to it in the generalized sense, then any odd power GMF of B can be written as an odd polynomial GMF of degree no larger than that of p by constructing an odd interpolant to the odd power in the roots of p. This parallels the fact that any power of a square matrix, taken in the standard sense, may be expressed as a polynomial function of the matrix of degree no larger than that of the matrix's minimal polynomial.

3. Computing generalized matrix functions.

We now describe our algorithm for computing GMFs in detail.

3.1. Chebyshev interpolation.

To build the polynomial approximations to f that form the basis for our method, we use *Chebyshev interpolation*—interpolation in Chebyshev points. Given a function f defined on $[-1, 1]$, the *degree-k Chebyshev interpolant* to f is the unique polynomial p_k of degree at most k that satisfies

$$p_k(x_i) = f(x_i), \qquad i = 0, \ldots, k,$$

where $x_i = \cos(i\pi/k)$, $i = 0, \ldots, k$, the $k + 1$ *Chebyshev points of the second kind*.

Chebyshev interpolants have two desirable properties. First, for smooth f, the error in the approximation $p_k \approx f$ decreases rapidly as k increases. More precisely, if $\|\cdot\|_\infty$ denotes the supremum norm on $[-1, 1]$ and if f is analytic in some open region of the complex plane containing $[-1, 1]$, then there exist $\rho > 1$ and a positive constant C such that $\|p_k - f\|_\infty < C\rho^{-k}$ [31, Chapter 8]. That is, for analytic f, the approximation error decays with k at a geometric rate.

The second property is that the expansion of p_k in the basis of Chebyshev polynomials of the first kind can be computed efficiently from the function values at the Chebyshev points by using the discrete cosine transform (DCT). Recall that the *Chebyshev polynomials of the first kind* are the polynomials T_i defined via the recurrence

$$T_0(x) = 1, \quad T_1(x) = x, \quad \text{and} \quad T_{i+1}(x) = 2xT_i(x) - T_{i-1}(x), \quad i \geq 1. \tag{2}$$

By appropriately choosing coefficients $\alpha_0, \ldots, \alpha_k$, we may write

$$p_k(x) = \sum_{i=0}^{k} \alpha_i T_i(x), \tag{3}$$

the *Chebyshev expansion* of p_k. The α_i are related to the function values $f(x_i)$ via a linear map that can be applied in $O(k \log k)$ time by using the DCT [31, Chapter 3].

3.2. Clenshaw's algorithm. A polynomial represented in a Chebyshev basis can be evaluated by using *Clenshaw's algorithm* [11], a generalization of Horner's method that can be applied to evaluate any expansion in which the basis is defined by a three-term recurrence. In this section, we develop a variant of this algorithm for evaluating the approximations $p^\diamond(B)$ to $f^\diamond(B)$ that we obtain from Chebyshev interpolation. We give an algorithm for computing the product $p^\diamond(B)w$ for a vector w, since this is what is needed in applications.

The results of section 2 tell us that for $p^\diamond(B)w$ to be computable via matrix multiplication with B (and B^*), p should be odd. Since T_i is even or odd according to whether i is even or odd, the even-indexed coefficients in the expansion (3) for odd p will be identically zero. Thus, we write

$$p(x) = \sum_{i=0}^{k} \alpha_{2i+1} T_{2i+1}(x), \tag{4}$$

where the degree of p is now $2k+1$.

There is a version of Clenshaw's algorithm tailored to evaluating expansions of the form (4) that avoids evaluating the even-degree Chebyshev polynomials [26, section 2.5, Problem 7]. One way to derive it is as follows. By substituting $2i-1$ for i and setting $j=2$ in the standard identity

$$2T_i(x)T_j(x) = T_{i+j}(x) + T_{|i-j|}(x), \tag{5}$$

we obtain a three-term recurrence for the odd-degree Chebyshev polynomials:

$$T_{2i+1}(x) = 2T_2(x)T_{2i-1}(x) - T_{|2i-3|}(x), \qquad i \geq 1. \tag{6}$$

Clenshaw's algorithm then yields the following recurrence for $p(x)$ when p is odd:

$$\gamma_{k+1} = \gamma_{k+2} = 0, \qquad \gamma_i = \alpha_{2i+1}x + 2T_2(x)\gamma_{i+1} - \gamma_{i+2}, \qquad i = k, \ldots, 0, \tag{7}$$

where $p(x) = \gamma_0$.

To convert this into an algorithm for $p^\diamond(B)w$, we need a three-term recurrence for the vectors $T^\diamond_{2i+1}(B)w$. Lemma 3.1 shows how to modify (6) using the SVD of B.

LEMMA 3.1 (three-term recurrence for $T^\diamond_{2i+1}(B)w$). *Let B be an $m \times n$ matrix, and let w be a vector of length n. The vectors $T^\diamond_{2i+1}(B)w$ satisfy the following recurrence relation:*

$$T^\diamond_{2i+1}(B)w = 2T_2\left(\sqrt{BB^*}\right) T^\diamond_{2i-1}(B)w - T^\diamond_{|2i-3|}(B)w, \qquad i \geq 1.$$

Proof. Let $B = U\Sigma V^*$ be the SVD of B. We have $T^\diamond_j(B)w = UT_j(\Sigma)V^*w$ and $T_2(\sqrt{BB^*}) = 2BB^* - I = 2U\Sigma^2U^* - I$. Therefore, the identity to be established is equivalent to

$$UT_{2i+1}(\Sigma)V^*w = 2(2U\Sigma^2U^* - I)UT_{2i-1}(\Sigma)V^*w - UT_{|2i-3|}(\Sigma)V^*w.$$

Factoring out U and V^*w on both sides and replacing $2\Sigma^2 - I$ with $T_2(\Sigma)$, we see that this will hold if

$$T_{2i+1}(\Sigma) = 2T_2(\Sigma)T_{2i-1}(\Sigma) - T_{|2i-3|}(\Sigma).$$

This is just (6) with Σ in place of x. Since Σ is square and diagonal, the equality holds, completing the proof. ∎

The following theorem shows how to modify Clenshaw's algorithm from (7) to compute $p^\diamond(B)w$ for odd p.

THEOREM 3.2 (Clenshaw's algorithm for generalized matrix functions). *Let B be an $m \times n$ matrix, and let p be an odd polynomial of degree $2k+1$ with Chebyshev expansion given by* (4). *Given a vector w of length n, we have $p^\diamond(B)w = g_0$, where g_0 is computed via the recurrence*

$$g_{k+1} = g_{k+2} = 0, \quad g_i = \alpha_{2i+1}v + 2(2BB^* - I)g_{i+1} - g_{i+2}, \quad i = k, \ldots, 0,$$

and $v = Bw$.

Proof. Apply Clenshaw's algorithm to the three-term recurrence in Lemma 3.1, and note that $T_2(\sqrt{BB^*}) = 2BB^* - I$. ∎

We emphasize that the recurrence in Theorem 3.2 uses only the action of B and B^* on vectors; there is no need to form the matrix BB^* explicitly. Therefore, for problems involving large sparse matrices, we can compute generalized matrix functions using only matrix-vector multiplication.

Note that one can arrive at Theorem 3.2 in a manner different from the one presented by taking the standard Clenshaw algorithm, zeroing the terms involving the even coefficients, and simplifying the recursion.

3.3. Scaling. The polynomial approximations we developed in section 3.1 are interpolants on the interval $[-1, 1]$; their accuracy away from that interval is not guaranteed. Therefore, if B has singular values outside of $[0, 1]$, our polynomial approximation to the GMF may no longer be accurate. To deal with this, we scale B by an upper bound β for the largest singular value σ_1 of B. Note that once β is computed, it can be used in approximating $f^\diamond(B)$ for multiple functions f.

One can compute an acceptable β in several ways; in our implementation, we use Lanczos bidiagonalization. Since this algorithm produces rapidly converging approximations to σ_1, we need only a few Lanczos steps to get an approximation to σ_1 sufficiently accurate to construct β. Moreover, since σ_1 is the only singular value of B that we need, we do not need to perform any (re)orthogonalization.

If one has access to the entries of B, an alternative way to choose β is to apply Gerschgorin's theorem to BB^*. Upper bounds obtained this way are, however, typically considerably less sharp than the bounds one can obtain through Lanczos bidiagonalization.

3.4. Summary of algorithm. A summary of our method for approximating $f^\diamond(B)w$ using polynomial interpolation is given in Algorithm 1.

Algorithm 1. Computing $f^\diamond(B)w$.

Require: Matrix B, function f, vector w, and an approximation tolerance $\varepsilon > 0$

(1) Compute β such that $\sigma_1 \leq \beta$ using Lanczos bidiagonalization, where σ_1 is the largest singular value of B.

(2) Let $h(x) = f^\diamond(x\beta)$—see (1)—and compute p such that $\|p - h\|_\infty < \varepsilon \|h\|_\infty$ using Chebyshev interpolation of h on $[-1, 1]$.

(3) Compute $p^\diamond(\beta^{-1}B)w$ using Clenshaw's algorithm (Theorem 3.2).

return $p^\diamond(\beta^{-1}B)w$

4. Backward stability. Clenshaw's algorithm (7) is a backward stable method for evaluating scalar polynomials. One might therefore expect that the variant of it given in Theorem 3.2 is backward stable as well. This is indeed the case; however, because of the noncommutativity of matrix-matrix and matrix-vector multiplication, the backward error assumes a nonstandard form. Since T_{2i+1} is odd, there is a degree-i polynomial q_i such that $T_{2i+1}(x) = q_i(x^2)x$. Thus, we can rewrite $p^\diamond(B)w$ as

$$p^\diamond(B)w = \left(\sum_{i=0}^{k} q_i(BB^*)\alpha_{2i+1}\right) Bw. \tag{8}$$

We will show that the version of Clenshaw's algorithm in Theorem 3.2 is backward stable in the sense that there exist an $m\times n$ matrix δB and $m\times m$ matrices $\{\delta A_{2i+1}\}_{i=0}^k$ such that

$$\mathrm{fl}\big(p^\diamond(B)w\big) = \left(\sum_{i=0}^{k} q_i(\widetilde{B}\widetilde{B}^*)\widetilde{A}_{2i+1}\right) \widetilde{B}w,$$

where $\widetilde{B} = B + \delta B$, $\widetilde{A}_{2i+1} = \alpha_{2i+1}I + \delta A_{2i+1}$, and $\mathrm{fl}\big(p^\diamond(B)w\big)$ is the output of Clenshaw's algorithm in floating-point arithmetic.

To state our result, we introduce the following notation. Let I be the $m \times m$ identity matrix, and let $M = 2BB^* - I$. Let $\{g_i\}_{i=0}^k$ be as in Theorem 3.2, let $\{\alpha_{2i+1}\}_{i=0}^k$ be as in (4), and define the block matrices

$$\mathbf{g} = \begin{bmatrix} g_k \\ \vdots \\ g_1 \\ g_0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \alpha_{2k+1}I \\ \alpha_{2k-1}I \\ \vdots \\ \alpha_1 I \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} I & & & & \\ -2M & I & & & \\ I & -2M & I & & \\ & I & -2M & I & \\ & & & & \ddots \end{bmatrix}.$$

Observe that $\mathbf{g}$ satisfies the linear system

$$\mathbf{L}\mathbf{g} = \mathbf{A}Bw, \tag{9}$$

and therefore, since $g_0 = p^\diamond(B)w$,

$$p^\diamond(B)w = \mathbf{e}_0^T\mathbf{g} = \mathbf{e}_0^T\mathbf{L}^{-1}\mathbf{A}Bw,$$

where $\mathbf{e}_0^T = \begin{bmatrix} 0 & \cdots & 0 & I \end{bmatrix}$. (We may invert $\mathbf{L}$ because it is square and lower triangular with all diagonal entries nonzero.) Let u be the *unit roundoff*. Given matrices X and Y, if there is a constant $C > 0$ such that $\|X\|_2 \le \big(Cu + O(u^2)\big)\|Y\|_2$, we say that X *is small relative to* Y and write $X \preceq Y$.

THEOREM 4.1 (backward stability of Clenshaw's algorithm). *Let B, w, and p be as in Theorem* 3.2*, and assume $\|B\|_2 \le 1$ and $u < 1$. Let $\{\widehat{g}\}_{i=0}^k$ be the perturbed versions of the iterates $\{g_i\}_{i=0}^k$ computed by using Theorem* 3.2 *in floating-point arithmetic. There exist a matrix $\widetilde{B} = B + \delta B$, $\delta B \preceq B$, and a block matrix $\delta\widetilde{\mathbf{L}}$ such that*

$$(\widetilde{\mathbf{L}} + \delta\widetilde{\mathbf{L}})\widehat{\mathbf{g}} = \mathbf{A}\widetilde{B}w, \tag{10}$$

where

$$\widehat{\mathbf{g}} = \begin{bmatrix} \widehat{g}_k \\ \vdots \\ \widehat{g}_1 \\ \widehat{g}_0 \end{bmatrix}, \quad \widetilde{\mathbf{L}} = \begin{bmatrix} I & & & & \\ -2\widetilde{M} & I & & & \\ I & -2\widetilde{M} & I & & \\ & I & -2\widetilde{M} & I & \\ & & & & \ddots \end{bmatrix},$$

and $\widetilde{M} = 2\widetilde{B}\widetilde{B}^* - I$. *Furthermore, if* $||\delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1}||_2 < 1$, *there exist* $m \times m$ *matrices* $\{\delta A_{2i+1}\}_{i=0}^k$, $\delta A_{2i+1} \preceq \sum_{j=i}^k |j-i+1||\alpha_{2j+1}|$, $i = 0, \ldots, k$, *such that*

$$\widetilde{\mathbf{L}}\widehat{\mathbf{g}} = \widetilde{\mathbf{A}}\widetilde{B}w, \tag{11}$$

where

$$\widetilde{\mathbf{A}} = \begin{bmatrix} \alpha_{2k+1}I + \delta A_{2k+1} \\ \alpha_{2k-1}I + \delta A_{2k-1} \\ \vdots \\ \alpha_1 I + \delta A_1 \end{bmatrix}.$$

In other words, the iterates $\widehat{g}_i$ (and, in particular, $\widehat{g}_0$) obtained by running the algorithm of Theorem 3.2 in floating-point arithmetic exactly satisfy a perturbed version of the linear system (9), and if the perturbation is sufficiently small, it can be pushed onto the coefficients $\{\alpha_{2i+1}\}_{i=0}^k$ in the sense described in the first paragraph of this section. Since the proof of this result is lengthy and technical, we defer it to the appendix.

The backward error δB in B found by Theorem 4.1 is small relative to B. The backward error δA_{2i+1} in the coefficient α_{2i+1}, on the other hand, is asserted to be small only relative to a weighted sum involving all higher-degree coefficients. The following corollary shows that when the coefficients decay geometrically, δA_{2i+1} is in fact small relative to α_{2i+1} on its own.

COROLLARY 4.2. *If there exists* ρ, $0 \le \rho < 1$, *such that the coefficients* $\{\alpha_{2i+1}\}_{i=0}^k$ *satisfy*

$$|\alpha_{2k+1}| \le \rho|\alpha_{2k-1}| \le \cdots \le \rho|\alpha_1|,$$

then $\delta A_{2i+1} \preceq \alpha_{2i+1}$, $i = 0, \ldots, k$.

Proof. By Theorem 4.1,

$$\delta A_{2i+1} \preceq \sum_{j=i}^{k} |j-i+1||\alpha_{2j+1}| \le |\alpha_{2i+1}| \sum_{j=0}^{k-i} |j+1|\rho^j.$$

That $\delta A_{2i+1} \preceq \alpha_{2i+1}$ now follows from the fact that the sum is finite. ∎

5. Numerical experiments. Theorem 4.1 tells us that Clenshaw's algorithm is stable and thus that Algorithm 1 is a reliable method for approximating generalized matrix functions. In this section we report on several numerical experiments that verify the error bounds in Theorem 4.1, and we compare Algorithm 1 with the Lanczos-based methods presented in [4].

5.1. Backward stability. In our first set of experiments, we used extended precision to verify numerically that the algorithm of Theorem 3.2 is backward stable and that the error bounds predicted by Theorem 4.1 and Corollary 4.2 are correct.

We fixed the dimensions of the matrix B to be $m = 3$ and $n = 2$, and we set the degree of p to 201, that is, $k = 100$. We then chose B and w randomly with entries normally distributed with mean 0 and standard deviation 1. We picked the coefficients of p randomly from a normal distribution with mean 1 and standard deviation .01. Thus, we had a set of coefficients such that each satisfied $|\alpha_{2i+1} - 1| \le .04$, 99.99% of the time. By Theorem 4.1,

$$\|\delta A_{2i+1}\|_2 \preceq \sum_{j=i}^{k} |j-i+1||\alpha_{2j+1}| \approx |\alpha_{2i+1}| \sum_{j=0}^{k-i} |j+1| = O\left((k-i+1)^2\right)|\alpha_{2i+1}|,$$

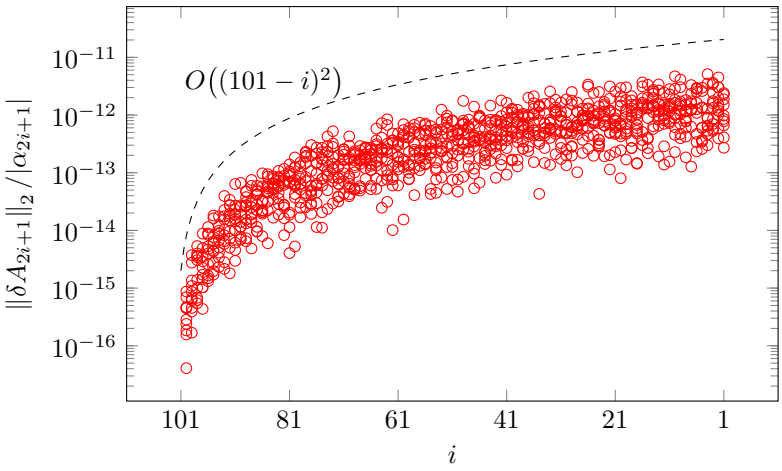

FIG. 1. *Relative backward error from running Clenshaw's algorithm for polynomials whose coefficients are all close to* 1. *Note that the value of* i *decreases from left to right.*

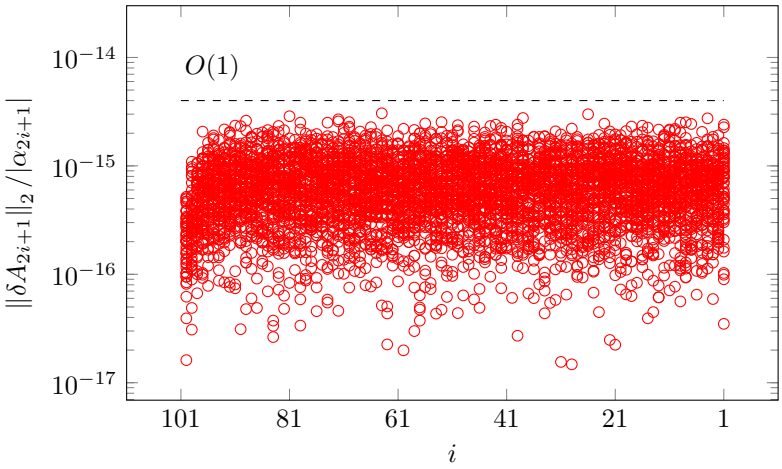

FIG. 2. *Relative backward error from running Clenshaw's algorithm for polynomials whose coefficients decay at a geometric rate,* $|\alpha_{2i+1}| \approx (1/2)^i$. *Note that the value of* i *decreases from left to right.*

so the relative backward error in each coefficient α_{2i+1} grows quadratically with $k - i + 1$. Figure 1 shows the distributions of the relative errors in each coefficient from performing this experiment 10 times together with the asymptotic quadratic error bound. The experimental results neatly match our theoretical predictions.

Next, we performed the same experiment but multiplied the same randomly generated coefficients by the geometrically decaying factor $\rho^i = (1/2)^i$. Corollary 4.2 asserts that the relative backward errors should be uniformly bounded. Figure 2 shows the distributions of the relative errors in each coefficient from performing the experiment 10 times. The uniform boundedness is readily apparent.

5.2. Comparison with Lanczos-based methods. We compared the runtime for approximating $f^\diamond(B)w$ using Algorithm 1 and the Lanczos-based method from [4,

section 5.4]. The Lanczos-based method uses Golub–Kahan bidiagonalization [20, section 10.4] to compute an approximate truncated SVD of B. After k steps of the bidiagonalization process started with the vector w, we have an $m \times (k+1)$ matrix U_k and an $n \times (k+1)$ matrix V_k, both with orthonormal columns, a $(k+1) \times (k+1)$ diagonal matrix Σ_k, and an $m \times n$ matrix R_k such that

$$B = U_k \Sigma_k V_k^* + R_k. \tag{12}$$

The vector $f^\diamond(B)w$ is then approximated as

$$f^\diamond(B)w \approx U_k f(\Sigma_k) V_k^* w.$$

Since the approximation is a linear combination of the columns of U_k and since the columns of U_k form an orthonormal basis for the Krylov subspace

$$\mathcal{K}_{k+1}(BB^*, Bw) = \operatorname{span}\{Bw, (BB^*)Bw, \ldots, (BB^*)^k Bw\},$$

there exists an odd polynomial p_k of degree $2k+1$ such that

$$p_k^\diamond(B)w = U_k f(\Sigma_k) V_k^* w.$$

Thus, the Lanczos-based method also constructs an odd polynomial approximation to $f^\diamond(x)$. Since the Lanczos polynomial p_k satisfies

$$p_k = \operatorname*{arg\,min}_{\substack{q \text{ odd} \\ \deg(q) \le 2k+1}} \left\| q^\diamond(B)w - f^\diamond(B)w \right\|_2$$

we know that for a given k, the Lanczos-based method will be at least as accurate as Algorithm 1 when the errors are measured in the Euclidean norm.

It follows that the Lanczos-based method should need a lower-degree polynomial than Algorithm 1 to attain a given level of error. Nevertheless, Algorithm 1 may still be more efficient because its cost scales better as k increases. The dominant operations in Algorithm 1 are the matrix-vector multiplications and vector additions in Clenshaw's algorithm. Assuming that matrix-vector multiplication takes $O(m+n)$ flops, as is generally the case for sparse matrices, Algorithm 1 has an asymptotic cost of $O\big((m+n)k\big)$ flops. For the Lanczos-based method to be stable, one must ensure the orthogonality of the columns of U_k and V_k, which we do via full reorthogonalization. Taking k steps of the algorithm therefore requires $O\left((m+n)k^2 + k^3\right)$ flops, and this becomes increasingly expensive as k gets large.

5.2.1. Experimental setup. For a given matrix B, vector w, function f, and tolerance ε, we computed polynomial approximations $p^\diamond(B)w$ to $f^\diamond(B)w$ such that

$$\left\| p^\diamond(B)w - f^\diamond(B)w \right\|_2 \le \varepsilon \left\| f^\diamond(B) \right\|_2 \|w\|_2$$

using three methods: *Chebyshev*, *Lanczos dynamic*, and *Lanczos fixed*.

The *Chebyshev* method is Algorithm 1. We estimated the leading singular value of B using Golub–Kahan bidiagonalization without reorthogonalization with w as the starting vector, exiting when the leading singular value had converged to a relative tolerance of 10^{-3}. We computed the upper bound β by adding the residual norm to this estimate, as is done in [18].[2] We then used the Chebfun software package

[2]This technique worked robustly for our set of test matrices, but for others (or for other choices of starting vector) it may fail with the coarse convergence tolerance that we have used [32]. If this happens, one solution is to take more bidiagonalization steps, although this increases the cost.

[16] to construct a Chebyshev interpolant p of degree large enough to ensure that $\|p - h\|_\infty < \varepsilon \|h\|_\infty$ and evaluated $p^\diamond(B)w$ using Clenshaw's algorithm. All runtimes we report for this method include the times required to estimate the leading singular value and to construct the interpolant.

In the *Lanczos dynamic* method, we ran Golub–Kahan bidiagonalization with full reorthogonalization using w as the starting vector. We checked convergence at every step, a process that requires computing the SVD of a $k \times k$ matrix, and considered the result converged when

$$\left\|p_{k-1}^\diamond(B)w - p_k^\diamond(B)w\right\|_2 < \varepsilon \left\|p_k^\diamond(B)w\right\|_2 .$$

Since the optimal (minimal) value of k needed to reach convergence cannot be known ahead of time, we consider this a realistic way to implement this method in practice.

The *Lanczos fixed* method is the same as Lanczos dynamic except that it uses the optimal k found during the Lanczos dynamic computation. Thus, the Lanczos fixed method yielded exactly the same results as the Lanczos dynamic method but ran more quickly, since it did not check convergence at each step.

The experiments were performed on the Mesabi Linux cluster at the Minnesota Supercomputing Institute.[3] We performed each experiment using MATLAB running on a single compute node with two Intel Haswell E5-2680v3 processors and 64 GB of memory.

Note that we use only odd functions f in all of our experiments. While our choices for f are driven by the applications considered, it is reasonable to ask whether it is worthwhile to consider even f as well. To this end, we recall from section 2.2 that since $f^\diamond(B)$ is defined using the SVD of B, the behavior of f on the nonnegative real axis is all that matters. Therefore, the only material difference between an odd f and an even f is the behavior of f at the origin, as any function defined on $[0, \infty)$ that assumes the value 0 at the origin has an odd extension to the entire real line. Noting that the definition of GMFs excludes contributions from zero singular values, this means that there is effectively no such thing as an "even GMF."

5.2.2. Graph metrics. One of the key applications of GMFs is to communication metrics for directed graphs. In [3, 4], it is shown that $f^\diamond(B)w$ with $w = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$ and $f(x) = \sinh(x)$ can be used to obtain a vector of *total hub communicabilities* for each node in the directed graph with adjacency matrix B. The early article [25] proposes a measure of centrality based on the matrix resolvent. This approach is reviewed in modern linear-algebraic terms in [9], and [4] explicitly connects it to GMFs. Here, the function to be applied is $f_\alpha(x) = \alpha x/\big(1 - (\alpha x)^2\big)$, where $\alpha \in (0, \sigma_1^{-1})$, and σ_1 is the largest singular value of B. Again, the relevant choice for w is the vector with all entries equal to 1.

Using the three methods (Chebyshev, Lanczos dynamic, and Lanczos fixed), we computed $f^\diamond(B)w$ for each of these metrics for each of the test matrices in Table 1. These matrices are adjacency matrices for a variety of directed graphs, and all of them are publicly available as part of the SuiteSparse Matrix Collection [13].[4] We used a tolerance of $\varepsilon = 10^{-5}$. For the resolvent-based metrics, we used values of $0.125/\sigma_1$, $0.500/\sigma_1$, and $0.850/\sigma_1$ for α, following the experiments in [4].

Figures 3–6 plot the ratios of the runtimes of each method to the runtime of the Chebyshev method. Table 2 reports the degrees of the polynomials computed in the

[3] https://www.msi.umn.edu

[4] https://sparse.tamu.edu

TABLE 1
Test matrices for computing graph metrics, along with their dimension m and number of nonzero entries. (All matrices are square, $m = n$.)

Matrix	Dimension	Number of nonzeros
`SNAP/soc-Slashdot0902`	82,168	948,464
`SNAP/amazon0302`	262,111	1,234,877
`SNAP/web-Stanford`	281,903	2,312,497
`SNAP/web-NotreDame`	325,729	1,497,134
`SNAP/amazon0312`	400,727	3,200,440
`SNAP/amazon0505`	410,236	3,356,824
`SNAP/amazon0601`	403,394	3,387,388
`SNAP/web-BerkStan`	685,230	7,600,595
`SNAP/web-Google`	916,428	5,105,039

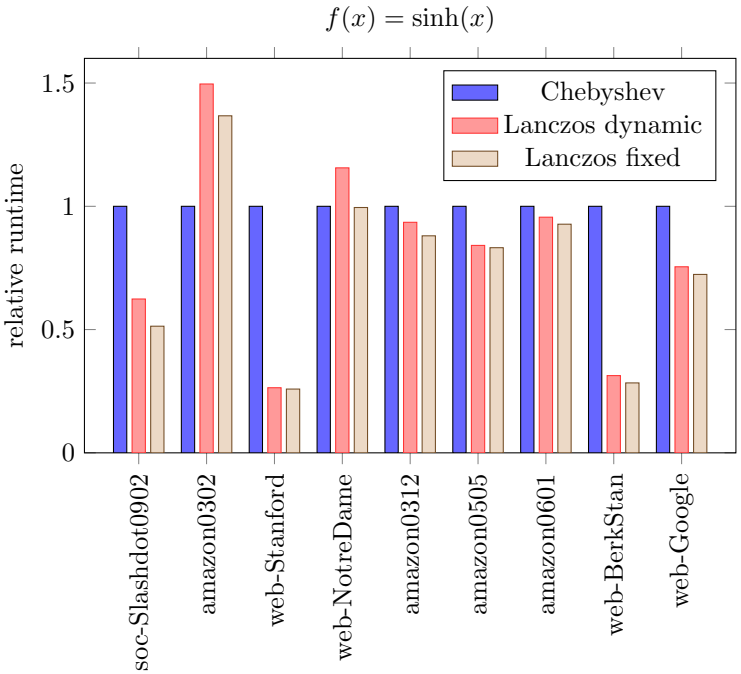

FIG. 3. *Plot of relative runtimes for approximating* $\sinh^\diamond(B)w$ *using the three methods.*

Chebyshev method and the number of steps taken by the Lanczos-based methods. With a few isolated exceptions, the Lanczos-based methods generally converged more quickly than the Chebyshev method for these examples. The reason is that the functions $\sinh(x)$ and $f_\alpha(x)$ are largest on the largest singular values of B, and only a few steps of the Lanczos bidiagonalization process are needed for the approximate SVD $B \approx U_k \Sigma_k V_k^*$ from (12) to accurately capture the part of B associated with these singular values.

This fact is observed in [4, section 5.4] and can be understood by considering the extreme case where $f(\sigma_1) = 1$ but $f(\sigma) \approx 0$ for all other singular values σ of B. For $f^\diamond(B) \approx U_k f(\Sigma_k) V_k$ to be a good approximation in this case, the k-step Lanczos

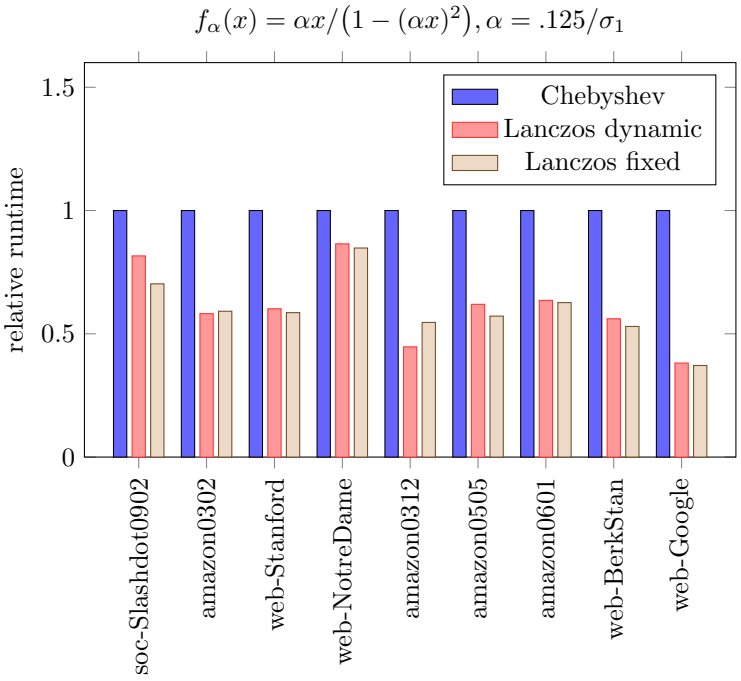

FIG. 4. *Plot of relative runtimes for approximating* $f_\alpha^\diamond(B)w$, $\alpha = .125/\sigma_1$, *using the three methods.*

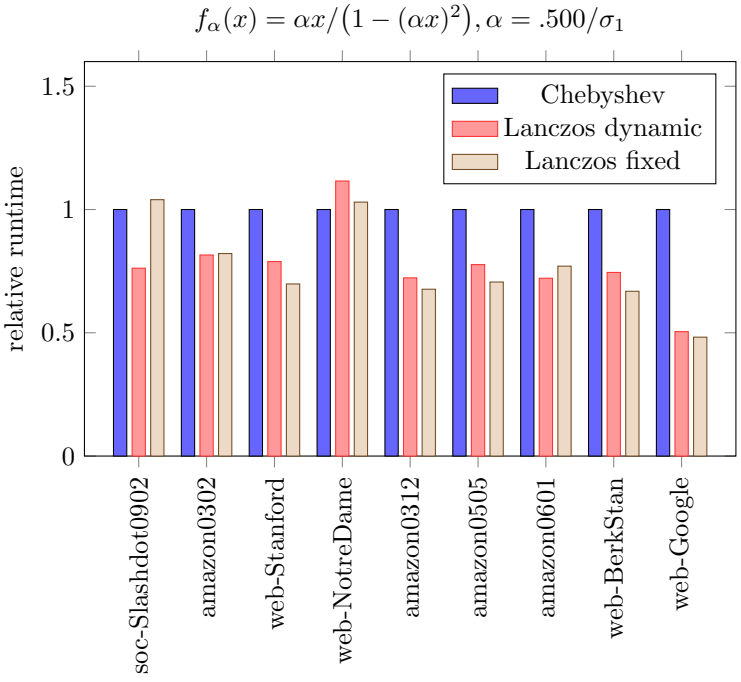

FIG. 5. *Plot of relative runtimes for approximating* $f_\alpha^\diamond(B)w$, $\alpha = .500/\sigma_1$, *using the three methods.*

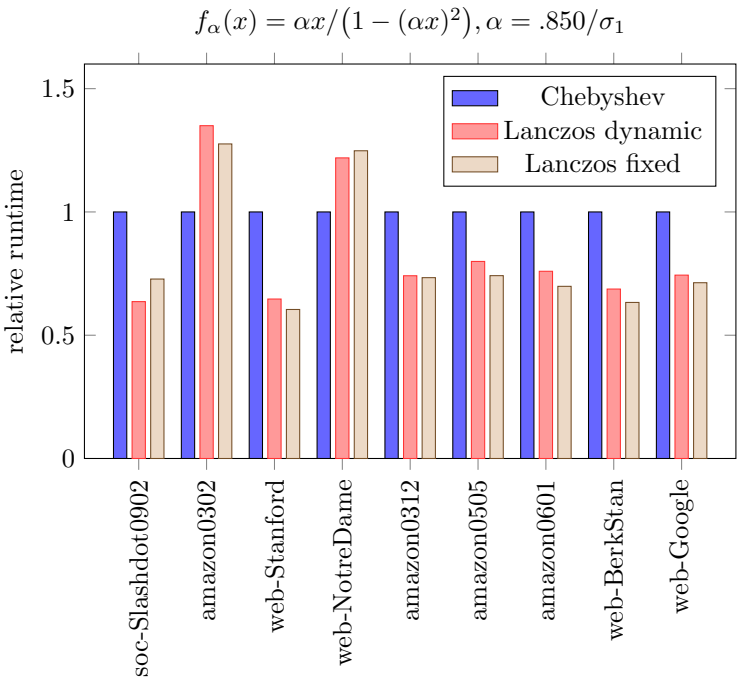

FIG. 6. *Plot of relative runtimes for approximating $f_\alpha^\diamond(B)w$, $\alpha = .850/\sigma_1$, using the three methods.*

TABLE 2

Degrees of polynomial approximations used by the Chebyshev method and Lanczos step counts for the examples in section 5.2.2. Note that for f_α, the fact that α is scaled by σ_1 ensures that the degree chosen by the Chebyshev method will not vary from matrix to matrix, as the polynomial that gets computed in each case is exactly the same.

Matrix	sinh		f_α, $\alpha = 0.125/\sigma_1$		f_α, $\alpha = 0.005/\sigma_1$		f_α, $\alpha = 0.850/\sigma_1$	
	Degree	Steps	Degree	Steps	Degree	Steps	Degree	Steps
`SNAP/soc-Slashdot0902`	51	8	5	4	9	5	21	7
`SNAP/amazon0302`	21	13	5	4	9	6	21	11
`SNAP/web-Stanford`	93	8	5	4	9	6	21	7
`SNAP/web-NotreDame`	59	11	5	4	9	5	21	8
`SNAP/amazon0312`	33	13	5	4	9	6	21	9
`SNAP/amazon0505`	35	12	5	4	9	6	21	9
`SNAP/amazon0601`	35	13	5	4	9	6	21	9
`SNAP/web-BerkStan`	117	11	5	4	9	6	21	8
`SNAP/web-Google`	47	16	5	4	9	6	21	11

process need only produce an estimate $\sigma_{k,1}$ to σ_1 that is good enough to guarantee $f(\sigma_{k,1}) \approx f(\sigma_1)$ to within the desired tolerance. Since Lanczos approximates leading singular values rapidly, this can usually be achieved for small k. In practice, f need not be so extreme for this to make a difference: the function f_α considered above with $\alpha = 0.125/\sigma_1$ is nearly linear over the relevant domain. For a function that grows exponentially like $\sinh(x)$, the convergence can be very rapid.

Recall that the Chebyshev method requires an estimate of the leading singular

value and that we find this using the same process that underlies the Lanczos-based methods (but without reorthogonalization). The Chebyshev method and the Lanczos-based ones therefore all start in the same way. Because of this, it is difficult for the Chebyshev method to beat the Lanczos-based ones in situations like the one described in the preceding paragraph: by the time the Chebyshev method has the information it needs to build the polynomial, the Lanczos-based ones are most of the way toward finding the answer.[5]

It is instructive to consider the alternative extreme case where $f(\sigma_\ell) = 1$, $\ell > 1$, where σ_ℓ is the ℓth singular value of B, but $f(\sigma) \approx 0$ for all other singular values σ, including $\sigma = \sigma_1$. Now the Lanczos-based methods will need to approximate σ_ℓ instead of σ_1, and depending on how the singular values of B are distributed, this can be (and typically is) a much more difficult task: it takes many more Lanczos steps to approximate interior singular values than extreme ones. In contrast, the Chebyshev method is insensitive to the singular value distribution: the polynomial approximation is constructed to be uniformly accurate at all singular values simultaneously. It thus stands to reason that the Chebyshev method should see greater success in applications where the function assumes significant values on the interior singular values, and we will see an example of this in the following section.

The Chebyshev method is notably more efficient than the Lanczos-based methods for the hyperbolic sine function applied to the `amazon0302` matrix. We suspect that this is a consequence of the fact that the leading singular values of this matrix are more strongly clustered than those of the other matrices. In addition to interior singular values, the Lanczos process has difficulty isolating singular values that are clustered together. Matrices with clustered or slowly decaying singular values therefore present another class of problems for which the Chebyshev method may be more effective.

5.2.3. Graph wave equation. The GMF $\sin^\diamond$ appears in applications involving exponential integrators for Hamiltonian systems, such as various forms of the graph wave equation [10, 27]. Given an undirected graph, we consider the equation

$$\ddot{u}(t) = -Lu(t),$$

where L is the graph Laplacian and u is a function on (the vertices of) the graph, represented as a vector. Wave equations of this form may be obtained via a method-of-lines discretization of the wave equation for a metric graph using finite differences. If B is any (oriented) incidence matrix for the graph, then $L = BB^T$, and if we define $\dot{v}(t) = B^T u(t)$, we have

$$\begin{bmatrix} \dot{u}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & -B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} u(t) \\ v(t) \end{bmatrix}.$$

Taking the matrix exponential, we are led to the solution

$$\begin{bmatrix} u(t) \\ v(t) \end{bmatrix} = \begin{bmatrix} \cos(t\sqrt{BB^T}) & -\sin^\diamond(tB) \\ \sin^\diamond(tB^T) & \cos(t\sqrt{B^TB}) \end{bmatrix} \begin{bmatrix} u(0) \\ v(0) \end{bmatrix},$$

[5] It is tempting to say that the Chebyshev method will have an advantage or disadvantage versus the Lanczos-based methods according to whether the leading singular value estimate is done quickly or slowly. In reality, the situation is more complicated. For instance, for the sinh function applied to the `SNAP/web-BerkStan` matrix, the Chebyshev method spent only about 9% of its total time estimating σ_1, but it was still slower than the Lanczos-based methods due to the rather large value of σ_1 (approximately 675) and the rapid growth of the sinh function combining to drive up the degree of the polynomial approximation. On the other hand, for the sinh function applied to the `SNAP/amazon0302` matrix, the Chebyshev method spent 95% of its time estimating the leading singular value but beat the Lanczos methods by 40–50%.

TABLE 3
Test matrices for integrating the graph wave equation, along with the dimensions (number of nodes × number of edges) of their incidence matrices.

Graph	Number of nodes	Number of edges
`SNAP/ca-HepTh`	9,877	25,998
`Newman/as-22july06`	22,963	48,436
`Gleich/usroads-48`	126,146	161,950
`SNAP/as-Skitter`	1,696,415	11,095,298
`DIMACS10/delaunay_n24`	16,777,216	50,331,601

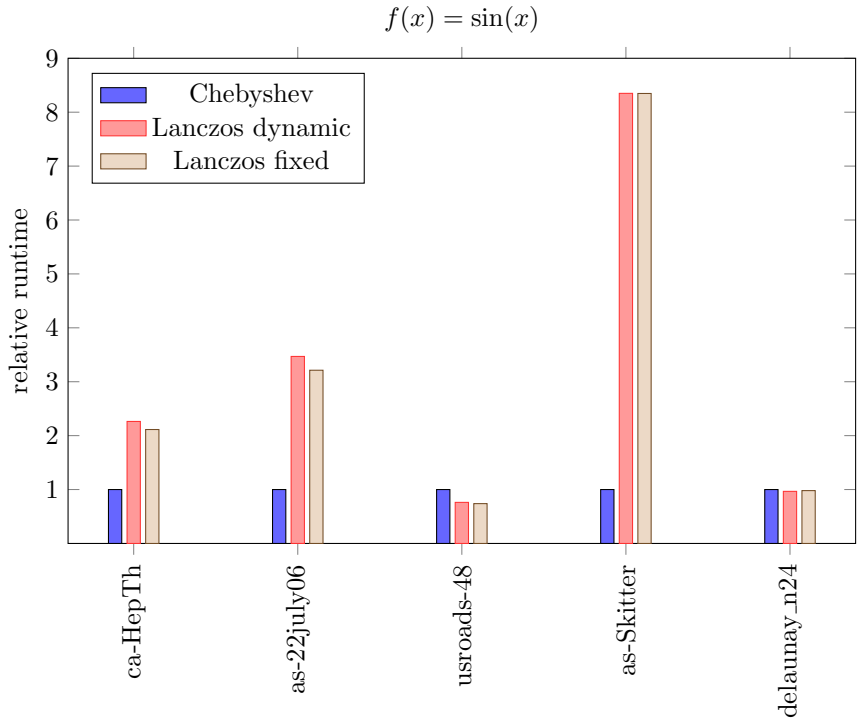

FIG. 7. *Plot of relative runtimes for approximating* $\sin^\diamond(B)w$ *using the three methods.*

in which $\sin^\diamond(tB)$ appears prominently. In the special case $u(0) = 0$, the solution is precisely $u(t) = -\sin^\diamond(tB)v(0)$.

We took five graphs whose adjacency matrices belong to the SuiteSparse collection and constructed the corresponding incidence matrices. The graphs are listed in Table 3. Note that each incidence matrix is rectangular. We then computed approximations to $\sin^\diamond(tB)w$, $w = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$, $t = 1, 4$, using each of the three methods with an error tolerance $\varepsilon = 10^{-5}$.

The results are displayed in Figures 7–8, which, like Figures 3–6, display the ratio of the runtime of each method to that of the Chebyshev method. In this experiment, the Chebyshev method fared much better, attaining speedups over the Lanczos-based methods by factors of approximately 9 and 17 for the `SNAP/as-Skitter` matrix for $t = 1$ and $t = 4$, respectively. Speedups for the other matrices are more modest but

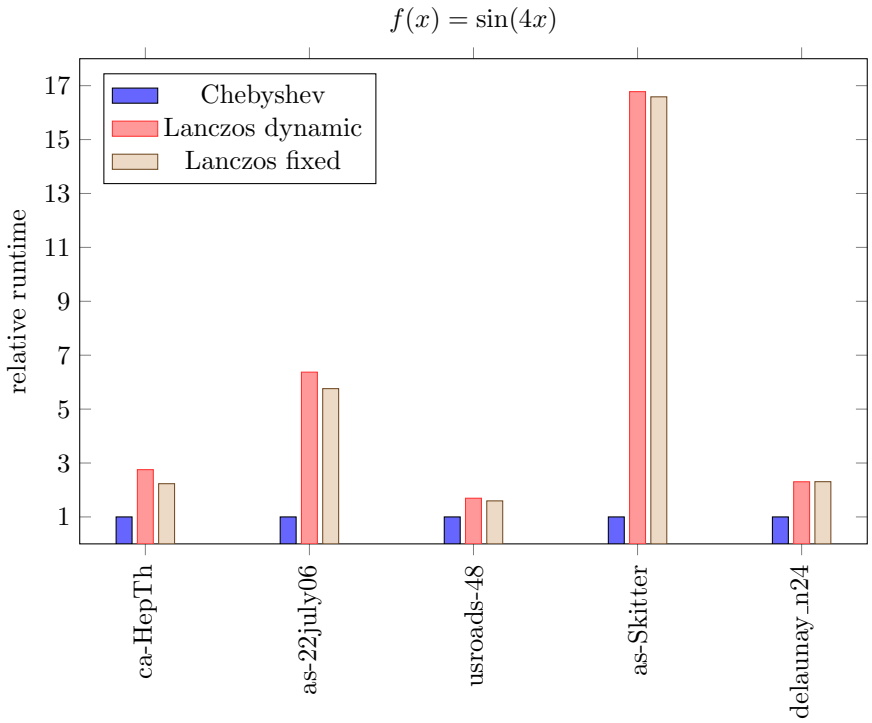

FIG. 8. *Plot of relative runtimes for approximating* $\sin^\diamond(4B)w$ *using the three different methods.*

still significant. The Chebyshev method is slower than the Lanczos-based methods only for the `Gleich/usroads-48` and `DIMACS10/delaunay_n24` matrices for $t = 1$, and even then only by a marginal amount.

The reason for this success is that, unlike the functions used in the experiments involving graph metrics, $\sin(x)$ and $\sin(4x)$ oscillate over the entire range of singular values, so the contributions to the results stemming from the interior singular values are significant. This drives up the degrees of the polynomial approximations needed by each of the methods to achieve the desired error tolerance. As explained in section 5.2, this situation favors the Chebyshev method because it scales better than the Lanczos-based ones as the degree increases.

We finish this example by noting that once a Chebyshev interpolant is constructed, it can be used to apply a GMF to many different vectors, e.g., if one wishes to solve the graph wave equation for multiple initial conditions. The multiple vectors can be blocked and processed simultaneously for greater efficiency.

It is also possible to handle multiple initial conditions with the Lanczos-based methods; however, it is not as simple. As described, these methods must be run from scratch for each initial condition. It is still possible to take advantage of the greater efficiency offered by blocking the matrix-vector multiplies; however, the additional orthogonalization costs make this an expensive proposition. Block Lanczos methods [4, 7] offer another approach; however, their convergence characteristics are more complicated than those of the single-vector methods, and the extra orthogonalization costs will still be high.

TABLE 4

Comparison of minimum memory requirements for the Chebyshev and Lanczos-based methods for $f(x) = \sin(x)$. The last column shows the ratio of memory required for the Lanczos-based method to that for the Chebyshev method.

	Chebyshev		Lanczos		
Matrix	Degree	Memory	Steps	Memory	Ratio
`SNAP/ca-HepTh`	19	703 KB	10	2.9 MB	4.1
`Newman/as-22july06`	69	1.3 MB	25	14.3 MB	10.6
`Gleich/usroads-48`	11	4.9 MB	6	13.8 MB	2.8
`SNAP/as-Skitter`	221	280 MB	88	9.0 GB	32.3
`DIMACS10/delaunay_n24`	15	1.3 GB	7	3.6 GB	2.8

5.2.4. Memory usage. Though the Chebyshev method may not always outperform the Lanczos-based methods in compute time, it consistently outperforms them in memory usage. The dominant memory cost in the Chebyshev method comes from Clenshaw's algorithm, which in our implementation requires 3 vectors of length equal to the number of columns of the matrix and 1 vector of length equal to the number of rows.[6] Crucially, the memory requirements for the Chebyshev method are independent of the degree of the polynomial being applied. The Lanczos-based methods, on the other hand, need N vectors of each size, where N is the number of Lanczos steps. Thus, if the matrix is size $m \times n$, the Lanczos-based methods will require approximately $N(m+n)/(m+3n)$ times as much memory as the Chebyshev method, ignoring the memory required to store the matrix itself. Thus, the ratio of memory usage between the Lanczos-based methods and the Chebyshev method should grow linearly with the number of Lanczos steps.

For large matrices, the difference can be substantial, even when N is small. Tables 4–5 list the amount of storage needed for the vectors in each of the experiments involving the graph wave equation conducted in the previous section, computed using the vector counts in the preceding paragraph and assuming the use of double precision (8-byte) floating-point numbers. These figures are thus lower bounds on the memory requirements for the methods. The tables also display the degree of the polynomial computed in the Chebyshev method, the number of steps needed by the Lanczos-based methods, and the ratio of the memory usage of the Chebyshev method to that of the Lanczos-based methods.

In all cases, the Chebyshev method used well less than half of the memory needed by the Lanczos-based methods. In the extreme case, the Chebyshev method beat the Lanczos-based methods in memory usage by a factor of nearly 90 when applying $\sin(4x)$ to the `SNAP/as-Skitter` matrix; that is, the Chebyshev method needed only a little more than 1.1% of the memory needed by the Lanczos-based methods. These savings are clearly significant and may make the Chebyshev method worth considering even when the Lanczos-based methods might be superior in runtime.

It is possible to improve both the memory usage and the runtime of the Lanczos-based methods by using the one-sided variant of Lanczos bidiagonalization [29]. In this variant, one explicitly orthogonalizes only the "short" Lanczos vectors in the bidiagonalization recurrence, leaving the orthogonality of the "long" Lanczos vectors

[6] When estimating the leading singular value with Golub–Kahan bidiagonalization, we need only two vectors—one each of length equal to each dimension—thanks to the fact that we do not need to perform reorthogonalization.

TABLE 5

Comparison of minimum memory requirements for the Chebyshev and Lanczos-based methods for $f(x) = \sin(4x)$. The last column shows the ratio of memory required for the Lanczos-based method to that for the Chebyshev method.

	Chebyshev		Lanczos		
Matrix	Degree	Memory	Steps	Memory	Ratio
`SNAP/ca-HepTh`	51	703 KB	24	6.9 MB	9.8
`Newman/as-22july06`	229	1.3 MB	60	34.3 MB	25.4
`Gleich/usroads-48`	23	4.9 MB	12	27.7 MB	5.6
`SNAP/as-Skitter`	811	280 MB	240	24.6 GB	87.8
`DIMACS10/delaunay_n24`	37	1.3 GB	17	9.1 GB	6.8

to be enforced implicitly. The long vectors need not be stored; they can be regenerated on demand using the recurrence. The disadvantage of this approach is that the orthogonality of the long vectors may not be adequately enforced if the matrix is not sufficiently well conditioned, leading to numerical instability.

While the one-sided variant can improve the performance of the Lanczos-based methods substantially, it is not necessarily sufficient to make them superior to the Chebyshev method. The Lanczos-based methods will still typically use more memory (a factor of $N \min(m, n)/(m + 3n)$ more, in the notation of the first paragraph in this subsection) than the Chebyshev method, and for large problems, the orthogonalization costs will still often be large enough for the Chebyshev method to enjoy a substantial advantage in terms of runtime for GMFs for which interior singular values play a significant role.

6. Conclusions. We presented a new method for computing the action of a generalized matrix function on a vector that is based on polynomial interpolation in Chebyshev points. We proved, for the first time in the literature, that Clenshaw's algorithm for computing the action of a polynomial function (expressed as a Chebyshev series) of a matrix on a vector is backward stable and verified this result numerically. We showed experimentally that our method competes with Lanczos-based methods for working with the GMFs that arise in computing graph metrics and that our method is a generally superior alternative to Lanczos-based methods for exponential integration of the graph wave equation. Finally, we showed that our method is considerably more efficient than Lanczos-based methods in memory usage, even in circumstances where Lanczos-based methods have the advantage in compute time.

Appendix A. Proof of Theorem 4.1. Our proof of Theorem 4.1 is similar to the one given for the backward stability of Clenshaw's algorithm in the scalar case in [30]. Much of the analysis carries over to the vector case, with a few exceptions that arise from the noncommutativity of matrix-matrix and matrix-vector multiplication. This noncommutativity is the reason that Theorem 4.1 pushes the error onto both the matrix B and the coefficients $\{\alpha_{2i+1}\}_{i=0}^{k}$ instead of just the latter.

Recall from section 4 that the iterates $\{g_i\}_{i=0}^{k}$ generated by the algorithm of Theorem 3.2 for computing $p^{\diamond}(B)w$ are the solution to the block-Toeplitz linear system (9), which involves B, w, and the coefficients of p. In our proof, we will need explicit bounds on the entries of $\mathbf{L}^{-1}$. We can compute $\mathbf{L}^{-1}$ explicitly using a recurrence for the even-degree Chebyshev polynomials. Substituting $i = 2i$ and $j = 2$ into the

identity (5) gives

$$T_{2(i+1)}(x) = 2T_2(x)T_{2i}(x) - T_{2|i-1|}(x), \qquad i \geq 0.$$

In Theorem 3.2, we saw that the matrix $M = 2BB^* - I$ can be rewritten as $M = T_2(\sqrt{BB^*})$. Since $\sqrt{BB^*}$ is square, we can substitute it into this recurrence to obtain

$$T_{2(i+1)}(\sqrt{BB^*}) = 2T_2(\sqrt{BB^*})T_{2i}(\sqrt{BB^*}) - T_{2|i-1|}(\sqrt{BB^*}), \qquad i \geq 0. \tag{13}$$

Now, let $\mathbf{T}$ be the block-Toeplitz matrix

$$\mathbf{T} = \begin{bmatrix} T_0(\sqrt{BB^*}) & & & \\ T_2(\sqrt{BB^*}) & T_0(\sqrt{BB^*}) & & \\ T_4(\sqrt{BB^*}) & T_2(\sqrt{BB^*}) & T_0(\sqrt{BB^*}) & \\ \vdots & & & \ddots \end{bmatrix}.$$

Since both $\mathbf{L}$ and $\mathbf{T}$ are block-Toeplitz and lower-triangular, the product $\mathbf{LT}$ is also block-Toeplitz and lower-triangular (see, e.g., [8, Chapter 2]). Thus, to compute $\mathbf{LT}$, we need to compute only the first block column. Using (13) and the definition of $\mathbf{L}$, we can compute this explicitly:

$$\begin{bmatrix} I & & & \\ -2T_2(\sqrt{BB^*}) & I & & \\ I & -2T_2(\sqrt{BB^*}) & I & \\ & & & \ddots \end{bmatrix} \begin{bmatrix} T_0(\sqrt{BB^*}) \\ T_2(\sqrt{BB^*}) \\ T_4(\sqrt{BB^*}) \\ \vdots \end{bmatrix} = \begin{bmatrix} T_0(\sqrt{BB^*}) \\ -T_2(\sqrt{BB^*}) \\ 0 \\ \vdots \end{bmatrix}.$$

Since $T_0(\sqrt{BB^*}) = I$ and $T_2(\sqrt{BB^*}) = 2BB^* - I = M$, the product $\mathbf{LT}$ simplifies to

$$\mathbf{LT} = \begin{bmatrix} I & & & & \\ -M & I & & & \\ & -M & I & & \\ & & -M & I & \\ & & & & \ddots \end{bmatrix}.$$

Finally, since $\mathbf{L}^{-1} = \mathbf{T}(\mathbf{LT})^{-1}$,

$$\mathbf{L}^{-1} = \begin{bmatrix} T_0(\sqrt{BB^*}) & & & \\ T_2(\sqrt{BB^*}) & T_0(\sqrt{BB^*}) & & \\ T_4(\sqrt{BB^*}) & T_2(\sqrt{BB^*}) & T_0(\sqrt{BB^*}) & \\ \vdots & & & \ddots \end{bmatrix} \begin{bmatrix} I & & & \\ M & I & & \\ M^2 & M & I & \\ \vdots & & & \ddots \end{bmatrix}. \tag{14}$$

In our argument, we will require the fact that matrix-vector multiplication and vector addition are backward stable [23]. This means that for a matrix B and vectors v and w, there exist matrices $\delta B \preceq B$ and δI, $||\delta I||_2 \leq u$, such that $\mathrm{fl}(Bw) = (B+\delta B)w$ and $\mathrm{fl}(v+w) = (I+\delta I)(v+w)$, where $\mathrm{fl}(\cdot)$ indicates that an operation is carried out in floating-point arithmetic.

Since our error bounds involve block matrices, we find it convenient to introduce some notation. Given a block matrix $\mathbf{H}$,

$$\mathbf{H} = \begin{bmatrix} H_{11} & H_{12} & \cdots \\ H_{21} & H_{22} & \\ \vdots & & \ddots \end{bmatrix},$$

with all block entries square (such as $\mathbf{L}$, $\mathbf{T}$, and $\mathbf{A}$), we denote by $|\mathbf{H}|$ the block matrix

$$|\mathbf{H}| = \begin{bmatrix} \|H_{11}\|_2 I & \|H_{12}\|_2 I & \cdots \\ \|H_{21}\|_2 I & \|H_{22}\|_2 I & \\ \vdots & & \ddots \end{bmatrix}.$$

It is straightforward to show using the definition of matrix-matrix multiplication that this operation is entrywise submultiplicative:

$$|\mathbf{L}^{-1}\mathbf{A}| \dot{\leq} |\mathbf{L}^{-1}||\mathbf{A}|,$$

where $\dot{\leq}$ denotes entrywise inequality.

We are now ready to prove Theorem 4.1, which we do in two steps. In the first step, we show in Lemma A.1 that there exists δB, as well as a block matrix $\delta\widetilde{\mathbf{L}}$, such that $\widehat{\mathbf{g}}$ is the exact solution of the perturbed linear system

$$\left(\widetilde{\mathbf{L}} + \delta\widetilde{\mathbf{L}}\right)\widehat{\mathbf{g}} = \mathbf{A}\widetilde{B}w.$$

In the second step we rewrite the above system as

$$\widetilde{\mathbf{L}}\widehat{\mathbf{g}} = \left(I + \delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1}\right)^{-1}\mathbf{A}\widetilde{B}w,$$

which when compared with (11) implies that $\widetilde{\mathbf{A}} = (I + \delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1})^{-1}\mathbf{A}$. We then use this to bound the error $\delta\mathbf{A} = \widetilde{\mathbf{A}} - \mathbf{A}$.

LEMMA A.1 (rounding error in Clenshaw's algorithm). *Let B, w, and $\{\alpha_{2i+1}\}_{i=0}^{k}$ be as in Theorem 3.2, and assume $\|B\|_2 \leq 1$ and $u < 1$. Then there exist a matrix $\widetilde{B} = B + \delta B$, $\delta B \preceq B$, a constant C, and a matrix $\delta\widetilde{\mathbf{L}}$ such that*

$$\left(\widetilde{\mathbf{L}} + \delta\widetilde{\mathbf{L}}\right)\widehat{\mathbf{g}} = \mathbf{A}\widetilde{B}w$$

and

$$|\delta\widetilde{\mathbf{L}}| \dot{\leq} \left(Cu + O(u^2)\right)\begin{bmatrix} I & & & & \\ I & I & & & \\ I & I & I & & \\ & I & I & I & \\ & & & & \ddots \end{bmatrix}.$$

Proof. Clenshaw's algorithm begins with the computation of $v = Bw$. By assumption, there exists $\delta B \preceq B$ such that $\hat{v} = \mathrm{fl}(v) = \mathrm{fl}(Bw) = (B + \delta B)w = \widetilde{B}w$. Since $\widehat{v}$ is computed once and used in all subsequent iterations, the matrix $\widetilde{B}$ is fixed and has no dependence on i.

To analyze the ith step, recall that $\widehat{g}_{i+1}$ and $\widehat{g}_{i+2}$ are the outputs of steps $i+1$ and $i+2$ in the algorithm. The vector $\widehat{g}_i$ is computed via

$$\widehat{g}_i = \mathrm{fl}(g_i) = \mathrm{fl}\Big(\mathrm{fl}(\alpha_{2i+1}\widehat{v}) + \mathrm{fl}\big(2\mathrm{fl}(M\widehat{g}_{i+1}) - \widehat{g}_{i+2}\big)\Big).$$

We begin with the innermost floating-point operation $\mathrm{fl}(M\widehat{g}_{i+1})$ and work outward.

The computation of $\hat{w}$ introduces a backward error in B. Since M is defined in terms of B, computing $\mathrm{fl}(M\widehat{g}_{i+1})$ would introduce new perturbations in B, making it

impossible to push a single error onto B. We can avoid this by redefining M in terms of $\widetilde{B}$ and then pushing any errors created by using $\widetilde{B}$ onto the matrix M. Substituting $B = \widetilde{B} - \delta B$ gives $M = 2(\widetilde{B} - \delta B)(\widetilde{B} - \delta B)^* - I$. Using this, we evaluate $\mathrm{fl}(M\widehat{g}_{g+1})$:

$$\mathrm{fl}\big(M\widehat{g}_{i+1}\big) = \mathrm{fl}\bigg(2\mathrm{fl}\Big((\widetilde{B} - \delta B)\mathrm{fl}\big((\widetilde{B} - \delta B)^*\widehat{g}_{i+1}\big)\Big) - \widehat{g}_{i+1}\bigg).$$

Replacing the floating-point operations with the appropriate perturbations gives

$$\mathrm{fl}(M\widehat{g}_{i+1}) = (I + \delta I)\big(2(\widetilde{B} - \delta B + \delta B_l)(\widetilde{B} - \delta B + \delta B_r)^*\widehat{g}_{i+1} - \widehat{g}_{i+1}\big),$$

where $\delta I \preceq I$ and $\delta B_l, \delta B_r \preceq B$. Expanding the right-hand side and using the definition of M, we have

$$\mathrm{fl}(M\widehat{g}_{i+1}) = (\widetilde{M} + \delta\widetilde{M}'_{i+1})\widehat{g}_{i+1},$$

where $\delta\widetilde{M}'_{i+1} = \delta I\widetilde{M} + 2(I + \delta I)\big((\delta B_l - \delta B)\widetilde{B}^* + \widetilde{B}(\delta B_r - \delta B)^* + (\delta B_l - \delta B)(\delta B_r - \delta B)^*\big)$. Since $\widetilde{B} = B + \delta B$ and $\|B\|_2 \le 1$, we know that $||\widetilde{M}||_2 \le 1 + 4||\delta B||_2 + O(u^2)$, which implies that $\delta\widetilde{M}'_{i+1} \preceq I$.

Next, we substitute $\mathrm{fl}(M\widehat{g}_{i+1})$ into the definition of $\widehat{g}_i$ and expand the floating-point operations using the appropriate perturbations:

$$\widehat{g}_i = (I + \delta I_1)\left(\alpha_{2i+1}(I + \delta I_2)\widehat{v} + (I + \delta I_3)\big(2(\widetilde{M} + \delta\widetilde{M}'_{i+1})\widehat{g}_{i+1} - \widehat{g}_{i+2}\big)\right),$$

where $||\delta I_1||_2, ||\delta I_2||_2, ||\delta I_3||_2 \le u$. We remove all perturbations of the product $\alpha_{2i+1}\widehat{v}$ by taking inverses, which is possible since $u < 1$:

$$(I + \delta I_2)^{-1}(I + \delta I_1)^{-1}\widehat{g}_i = \alpha_{2i+1}\widehat{v} + (I + \delta I_2)^{-1}(I + \delta I_3)(2(\widetilde{M} + \delta\widetilde{M}'_{i+1})\widehat{g}_{i+1} - \widehat{g}_{i+2}).$$

Let $\delta I_{i,i} = (I + \delta I_2)^{-1}(I + \delta I_1)^{-1} - I$ and $\delta I_{i,i+2} = (I + \delta I_2)^{-1}(I + \delta I_3) - I$, and rewrite $\widehat{g}_i$ as follows:

$$(I + \delta I_{i,i})\widehat{g}_i = \alpha_{2i+1}\widehat{v} + 2(\widetilde{M} + \delta\widetilde{M}_{i+1})\widehat{g}_{i+1} - (I + \delta I_{i,i+2})\widehat{g}_{i+2}, \tag{15}$$

where $\delta\widetilde{M}_{i+1} = \delta I_{i,i+2}\widetilde{M} + (I + \delta I_{i,i+2})\delta\widetilde{M}'_{i+1}$. It is straightforward to show that $\delta I_{i,i}, \delta I_{i,i+2}, \delta\widetilde{M}_{i+1} \preceq I$ using series expansions of $(I + \delta I_1)^{-1}$ and $(I + \delta I_2)^{-1}$.

Now, let

$$\delta\widetilde{\mathbf{L}} = \begin{bmatrix} \delta I_{k,k} & & & & & \\ -2\delta\widetilde{M}_{k-1} & \delta I_{k-1,k-1} & & & & \\ \delta I_{k-2,k} & -2\delta\widetilde{M}_{k-2} & \delta I_{k-2,k-2} & & & \\ & & \ddots & & & \\ & & \delta I_{0,2} & -2\delta\widetilde{M}_0 & \delta I_{0,0} \end{bmatrix}.$$

From (15) we have

$$(\widetilde{\mathbf{L}} + \delta\widetilde{\mathbf{L}})\widehat{\mathbf{g}} = \mathbf{A}\widetilde{B}w, \tag{16}$$

and since each block entry of $\delta\widetilde{\mathbf{L}}$ is small relative to I, there is a constant C such that

$$|\delta\widetilde{\mathbf{L}}| \dot{\le} \big(Cu + O(u^2)\big)\begin{bmatrix} I & & & & \\ I & I & & & \\ I & I & I & & \\ & I & I & I & \\ & & & & \ddots \end{bmatrix}.$$

This completes the proof. ∎

Lemma A.1 says that the rounding errors produced when running Clenshaw's algorithm in floating-point arithmetic result in iterates $\widehat{g}_i$ that satisfy a "nearby" linear system. The remainder of the argument consists of showing how to push this error onto the coefficients α_{2i+1} when the norm of $\delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1}$ is less than 1. Whether this holds depends on the constant C from Lemma A.1 and on the norm of $\widetilde{\mathbf{L}}^{-1}$, which in turn depend on the size and sparsity of B and the degree of the polynomial p. We experienced no stability issues in the experiments of section 5.1, which were carried out in double precision ($u \approx 10^{-16}$) using matrices with no more than 10^7 nonzeros and polynomials formed by interpolating smooth functions.

Proof of Theorem 4.1. We can rewrite (16) as

$$\widetilde{\mathbf{L}}\widehat{\mathbf{g}} = (I + \delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1})^{-1}\mathbf{A}\widetilde{B}w.$$

Since $\delta\widetilde{\mathbf{L}} \preceq I$ and $\|\delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1}\|_2 < 1$, we can take a first-order expansion of the inverse:

$$\widetilde{\mathbf{L}}\widehat{\mathbf{g}} = \left(I - \delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1} + O(u^2)\right)\mathbf{A}\widetilde{B}w = (\mathbf{A} + \delta\mathbf{A})\,\widetilde{B}w,$$

where $\delta\mathbf{A} = (-\delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1} + O(u^2))\mathbf{A}$. We have the following upper bound on $|\delta\mathbf{A}|$:

$$|\delta\mathbf{A}| = \left|\left(-\delta\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^{-1} + O(u^2)\right)\mathbf{A}\right| \dot{\leq} \left(|\delta\widetilde{\mathbf{L}}||\widetilde{\mathbf{L}}^{-1}| + O(u^2)\right)|\mathbf{A}|.$$

To bound the entries of $|\delta\mathbf{A}|$, we need a bound on the entries of $|\delta\widetilde{\mathbf{L}}||\widetilde{\mathbf{L}}^{-1}|$. Since $\widetilde{\mathbf{L}}$ has the same structure as $\mathbf{L}$, we can compute its entries explicitly:

$$|\widetilde{\mathbf{L}}^{-1}| \dot{\leq} \begin{bmatrix} \left\|T_0\left(\sqrt{\widetilde{B}\widetilde{B}^*}\right)\right\|_2 I & & \\ \left\|T_2\left(\sqrt{\widetilde{B}\widetilde{B}^*}\right)\right\|_2 I & \left\|T_0\left(\sqrt{\widetilde{B}\widetilde{B}^*}\right)\right\|_2 I & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} I & & \\ ||\widetilde{M}||_2 I & I & \\ ||\widetilde{M}||_2^2 I & ||\widetilde{M}||_2 I & I \\ \vdots & & \ddots \end{bmatrix}.$$

Now, since $\widetilde{B} = B + \delta B$, we know that

$$\left\|\sqrt{\widetilde{B}\widetilde{B}^*}\right\|_2 = \|\widetilde{B}\|_2 \leq 1 + \|\delta B\|_2,$$

and thus the largest eigenvalue of $\sqrt{\widetilde{B}\widetilde{B}^*}$ is bounded by $1 + \|\delta B\|_2$. The Chebyshev polynomials are uniformly bounded by 1 on $[-1, 1]$. We can get a first-order bound for $T_{2i}(1 + \|\delta B\|_2)$ by taking the Taylor expansion of $T_{2i}(x)$ at $x = 1$ and using the fact that $T'_{2i}(1) = 2i$:

$$\left\|T_{2i}\left(\sqrt{\widetilde{B}\widetilde{B}^*}\right)\right\|_2 \leq 1 + 2i\,\|\delta B\|_2 + O(u^2).$$

Furthermore, from the proof of Lemma A.1 we know that $||\widetilde{M}||_2 \leq 1+4||\delta B||_2+O(u^2)$, which implies $||\widetilde{M}||_2^i \leq 1 + 4i||\delta B||_2 + O(u^2)$. Hence, there exists a constant C such that

$$|\widetilde{\mathbf{L}}^{-1}| \dot{\leq} \left(C + O(u)\right)\begin{bmatrix} I & & \\ I & I & \\ \vdots & & \ddots \end{bmatrix}\begin{bmatrix} I & & \\ I & I & \\ \vdots & & \ddots \end{bmatrix} = \left(C + O(u)\right)\begin{bmatrix} I & & \\ 2I & I & \\ 3I & 2I & I \\ \vdots & & \ddots \end{bmatrix}.$$

By Lemma A.1, there exists a constant C' such that

$$|\delta\widetilde{\mathbf{L}}| \dot{\leq} \left(C'u + O(u^2)\right) \begin{bmatrix} I & & & & \\ I & I & & & \\ I & I & I & & \\ & I & I & I & \\ & & & & \ddots \end{bmatrix},$$

and combining this with the bound on $|\widetilde{\mathbf{L}}^{-1}|$ gives

$$\begin{aligned} |\delta\widetilde{\mathbf{L}}||\widetilde{\mathbf{L}}^{-1}| &\dot{\leq} \left(CC'u + O(u^2)\right) \begin{bmatrix} I & & & & \\ I & I & & & \\ I & I & I & & \\ & I & I & I & \\ & & & & \ddots \end{bmatrix} \begin{bmatrix} I & & & \\ 2I & I & & \\ 3I & 2I & I & \\ \vdots & & & \ddots \end{bmatrix} \\ &\dot{\leq} \left(3CC'u + O(u^2)\right) \begin{bmatrix} I & & & & \\ 2I & I & & & \\ 3I & 2I & I & & \\ 4I & 3I & 2I & I & \\ \vdots & & & & \ddots \end{bmatrix}. \end{aligned}$$

This, together with the bound $|\delta\mathbf{A}| \dot{\leq} \left(|\delta\widetilde{\mathbf{L}}||\widetilde{\mathbf{L}}^{-1}| + O(u^2)\right)|\mathbf{A}|$, implies that

$$\delta A_{2i+1} \preceq \sum_{j=i}^{k} |j - i + 1||\alpha_{2j+1}|, \quad i = 0, \ldots, k,$$

completing the proof. ∎

Acknowledgments. We thank F. Arrigo and C. Fenu for providing their code for use as a reference in developing our numerical experiments and Yousef Saad for his inspiration and insightful comments throughout this project. We also thank the Minnesota Supercomputing Institute at the University of Minnesota for providing resources that contributed to the experimental results reported within this paper. Finally, we thank the anonymous referees, whose feedback led us to greatly improve our work in content as well as style.

REFERENCES

[1] F. Andersson, M. Carlsson, and K.-M. Perfekt, *Operator-Lipschitz estimates for the singular value functional calculus*, Proc. Amer. Math. Soc., 144 (2016), pp. 1867–1875, https://doi.org/10.1090/proc/12843.

[2] F. Andersson, M. Carlsson, J.-Y. Tourneret, and H. Wendt, *A new frequency estimation method for equally and unequally spaced data*, IEEE Trans. Signal Process., 62 (2014), pp. 5761–5774, https://doi.org/10.1109/TSP.2014.2358961.

[3] F. Arrigo and M. Benzi, *Edge modification criteria for enhancing the communicability of digraphs*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 443–468, https://doi.org/10.1137/15M1034131.

[4] F. Arrigo, M. Benzi, and C. Fenu, *Computation of generalized matrix functions*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 836–860, https://doi.org/10.1137/15M1049634.

[5] J. L. Aurentz, *GPU Accelerated Polynomial Spectral Transformation Methods*, Ph.D. thesis, Washington State University, 2014, https://doi.org/2376/5177.

[6] J. L. Aurentz, V. Kalantzis, and Y. Saad, *Cucheb: A GPU implementation of the filtered Lanczos procedure*, Comput. Phys. Commun., 220 (2017), pp. 332–340, https://doi.org/10.1016/j.cpc.2017.06.016.
[7] J. Baglama and L. Reichel, *Restarted block Lanczos bidiagonalization methods*, Numer. Algorithms, 43 (2006), pp. 251–272, https://doi.org/10.1007/s11075-006-9057-z.
[8] M. Benzi, D. A. Bini, D. Kressner, H. Munthe-Kaas, and C. Van Loan, *Exploiting Hidden Structure in Matrix Computations: Algorithms and Applications*, Lecture Notes in Math. 2173, Springer, New York, 2016.
[9] M. Benzi, E. Estrada, and C. Klymko, *Ranking hubs and authorities using matrix functions*, Linear Algebra Appl., 438 (2013), pp. 2447–2474, https://doi.org/10.1016/j.laa.2012.10.022.
[10] G. Berkolaiko and P. Kuchment, *Introduction to Quantum Graphs*, AMS, Providence, RI, 2013.
[11] C. W. Clenshaw, *A note on the summation of Chebyshev series*, Math. Comp., 9 (1955), pp. 118–120, https://doi.org/10.1090/S0025-5718-1955-0071856-0.
[12] J. J. Crofts, E. Estrada, D. J. Higham, and A. Taylor, *Mapping directed networks*, Electron. Trans. Numer. Anal., 37 (2010), pp. 337–350.
[13] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Soft., 38 (2011), pp. 1:1–1:25, https://doi.org/10.1145/2049662.2049663.
[14] N. Del Buono, L. Lopez, and R. Peluso, *Computation of the exponential of large sparse skew-symmetric matrices*, SIAM J. Sci. Comput., 27 (2005), pp. 278–293, https://doi.org/10.1137/030600758.
[15] N. Del Buono, L. Lopez, and T. Politi, *Computation of functions of Hamiltonian and skew-symmetric matrices*, Math. Comput. Simulation, 79 (2008), pp. 1284–1297, https://doi.org/10.1016/j.matcom.2008.03.011.
[16] T. A. Driscoll, N. Hale, and L. N. Trefethen, eds., *Chebfun Guide*, Pafnuty Publications, Oxford, 2014.
[17] V. L. Druskin and L. A. Knizhnerman, *Two polynomial methods of calculating functions of symmetric matrices*, U.S.S.R. Comput. Math. and Math. Phys., 29 (1989), pp. 112–121, https://doi.org/10.1016/S0041-5553(89)80020-5.
[18] H.-R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, SIAM J. Sci. Comput., 34 (2012), pp. A2220–A2246, https://doi.org/10.1137/110836535.
[19] F. R. Gantmacher, *The Theory of Matrices. Vol.* 1, AMS Chelsea Publishing, Providence, RI, 2000.
[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, MD, 2013.
[21] M. Hanke, J. Nagy, and R. Plemmons, *Preconditioned iterative regularization for ill-posed problems*, in Numerical Linear Algebra: Proceedings of the Conference in Numerical Linear Algebra and Scientific Computation (Kent, OH, 1992), L. Reichel, A. Ruttan, and R. S. Varga, eds., de Gruyter, Berlin, 1993, pp. 141–163.
[22] J. B. Hawkins and A. Ben-Israel, *On generalized matrix functions*, Linear Multilinear Algebra, 1 (1973), pp. 163–171, https://doi.org/10.1080/03081087308817015.
[23] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002, https://doi.org/10.1137/1.9780898718027.
[24] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008, https://doi.org/10.1137/1.9780898717778.
[25] L. Katz, *A new status index derived from sociometric analysis*, Psychometrika, 18 (1953), pp. 39–43, https://doi.org/10.1007/BF02289026.
[26] J. C. Mason and D. C. Handscomb, *Chebyshev Polynomials*, CRC Press, Boca Raton, FL, 2003.
[27] D. Mugnolo, *Semigroup Methods for Evolution Equations on Networks*, Springer, Cham, 2014.
[28] V. Noferini, *A formula for the Fréchet derivative of a generalized matrix function*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 434–457, https://doi.org/10.1137/16m1072851.
[29] H. D. Simon and H. Zha, *Low-rank matrix approximation using the Lanczos bidiagonalization process with applications*, SIAM J. Sci. Comput., 21 (2000), pp. 2257–2274, https://doi.org/10.1137/s1064827597327309.
[30] A. Smoktunowicz, *Backward stability of Clenshaw's algorithm*, BIT, 42 (2002), pp. 600–610, https://doi.org/10.1023/A:1022001931526.
[31] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2013.
[32] Y. Zhou and R.-C. Li, *Bounding the spectrum of large Hermitian matrices*, Linear Algebra Appl., 435 (2011), pp. 480–493, https://doi.org/10.1016/j.laa.2010.06.034.