Bounding matrix functionals via partial global block Lanczos decomposition

M. Bellalij

Laboratoire de Mathématiques et Leurs Applications, Université de Valenciennes, Le Mont Houy, 59313 Valenciennes Cedex, France.

L. Reichel

Department of Mathematical Sciences, Kent State University, Kent, OH 44242, USA.

G. Rodriguez

Dipartimento di Matematica e Informatica, Università di Cagliari, Viale Merello 92, 09123 Cagliari, Italy.

H. Sadok

Laboratoire de Mathématiques Pures et Appliquées, Université du Littoral, Centre Universtaire de la Mi-Voix, Batiment H. Poincarré, 50 Rue F. Buisson, BP 699, 62228 Calais Cedex, France.

Abstract

Approximations of expressions of the form $\mathcal{I}f := \mathrm{trace}(W^T f(A)W)$, where $A \in \mathbb{R}^{m\times m}$ is a large symmetric matrix, $W \in \mathbb{R}^{m\times k}$ with $k \ll m$, and f is a function, can be computed without evaluating $f(A)$ by applying a few steps of the global block Lanczos method to A with initial block-vector W. This yields a partial global Lanczos decomposition of A. We show that for suitable functions f upper and lower bounds for $\mathcal{I}f$ can be determined by exploiting the connection between the global block Lanczos method and Gauss-type quadrature rules. Our approach generalizes techniques advocated by Golub and Meurant for the standard Lanczos method (with block size one) to the global block Lanczos method. We describe applications to the computation of upper and lower bounds of the trace of $f(A)$ and consider, in partic-

Email addresses: Mohammed.Bellalij@univ-valenciennes.fr (M. Bellalij), reichel@math.kent.edu (L. Reichel), rodriguez@unica.it (G. Rodriguez), sadok@lmpa.univ-littoral.fr (H. Sadok)

ular, the computation of upper and lower bounds for the Estrada index, which arises in network analysis. We also discuss an application to machine learning.

Keywords: Gauss quadrature, global block Lanczos algorithm, trace computation, network analysis, machine learning.

---

1. Introduction

Let $A \in \mathbb{R}^{m\times m}$ be a large symmetric matrix, f a function, and $\boldsymbol{w} \in \mathbb{R}^m$ a vector of unit Euclidean norm. Application of ℓ steps of the (standard) Lanczos method to A with initial vector $\boldsymbol{w}$ yields the partial Lanczos decomposition

$$AV_\ell = V_\ell T_\ell + \boldsymbol{g}_\ell \boldsymbol{e}_\ell^T, \tag{1.1}$$

where the matrix $V_\ell \in \mathbb{R}^{m\times \ell}$ has orthonormal columns with initial column $\boldsymbol{w}$, $T_\ell \in \mathbb{R}^{\ell\times\ell}$ is a symmetric tridiagonal matrix, and $\boldsymbol{g}_\ell \in \mathbb{R}^m$ satisfies $V_\ell^T \boldsymbol{g}_\ell = \boldsymbol{0}$. Throughout this paper $\boldsymbol{e}_i$ denotes the ith column of the identity matrix of suitable size, and the superscript ^{T} stands for transposition. We assume that ℓ is small enough so that the Lanczos decomposition (1.1) with the stated properties exists.

It is well known that if the function f is analytic in a convex set that contains all the eigenvalues of A, then $V_\ell f(T_\ell)\boldsymbol{e}_1$ is an accurate approximation of $f(A)\boldsymbol{w}$ when ℓ is sufficiently large; see, e.g., [5] for error bounds. This observation makes it natural to use the approximation

$$\boldsymbol{w}^T f(A)\boldsymbol{w} \approx \boldsymbol{w}^T V_\ell f(T_\ell)\boldsymbol{e}_1 = \boldsymbol{e}_1^T f(T_\ell)\boldsymbol{e}_1. \tag{1.2}$$

Let

$$A = U\Lambda U^T, \quad \Lambda = \mathrm{diag}[\lambda_1, \ldots, \lambda_m] \in \mathbb{R}^{m\times m}, \tag{1.3}$$

be the spectral factorization of A with $U \in \mathbb{R}^{m\times m}$ orthogonal. We assume that the eigenvalues are ordered according to

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m.$$

In applications of interest the function f is analytic in an open simply connected set in the complex plane that contains all the eigenvalues of A. For now, we will assume f to satisfy this condition. More precise requirements on the regularity of f are stated in connection with the remainder formulas (3.6) and (3.7) in Section 3.

Substituting (1.3) into the left-hand side of (1.2) yields

$$\boldsymbol{w}^T f(A)\boldsymbol{w} = \boldsymbol{w}^T U f(\Lambda) U^T \boldsymbol{w} = \sum_{j=1}^{m} f(\lambda_j)\mu_j^2, \tag{1.4}$$

where $\mu_j = \boldsymbol{e}_j^T U^T \boldsymbol{w}$. The right-hand side of (1.4) can be written as a Stieltjes integral with a piecewise constant nondecreasing distribution function $\mu(\lambda)$ with jumps at the λ_j. Thus,

$$\boldsymbol{w}^T f(A)\boldsymbol{w} = \int f(\lambda) d\mu(\lambda). \tag{1.5}$$

Golub and Meurant [22, 23] observed that the expression on the right-hand side of (1.2) can be interpreted as an ℓ-point Gauss quadrature rule for the Stieltjes integral (1.5). It follows that

$$\boldsymbol{w}^T f(A)\boldsymbol{w} = \boldsymbol{e}_1^T f(T_\ell)\boldsymbol{e}_1 \qquad \forall f \in \mathbb{P}_{2\ell-1}, \tag{1.6}$$

where $\mathbb{P}_{2\ell-1}$ denotes the set of all polynomials of degree at most $2\ell - 1$. While the right-hand side of (1.6) may not be the most common way to express a Gauss quadrature rule, it can easily be written as a sum of ℓ terms by substitution of the spectral factorization of T_ℓ. This substitution shows that the Gaussian nodes are the eigenvalues of T_ℓ and the weights are the square of the first components of normalized eigenvectors.

Assume for the moment that the 2ℓth derivative of f is nonnegative (nonpositive) in the open interval (λ_1, λ_m). Then the remainder formula for Gauss quadrature shows that the right-hand side of (1.2) provides a lower (an upper) bound for the left-hand side; see Section 2 or [22, 23] for details.

Gauss–Radau quadrature rules are Gauss-type quadrature rules with one prescribed node. The $(\ell + 1)$-point Gauss–Radau rule with one prescribed node $\zeta \in \mathbb{R}$ outside the open interval (λ_1, λ_m) is exact for all polynomials in $\mathbb{P}_{2\ell}$. When the $(2\ell + 1)$st derivative of f does not change sign in the convex hull of $\{\lambda_1, \lambda_m, \zeta\}$, the remainder formula for Gauss–Radau quadrature shows that the Gauss–Radau rule yields an upper or a lower bound for (1.5). Whether the bound is upper or lower depends on the sign of the $(2\ell + 1)$st derivative and on the location of the fixed node ζ; see Section 2 or [22, 23] for further discussions. The $(\ell + 1)$-point Gauss–Radau rule can be evaluated by a formula analogous to the right-hand side of (1.2).

It is the purpose of the present paper to extend available results on the connection between Gauss-type quadrature and the Lanczos decomposition (1.1) to decompositions determined by the global block Lanczos method.

This is a block Lanczos method with a particular inner product. It was proposed and investigated by Elbouyahyaoui et al. [13] and Jbilou et al. [25]. We assume the block size k to be much smaller than m. The global block Lanczos method uses the inner product between block-vectors

$$\langle W_1, W_2 \rangle := \mathrm{trace}(W_1^T W_2), \qquad W_1, W_2 \in \mathbb{R}^{m\times k}, \tag{1.7}$$

and the induced Frobenius norm

$$\|W_1\|_F := \langle W_1, W_1 \rangle^{1/2}.$$

The k columns within each block-column generated by the global block Lanczos method are not required to be orthonormal. This reduces the computational effort needed by the global block Lanczos method when compared to the standard block Lanczos method for the same block size, because the latter method imposes that all vectors in every block be orthogonal to every other vector in every block. Another advantage of the global block Lanczos method is that the situation when a newly generated column is in the span of already available columns can be handled in a simple manner. This is discussed in Section 2. For brevity, we will refer to the global block Lanczos method simply as the global Lanczos method, and to the standard Lanczos method (with block size one) as the scalar Lanczos method.

This paper is organized as follows. We review the global Lanczos method in Section 2 and then discuss its connection with Gauss-type quadrature rules in Section 3. Numerical examples with applications to network analysis and machine learning are presented in Section 4. Concluding remarks can be found in Section 5.

We conclude this section with a brief outline of one of the applications to be considered in Section 4. Let $W \in \mathbb{R}^{m\times k}$, with $k \ll m$, be the initial block-vector for the global Lanczos method applied to the symmetric matrix $A \in \mathbb{R}^{m\times m}$. We will show in Section 3 that for suitable integrands f, application of ℓ steps of the global Lanczos method gives pairs of Gauss and Gauss–Radau quadrature rules that furnish upper and lower bounds for expressions of the form

$$\mathcal{I}f := \mathrm{trace}(W^T f(A) W).$$

Let

$$E_j = [\boldsymbol{e}_{k(j-1)+1}, \ldots, \boldsymbol{e}_{\min\{jk,m\}}], \qquad j = 1, 2, \ldots, \left\lfloor \frac{m+k-1}{k} \right\rfloor, \tag{1.8}$$

where $\lfloor \alpha \rfloor$ denotes the largest integer smaller than or equal to $\alpha \geq 0$. Application of the global Lanczos method to the matrix A with the initial block-vectors (1.8) determines pairs of Gauss and Gauss–Radau quadrature rules that, for appropriate integrands, yield upper and lower bounds for $\mathrm{trace}(f(A))$. The bounds improve with the number of global Lanczos steps applied to A for each initial block-vector E_j.

Estimates for the trace of a function of a large matrix are required in a variety of applications, including network analysis, machine learning, and quantum chromodynamics; see Bai et al. [3, 4] and Brezinski et al. [9] for discussions of and references to many applications. For instance, in network analysis, one is interested in determining the trace of the exponential of a symmetric adjacency matrix that describes an undirected simple graph. This trace is commonly referred to as the Estrada index; see, e.g., [16, 12]. We will apply the global Lanczos method to the computation of upper and lower bounds for the Estrada index, as well as to cluster analysis of networks.

A variety of methods for computing an approximation of the trace of a function of a large symmetric matrix have been proposed; see Bai et al. [3, 4], Brezinski et al. [8, 9], Fika et al. [19], and Tang and Saad [36]. Some methods focus on particular functions f and none of them provide upper and lower bounds. Several randomized algorithms for estimating the trace of a large matrix have been developed, such as the stochastic trace estimator by Hutchinson [24]. This method works well for symmetric diagonally dominant matrices. Avron and Toledo [1] provide a recent survey of randomized algorithms. They typically require a large number of matrix-vector product evaluations with A to yield estimates of moderate to high accuracy when A is a general symmetric matrix. These algorithms are attractive for execution on parallel computers with many processors when only fairly crude estimates of the trace are required. An approach that combines the application of modified moments with Hutchinson's trace estimator is described by Meurant [29].

We finally remark that the simple method of determining upper and lower bounds for the trace of $f(A)$ by application of the scalar Lanczos method to $A \in \mathbb{R}^{m\times m}$ with each one of the initial vectors $\boldsymbol{e}_j$, $j = 1, 2, \ldots, m$, requires considerable computational effort and therefore can be quite slow when A is large. This approach typically requires more computing time than the use of the global Lanczos method, because block methods can be executed efficiently on modern computers with a hierarchical memory structure. Indeed, the evaluation of the product of A with a block-vector $W \in \mathbb{R}^{m\times k}$ for moderate $k \ll m$ often is only slightly slower than the evaluation of the product of A with a single vector. Moreover, block methods

perform well in multi-processor computing environments; see, e.g., Gallivan et al. [20] for discussions and illustrations.

2. The global Lanczos method

This section outlines the global Lanczos method. Further details and properties can be found in [13, 25]. Application of ℓ steps of this method to a symmetric matrix $A \in \mathbb{R}^{m\times m}$ with initial block-vector $W \in \mathbb{R}^{m\times k}$, $k \ll m$, yields the partial global Lanczos decomposition

$$A[V_1, \ldots, V_\ell] = [V_1, \ldots, V_\ell]\widehat{T}_\ell + \beta_{\ell+1} V_{\ell+1} E_\ell^T, \tag{2.1}$$

where the block-columns $V_j \in \mathbb{R}^{m\times k}$ are orthonormal with respect to the inner product (1.7), i.e.,

$$\langle V_i, V_j \rangle = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

The matrix $\widehat{T}_\ell \in \mathbb{R}^{\ell k\times \ell k}$ can be expressed as

$$\widehat{T}_\ell = T_\ell \otimes I_k, \tag{2.2}$$

where $I_k \in \mathbb{R}^{k\times k}$ is the identity matrix and $T_\ell \in \mathbb{R}^{\ell\times\ell}$ is a symmetric and tridiagonal matrix. Moreover, $E_\ell = [\boldsymbol{e}_{k(\ell-1)+1}, \ldots, \boldsymbol{e}_{k\ell}] \in \mathbb{R}^{m\times k}$ is the ℓth block axis vector, $\beta_{\ell+1} > 0$, and $\otimes$ denotes the Kronecker product. We assume that ℓ is small enough so that the decomposition (2.1) with the stated properties exists. This is the generic situation. The following algorithm outlines the computations required by the global Lanczos method. Some algorithmic details are commented on below.

Algorithm 2.1 generates the decomposition (2.1) with initial block-column $V_1 = W/\|W\|_F$. The algorithm ignores possible breakdown caused by the vanishing of a coefficient β_{j+1}. We will comment on this rare situation below. The matrix $\widehat{T}_\ell$ in (2.1) is defined by the decomposition (2.2), where the symmetric tridiagonal matrix T_ℓ is determined by the coefficients α_j and β_j computed by the algorithm. We have

$$T_\ell = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \mathrm{O} \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & \cdot & \beta_\ell \\ \mathrm{O} & & & & \beta_\ell & \alpha_\ell \end{bmatrix}. \tag{2.3}$$

Algorithm 2.1 The global Lanczos method

Input: symmetric matrix $A \in \mathbb{R}^{m\times m}$, initial block-vector $W \in \mathbb{R}^{m\times k}$,
number of steps ℓ
$V_0 = 0$, $\beta_1 = \|W\|_F$, $V_1 = W/\beta_1$
for $j = 1$ **to** ℓ
  $\widetilde{V} = AV_j - \beta_j V_{j-1}$, $\alpha_j = \langle V_j, \widetilde{V}\rangle$
  $\widetilde{V} = \widetilde{V} - \alpha_j V_j$
  $\beta_{j+1} = \|\widetilde{V}\|_F$, $V_{j+1} = \widetilde{V}/\beta_{j+1}$
end for
Output: partial global Lanczos decomposition (2.1)

We will refer to this matrix as $\mathbf{tridiag}([\alpha_1, \ldots, \alpha_\ell], [\beta_2, \ldots, \beta_\ell])$.

The recursion formulas of Algorithm 2.1 show that

$$V_j = p_{j-1}(A)V_1, \qquad j = 1, 2, \ldots, \ell, \tag{2.4}$$

for some polynomials p_{j-1} of degree $j-1$.

We turn to the case when Algorithm 2.1 breaks down.

Proposition 2.1. *Assume that Algorithm 2.1 breaks down at step ℓ, i.e., $\beta_j > 0$ for $2 \le j \le \ell$, and $\beta_{\ell+1}$ vanishes. (In this situation the block-vector $V_{\ell+1}$ cannot be computed.) Then*

$$f(A)[V_1, \ldots, V_\ell] = [V_1, \ldots, V_\ell] f(\widehat{T}_\ell) \tag{2.5}$$

for any function f that is continuous on the convex hull of the spectrum of A.

Proof. Under the conditions of the proposition, decomposition (2.1) simplifies to

$$A[V_1, \ldots, V_\ell] = [V_1, \ldots, V_\ell]\widehat{T}_\ell.$$

It follows that

$$p(A)[V_1, \ldots, V_\ell] = [V_1, \ldots, V_\ell] p(\widehat{T}_\ell)$$

for any polynomial p. Let $\|\cdot\|_2$ denote the spectral norm and let $\Lambda(M)$

denote the spectrum of the square matrix M. Then

$$\begin{aligned}
& \|f(A)[V_1,\ldots,V_\ell] - [V_1,\ldots,V_\ell]f(\widehat{T}_\ell)\|_2 \\
= \;& \|(f(A)-p(A))[V_1,\ldots,V_\ell] - [V_1,\ldots,V_\ell](f(\widehat{T}_\ell)-p(\widehat{T}_\ell))\|_2 \\
\le \;& \|f(A)-p(A)\|_2\|[V_1,\ldots,V_\ell]\|_2 + \|[V_1,\ldots,V_\ell]\|_2\|f(\widehat{T}_\ell)-p(\widehat{T}_\ell)\|_2 \\
\le \;& \|f(A)-p(A)\|_2\|[V_1,\ldots,V_\ell]\|_F + \|[V_1,\ldots,V_\ell]\|_F\|f(\widehat{T}_\ell)-p(\widehat{T}_\ell)\|_2 \\
= \;& \sqrt{\ell}\|f(A)-p(A)\|_2 + \sqrt{\ell}\|f(\widehat{T}_\ell)-p(\widehat{T}_\ell)\|_2 \\
= \;& \sqrt{\ell}\max_{\lambda\in\Lambda(A)}|f(\lambda)-p(\lambda)| + \sqrt{\ell}\max_{\lambda\in\Lambda(T_\ell)}|f(\lambda)-p(\lambda)|.
\end{aligned}$$

The right-hand side can be made arbitrarily small by letting p be an arbitrarily accurate approximation of f on the union of the convex hulls of the spectra of A and T_ℓ. The existence of polynomials that approximate f arbitrarily accurately on this set follows from the Weierstraß approximation theorem. This establishes (2.5). □

3. Gauss-type quadrature

The relation between the (standard) block Lanczos method, in which the columns of each block are orthonormalized, and Gauss quadrature is discussed by Golub and Meurant [22, 23]; see also [18] for a recent treatment. When the block size is larger than one, the remainder term for block Gauss quadrature rules does not provide information about the sign of the error even when derivatives of the integrand do not change sign in the open interval (λ_1, λ_m). This section shows that for the global Lanczos method the situation is analogous to the one for the scalar Lanczos method, i.e., Gauss-type quadrature rules yield upper and lower bounds for the integral when suitable derivatives of the integrand do not change sign in the interval (λ_1, λ_m).

Consider the expression

$$\begin{aligned}
\mathcal{I}f \;&:=\; \operatorname{trace}(W^T f(A) W) \\
&=\; \|W\|_F^2 \operatorname{trace}(V_1^T f(A) V_1) \\
&=\; \|W\|_F^2 \operatorname{trace}(\widetilde{V}_1^T f(\Lambda)\widetilde{V}_1),
\end{aligned} \tag{3.1}$$

where $W \in \mathbb{R}^{m\times k}$, $V_1 = W/\|W\|_F$, and $\widetilde{V}_1 = U^T V_1$; cf. (1.3). The block size is assumed to be $k \ll m$. Similarly as (1.5), we can write

$$\boldsymbol{e}_i^T \widetilde{V}_1^T f(\Lambda)\widetilde{V}_1 \boldsymbol{e}_i = \int f(\lambda)\, d\mu_i(\lambda), \qquad i = 1, 2, \ldots, k,$$

where $\mu_i(\lambda)$ is a piecewise constant nondecreasing distribution function with jumps at the eigenvalues λ_j of A. Therefore,

$$\mathcal{I}f = \|W\|_F^2 \sum_{i=1}^{k} \int f(\lambda) d\mu_i(\lambda) = \|W\|_F^2 \int f(\lambda) d\mu(\lambda), \tag{3.2}$$

where

$$\mu(\lambda) := \sum_{i=1}^{k} \mu_i(\lambda) \tag{3.3}$$

is a piecewise constant nondecreasing distribution function with jumps at the eigenvalues λ_j. We will show that the entries of the tridiagonal matrix (2.3) are recursion coefficients for orthonormal polynomials associated with this distribution function. This yields the connection between the global Lanczos method and Gauss quadrature. We recall that an ℓ-point Gauss quadrature rule is characterized by the property that it is exact for all polynomials of degree $2\ell - 1$ or less.

Theorem 3.1. *Let the symmetric tridiagonal matrix (2.3) be determined by Algorithm 1. Then*

$$\mathcal{G}_\ell f = \|W\|_F^2 \boldsymbol{e}_1^T f(T_\ell) \boldsymbol{e}_1 \tag{3.4}$$

is an ℓ-point Gauss quadrature rule associated with the distribution function (3.3). Substituting the spectral factorization of T_ℓ into (3.4) yields the quadrature rule in terms of its nodes and weights. In particular, the eigenvalues of T_ℓ are the nodes. The weights are the square of the first components of suitably normalized eigenvectors.

Proof. Let the polynomials p_j be defined by (2.4). Then

$$\langle p_{j-1}(A)V_1, p_{i-1}(A)V_1 \rangle = \langle V_j, V_i \rangle = \begin{cases} 1, & j = i, \\ 0, & j \neq i. \end{cases}$$

Using (3.1) and (3.2), we obtain that

$$\langle p_{j-1}(A)V_1, p_{i-1}(A)V_1 \rangle = \int p_{j-1}(\lambda) p_{i-1}(\lambda) d\mu(\lambda).$$

Thus, the polynomials (2.4) implicitly determined by Algorithm 1 are orthonormal polynomials with respect to the inner product associated with the distribution function (3.3), at least if their degree is small enough. Substituting (2.4) into Algorithm 2.1 shows that the polynomials p_j satisfy a

three-term recurrence relation determined by the recursion formula for the block-vectors generated by the global Lanczos method. The recursion coefficients are the entries of the matrix (2.3). It follows that (3.4) is the ℓ-point Gauss quadrature rule associated with the distribution function (3.3). $\square$

If ℓ is replaced by $\ell+1$ in Algorithm 1, then the algorithm determines the symmetric tridiagonal matrix $T_{\ell+1} \in \mathbb{R}^{(\ell+1)\times(\ell+1)}$, which is associated with an $(\ell+1)$-point Gauss rule. An associated $(\ell+1)$-point Gauss–Radau rule is obtained by prescribing one of the nodes, denoted by ζ, and determining the $\ell+1$ weights and the remaining ℓ nodes so that the quadrature rule is exact for all polynomials of as high degree as possible. Let the prescribed node ζ be allocated in the intervals $\lambda \leq \lambda_1$ or $\lambda \geq \lambda_m$. Then the $(\ell+1)$-point Gauss–Radau rule exists and is exact for all polynomials $p \in \mathbb{P}_{2\ell}$. Similarly as the matrix $T_{\ell+1}$ is associated with an $(\ell+1)$-point Gauss rule, there is a symmetric tridiagonal matrix $T_{\ell+1,\zeta} \in \mathbb{R}^{(\ell+1)\times(\ell+1)}$ associated with the $(\ell+1)$-point Gauss–Radau rule. The matrix $T_{\ell+1,\zeta}$ can be determined by modifying the last diagonal entry of $T_{\ell+1}$ so that $T_{\ell+1,\zeta}$ has an eigenvalue at ζ. This is done in line 18 of Algorithm 3.1 below; see [22, 23] for details. The Gauss–Radau rule can be expressed as

$$\mathcal{R}_{\ell+1,\zeta} f = \|W\|_F^2 \boldsymbol{e}_1^T f(T_{\ell+1,\zeta}) \boldsymbol{e}_1. \tag{3.5}$$

Note that ℓ steps with Algorithm 1 determines the last subdiagonal entry $\beta_{\ell+1}$ of the Gauss–Radau matrix $T_{\ell+1,\zeta}$. Therefore, this matrix can be computed after ℓ steps have been carried out with the algorithm.

Let $\theta_1, \ldots, \theta_\ell$ denote the nodes of the Gauss rule (3.4). The remainder term for this rule is given by

$$\mathcal{I} f - \mathcal{G}_\ell f = \frac{f^{(2\ell)}(\breve{\theta})}{(2\ell)!} \int \prod_{j=1}^{\ell} (\lambda - \theta_j)^2 d\mu(\lambda), \tag{3.6}$$

where $\breve{\theta}$ lives in the open interval (λ_1, λ_m) and $f^{(2\ell)}$ denotes the derivative of f of order 2ℓ; see, e.g., Gautschi [21] for a proof. Similarly, let $\theta_{1,\zeta}, \ldots, \theta_{\ell,\zeta}$ be the free nodes of the Gauss–Radau rule (3.5). Then the remainder term is of the form

$$\mathcal{I} f - \mathcal{R}_{\ell+1,\zeta} f = \frac{f^{(2\ell+1)}(\breve{\theta}_\zeta)}{(2\ell+1)!} \int (\lambda - \zeta) \prod_{j=1}^{\ell} (\lambda - \theta_{j,\zeta})^2 d\mu(\lambda), \tag{3.7}$$

where $\breve{\theta}_\zeta$ lives in the largest open interval contained in the convex hull of $\{\lambda_1, \lambda_m, \zeta\}$; see [21] for a proof. These remainder formulas show that when

$f^{(2\ell)}$ is positive in the open interval (λ_1, λ_m), $\lambda_m \leq \zeta$, and $f^{(2\ell+1)}$ is positive in (λ_1, ζ), we have that

$$\mathcal{G}_\ell f < \mathcal{I} f < \mathcal{R}_{\ell+1,\zeta} f. \tag{3.8}$$

In fact, under these conditions,

$$\mathcal{G}_{\ell-1} f < \mathcal{G}_\ell f < \mathcal{I} f < \mathcal{R}_{\ell+1,\zeta} f < \mathcal{R}_{\ell,\zeta} f;$$

see, e.g., [28] for a proof. These inequalities hold, e.g., for $f(t) = \exp(t)$, a function of interest in network analysis, and show that the computed upper and lower bounds improve with each step of the global Lanczos method.

Algorithm 3.1 describes how approximations of $\mathrm{trace}(W^T f(A)W)$ are computed. For suitable integrands f, the algorithm determines upper and lower bounds for the trace. The algorithm is used to compute $\mathrm{trace}(f(A))$ by letting $W = E_j$, $j = 1, \ldots, \lfloor (m+k-1)/k \rfloor$, as in (1.8). We comment on some implementation details, starting with a discussion of the constants that have to be specified:

- ζ: an upper bound for the largest eigenvalue of A. We compute it by using the MATLAB function `eigs`, which implements the implicitly restarted Lanczos method [27]. One also may determine such a bound with the aid of Gershgorin disks or by using the fact that any matrix norm induced by a vector norm furnishes an upper bound for the largest eigenvalue of the matrix. We assume in the algorithm that all derivatives of the integrand f are positive in the open interval (λ_1, ζ). This is the case for the problems considered in the computed examples. The algorithm can easily be modified to yield upper and lower bounds of $\mathrm{trace}(W^T f(A)W)$ when the integrand has derivatives of constant sign (not necessarily positive).

- τ: stop tolerance

$$\frac{|\mathcal{R}_{\ell+1,\zeta} f - \mathcal{G}_\ell f|}{2|\mathcal{G}_\ell f|} < \tau. \tag{3.9}$$

  Thus, we terminate the computations when the relative difference between the upper and lower bounds is small enough.

- $N_{\max}$: maximum number of iterations.

- ε: constant used to detect breakdown in the iterations, i.e., when $\beta_{\ell+1}$ is "tiny." Breakdown is very unusual. Differently from the situation for standard block Lanczos methods, linear dependence of some columns

Algorithm 3.1 Approximations of $\mathrm{trace}(W^T f(A)W)$ by Gauss-type quadrature based on the global Lanczos algorithm.

1: **Input:** symmetric matrix $A \in \mathbb{R}^{m\times m}$, function f,
2: block-vector $W \in \mathbb{R}^{m\times k}$, constants: ζ, τ, $N_{\max}$, ε
3: $V_0 = 0$, $\beta_1 = \|W\|_F$, $V_1 = W/\beta_1$
4: $\ell = 0$, flag = **true**
5: **while** flag **and** $(\ell < N_{\max})$
6: $\ell = \ell + 1$
7: $\widetilde{V} = AV_\ell$, $\alpha_\ell = \mathrm{trace}(V_\ell^T \widetilde{V})$
8: $\widetilde{V} = \widetilde{V} - \beta_j V_{\ell-1} - \alpha_j V_\ell$
9: $\beta_{\ell+1} = \|\widetilde{V}\|_F$
10: **if** $\beta_{\ell+1} < \varepsilon$
11: $T = \mathbf{tridiag}([\alpha_1, \ldots, \alpha_\ell], [\beta_2, \ldots, \beta_\ell])$
12: $\mathcal{G}_\ell = [f(T)]_{11}$ (possibly with spectrum scaling; see below)
13: $\mathcal{R}_{\ell+1} = \mathcal{G}_\ell$
14: **break** *// exit the while loop*
15: **else**
16: $V_{\ell+1} = \widetilde{V}/\beta_{\ell+1}$
17: **end if**
18: $\hat{\alpha}_{\ell+1} = \zeta - \beta_{\ell+1} P_{\ell-1}(\zeta)/P_\ell(\zeta)$ *// $P_{\ell-1}$, P_ℓ orthogonal polynomials*
19: $T = \mathbf{tridiag}([\alpha_1, \ldots, \alpha_\ell], [\beta_2, \ldots, \beta_\ell])$
20: $\mathcal{G}_\ell = [f(T)]_{11}$ (possibly with spectrum scaling)
21: $T_\zeta = \mathbf{tridiag}([\alpha_1, \ldots, \alpha_\ell, \hat{\alpha}_{\ell+1}], [\beta_2, \ldots, \beta_{\ell+1}])$
22: $\mathcal{R}_{\ell+1} = [f(T_\zeta)]_{11}$ (possibly with spectrum scaling)
23: flag $= |\mathcal{R}_{\ell+1} - \mathcal{G}_\ell| > 2\tau|\mathcal{G}_\ell|$
24: **end while**
25: $\mathcal{F} = \frac{1}{2}\beta_1^2(\mathcal{G}_\ell + \mathcal{R}_{\ell+1})$
26: **Output:** Approximation $\mathcal{F}$ of $\mathrm{trace}(W^T f(A)W)$, lower and upper
27: bounds $\beta_1^2\mathcal{G}_\ell$ and $\beta_1^2\mathcal{R}_{\ell+1}$ for suitable functions f,
28: number of iterations ℓ

of the matrix $V_{\ell+1}$ does not necessarily result in breakdown. Lines 10-14 of the algorithm handle breakdown. When $\beta_{\ell+1} = 0$, the Gauss rule provides the exact value of the integral in the absence of round-off errors.

- The number of steps ℓ required to satisfy the stop tolerance (3.9) depends both on the function f and the matrix A, because the remainder terms (3.6) and (3.7) depend on the function f and the measure $d\mu$. For instance, when $f(\lambda) = 1/\lambda$ and A is a symmetric positive definite matrix with maximal eigenvalue one, the quality of the error bounds (3.8) generally deteriorates when the smallest eigenvalues of A approach the origin.

Line 18 of Algorithm 3.1 determines the last diagonal entry of the symmetric tridiagonal matrix associated with Gauss–Radau quadrature with a fixed node at ζ. Details are described in [22, 23].

Spectrum scaling may be required to avoid overflow in the computations with certain functions f, such as the exponential function. This function is used in our first numerical experiment of Section 4. Instead of evaluating $f(T)$ at lines 12 and 20 of Algorithm 3.1, we compute the spectral factorization $T = \widetilde{U}\widetilde{D}\widetilde{U}^T$ of the symmetric tridiagonal matrix T, and evaluate

$$f_\rho(T) = \widetilde{U}\exp(\widetilde{D} - \rho I)\widetilde{U}^T,$$

for a scaling parameter ρ. For instance, we may let ρ be the largest eigenvalue of T. The value $f(T)$ is given by $\exp(\rho)f_\rho(T)$. Its explicit evaluation may lead to overflow. The same approach is used for the evaluation of $f(T_\zeta)$ in line 22 of Algorithm 3.1.

We also consider the functions $f(A) = A^p$, $p = 1, 2, \dots$, in the computed examples of Section 4. In this case we compute at each iteration

$$f_\rho(T) = \widetilde{U}(\rho^{-1}\widetilde{D})^p\widetilde{U}^T.$$

The desired value is $f(T) = \rho^p f_\rho(T)$.

The obvious choice for the spectrum scaling parameter ρ is to set it to the largest eigenvalue of A, in particular, when this eigenvalue is determined anyway to define the fixed node of Gauss–Radau rules. However, there are alternative cheaper choices. Since the spectral factorization of T is needed to compute $f_\rho(T)$, the spectrum scaling parameter ρ can be chosen to be the largest eigenvalue of T with no additional cost. The latter choice is attractive when the Gauss–Radau quadrature rule is not required, e.g., because the Gauss quadrature formula $\mathcal{G}_\ell$ is exact. This is the case when $f(A) = A^p$ for a small enough power p. Subsection 4.1 discusses an application in which this kind of functions arise. Also the application to machine learning, discussed in Subsection 4.2, does not require the evaluation of Gauss–Radau rules.

When the Gauss and Gauss–Radau rules are not exact and the derivatives of the function f in the remainder terms for Gauss and Gauss–Radau quadrature, see (3.6) and (3.7), change sign in the interval of integration, these quadrature rules are not guaranteed to bracket the desired quantity. In this situation, it may be attractive to use pairs of Gauss and anti-Gauss quadrature rules to determine approximations of upper and lower bounds. Anti-Gauss quadrature rules have been introduced by Laurie [26]. Their application in linear algebra and network analysis is described in [10, 18].

4. Numerical examples

This section compares the application of Gauss and Gauss–Radau quadrature rules based on the global Lanczos method to the use of Gauss and Gauss–Radau quadrature rules based on the scalar Lanczos algorithm, with the aim of assessing the performance of Algorithm 3.1. When possible, we compare these methods to standard computational procedures based on functions available in MATLAB. All computations were performed using MATLAB 8.1 (R2013a) on an Intel Core i7/860 computer with 8Gb RAM running Linux. The MATLAB functions used in the experiments are available from the authors upon request.

4.1. Applications to network analysis

The first test example consists of computing upper and lower bounds for the trace of the exponential of a large adjacency matrix associated with a simple undirected graph. Gauss-type quadrature rules have previously been applied to bound individual elements of the matrix exponential, such as elements on the diagonal; see Benzi and Boito [6] and Fenu et al. [17], who considered undirected networks, and Baglama et al. [2], who extended results from [17] to directed networks. The block quadrature rules described in [18] were found to generally furnish upper and lower bounds of the entries of the matrix exponential, also for nonsymmetric adjacency matrices, but are not guaranteed to do so. Gauss-type quadrature rules also are applied by Bonchi et al. [7] to bound individual entries of matrix functions evaluated for an adjacency matrix.

Let $G = \{\mathcal{V}, \mathcal{E}\}$ be a graph defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. The adjacency matrix associated with G is the matrix $A = [a_{ij}] \in \mathbb{R}^{m\times m}$ defined by $a_{ij} = 1$ if there is an edge from node i to node j, and $a_{ij} = 0$ otherwise. The adjacency matrix is symmetric if and only if G is undirected. We assume that G is undirected, large (i.e., the number of nodes m is large), and sparse (having much fewer than $O(m^2)$ edges between the nodes). Such

graphs arise in numerous scientific and industrial applications; see, e.g., [15, 30].

The Estrada index of a graph is defined as

$$\text{trace}(\exp(A)) = \sum_{i=1}^{m} [\exp(A)]_{ii} = \sum_{i=1}^{m} \exp(\lambda_i),$$

where λ_i, $i = 1, \ldots, m$, denote the eigenvalues of the adjacency matrix A associated with the graph. This index provides a global characterization of the graph, and can be used to express various relevant quantities; see [14]. We considered six undirected networks obtained from real world applications, which were used in the numerical experiments in [18] and are described there in more detail: Email (1133 nodes, 10902 edges), Yeast (2114 nodes, 4480 edges), Power (4941 nodes, 13188 edges), Internet (22963 nodes, 96872 edges), Collaboration (40421 nodes, 351304 edges), and Facebook (63731 nodes, 1545686 edges). We compute the Estrada index for each network both by the scalar Lanczos method and by the global Lanczos method as implemented by Algorithm 3.1 with block size k and stop tolerance $\tau = 10^{-3}$, i.e., the computations are terminated when the relative distance between the computed upper and lower bounds for the trace are sufficiently close; cf. (3.9) and line 23 of Algorithm 3.1. The parameter $N_{\max}$ is chosen large enough not to affect the computations. The adjacency matrices are stored using MATLAB's sparse storage format. For the four smallest networks, which all have fewer than 5000 nodes, we also compute the Estrada index by using the MATLAB function `expm`, which evaluates the matrix exponential. For large networks this approach is too time consuming to be feasible.

Table 4.1 reports execution times for the three methods. We selected the block size $k_{\min}$ that gave the shortest execution time for the global Lanczos algorithm. For the scalar Lanczos method and the MATLAB function `expm`, we also report the speedup factor, i.e., the ratios between the execution times for the alternative methods and the execution time for the global method. The global Lanczos method is seen to be much faster than scalar Lanczos and the `expm` function; global Lanczos is at least three times faster than scalar Lanczos, and more than ten times faster than `expm`. It was not possible to apply the last method to the three largest networks. We remark that the execution time only grows slowly with k for $k \geq k_{\min}$. This is commented on further below and illustrated by Figure 4.2.

Table 4.2 complements Table 4.1 by showing the number of iterations and the number of matrix-vector product evaluations carried out during the computations for the latter table. Here we count a matrix-vector product

Table 4.1: Execution times for computing the Estrada index of our test networks. We report the minimal computing time for the global Lanczos algorithm and the corresponding block size $k_{\min}$. For the scalar Lanczos method and the `expm` function we report the execution times and the speedup factors (SF) with respect to the block method, i.e., the ratio between the computing times.

		Global Lanczos		Scalar Lanczos		`expm`	
Matrix	Nodes	time	$k_{\min}$	time	SF	time	SF
Email	1133	3.45e-01	80	3.54e+00	(10)	1.18e+01	(34)
Yeast	2114	4.73e-01	60	3.20e+00	(7)	1.01e+01	(21)
Power	4941	1.89e+00	40	1.28e+01	(7)	2.14e+01	(11)
Internet	22963	1.22e+02	8	3.30e+02	(3)	-	-
Collaborations	40421	4.80e+02	40	1.25e+03	(3)	-	-
Facebook	63731	2.64e+03	60	8.82e+03	(3)	-	-

Table 4.2: Number of iterations and matrix-vector product evaluations (MVPs)

			Global Lanczos		Scalar Lanczos
Matrix	Nodes	$k_{\min}$	iterations	MVPs	MVPs
Email	1133	80	130	9730	9745
Yeast	2114	60	213	12596	8652
Power	4941	40	618	24644	22209
Internet	22963	8	28776	230158	226433
Collaborations	40421	40	10115	404372	416648
Facebook	63731	60	12075	723912	758618

of A and a block-vector with k columns as k matrix-vector product evaluations. Thus, Table 4.2 reports the total number of iterations and matrix-vector product evaluations required to compute upper and lower bounds with specified accuracy for

$$\mathcal{I}f = \operatorname{trace}(E_j^T f(A) E_j), \qquad j = 1, 2, \ldots, \left\lfloor \frac{m+k-1}{k} \right\rfloor,$$

with the matrices E_j defined by (1.8) and $f(A) = \exp(A)$. It is interesting to note that for several of the problems, the number of matrix-vector product evaluations is essentially the same for the block sizes 1 and $k_{\min}$.

Both the global Lanczos and Lanczos methods satisfied the stopping criterion (3.9) without exceeding the maximum number of allowed iterations, $N_{\max}$, for all networks. For the smallest networks, for which the MATLAB

function `expm` can be evaluated, the global and scalar Lanczos methods gave approximations of the Estrada index with a relative error bounded by τ.

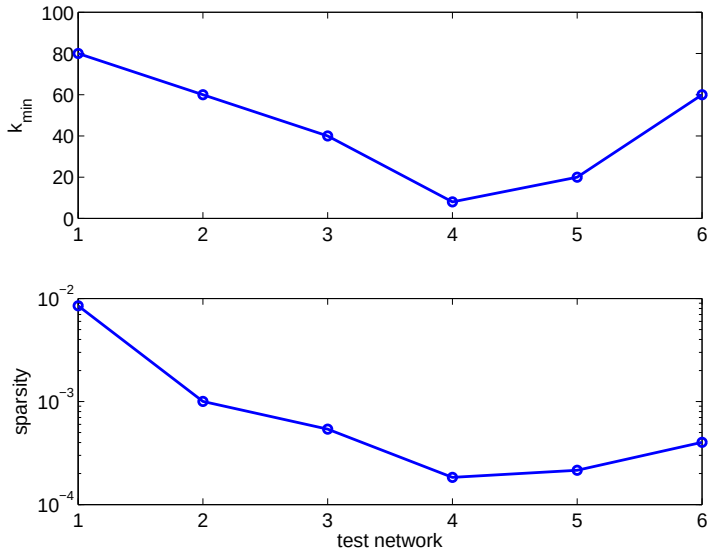

Figure 4.1: For each of the six networks, ordered as in Tables 4.1 and 4.2, the top graph report the block size $k_{\min}$ which produces the optimal execution time. The bottom graph represents the sparsity of each adjacency matrix, that it, the ratio between the number of nonzero elements and m^2.

It is interesting that the optimal block size $k_{\min}$ depends on the network and does not vary monotonically with the size of the problem. Instead, $k_{\min}$ is strongly related to the sparsity of the network. This is evident in Figure 4.1, whose top graph displays $k_{\min}$ for the six networks, while the bottom graph reports the ratio between the number of existing edges and the largest admissible number of edges, that is, m^2. We believe that the dependence on sparsity is caused by the optimized implementation of the sparse matrix operations in MATLAB, and not simply by the presence of many null entries in the matrix. This conjecture is supported by the fact that repeating the experiment for the first three networks with nonsparse matrices, that is, using the standard storage and computational routines in MATLAB, the value of $k_{\min}$ was found to oscillate around 250. Figure 4.2 graphs the execution time in seconds with respect to block size for the three largest networks and shows that a careful choice of the block size k is not essential, and that $k_{\min}$ can safely be overestimated without burdening the

algorithm. It is only important to avoid very small block sizes.

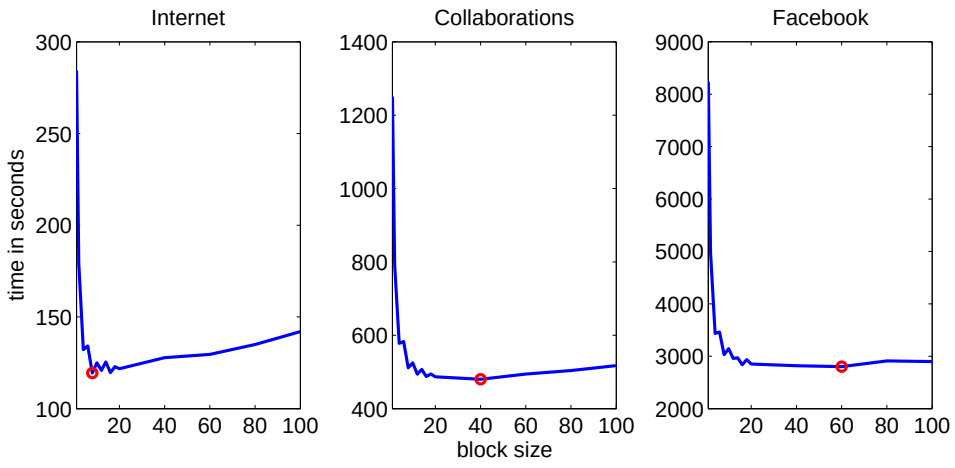

Figure 4.2: Execution times to compute the Estrada index for the three largest networks, versus the block size k.

To illustrate the progress in the approximation of the Estrada index during the computations, we report in Figure 4.3 the upper and lower bounds for the Email network (top-left) and the Yeast network (top-right) versus the number of iterations. The global Lanczos method is applied with block size $k = 50$ to compute bounds for $\mathrm{trace}(W^T f(A)W)$, with $W = E_1$; see (1.8). The parameter ζ in (3.7), required to obtain upper bounds by Gauss–Radau quadrature rules, is chosen to be the largest eigenvalue λ_m of A, as computed by the `eigs` function of MATLAB. Similarly as in the other tests, the stop tolerance is $\tau = 10^{-3}$. In the bottom graphs of Figure 4.3, we report the quantities

$$\frac{|\mathcal{R}_{\ell+1} - \mathcal{G}_\ell|}{2|\mathcal{G}_\ell|}, \tag{4.1}$$

cf. (3.9), obtained when $\zeta = \lambda_m$ (continuous curves) and when $\zeta = \|A\|_\infty$, where $\|\cdot\|_\infty$ denotes the matrix norm induced by the uniform vector norm (dashed curves). Results for both the Email and Yeast networks are displayed. The graphs show that using the cheaper upper bound $\zeta = \|A\|_\infty$ for the spectrum leads to just a small increase in the number of iterations required to reach convergence. We remark that, especially for the largest networks, the first upper bounds computed by the global Lanczos method are so large that it is essential to apply a spectrum scaling procedure.

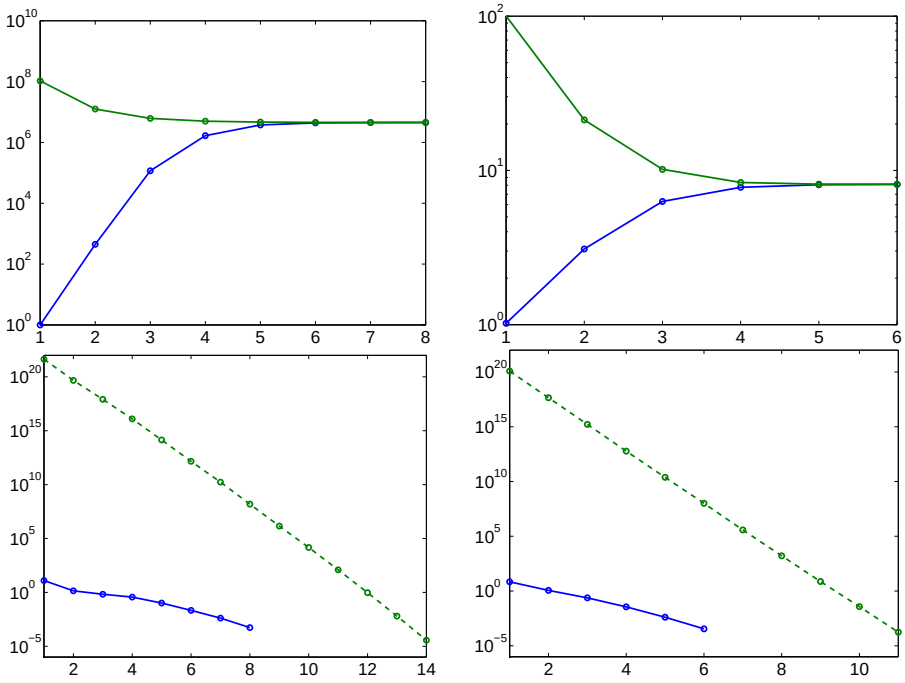

Figure 4.3: Left-hand side graphs are for the Email network and right-hand side graphs for the Yeast network. Top row: upper and lower bounds for the Estrada index versus number of iterations. Bottom row: the continuous curves show the quotient (4.1) at each iteration when $\zeta = \lambda_m$ in (3.7), and the dashed curves show the same quotient when $\zeta = \|A\|_\infty$.

One of the typical features of a complex network is the tendency to exhibit clusters of nodes, sometimes referred to as "communities". Various indices have been proposed to measure clustering, and most of them are connected to the number of "triangles" present in the network, that is, triplets of nodes that are reciprocally connected; see, e.g., [30]. As the element (i, j) of the pth power of the adjacency matrix contains the number of walks of length p connecting node i to node j, the number of triangles T may be computed as

$$T = \frac{1}{6}\,\mathrm{trace}(A^3).$$

Table 4.3 reports a comparison of the global and scalar Lanczos methods to the direct computation of `trace(`A^3`)` in MATLAB in the fashion of Table 4.1. We only consider the three largest test networks. As in the previous experiment, the block method is much faster than the scalar approach, but it turns out that the MATLAB sparse library is extremely efficient for

this kind of computations. We were not able to find documentation for the algorithm used by MATLAB for sparse matrix power. The global Lanczos method was faster for the largest network, Facebook, which contains 63731 nodes.

Table 4.3: Execution times for computing the number of triangles in our test networks. We report the minimal computing time for the global Lanczos algorithm and the corresponding block size $k_{\min}$. For the scalar Lanczos method and the MATLAB computation we report the execution times and the speedup factors (SF) with respect to the block method, i.e., the ratio between the computing times.

		Global Lanczos		Scalar Lanczos		`trace`(A^3)	
Matrix	Nodes	time	$k_{\min}$	time	SF	time	SF
Internet	22963	2.71e+01	16	1.09e+02	(4)	7.88e+00	(0.3)
Collaborations	40421	1.09e+02	60	6.11e+02	(6)	3.53e+00	(0.03)
Facebook	63731	4.93e+02	120	4.37e+03	(9)	9.66e+02	(2)

We observed that the very good performance of the MATLAB function `trace` degrades as the power of the adjacency matrix grows. To investigate this further, we considered the problem of counting "polygons", rather than just triangles. We let the exponent p vary and computed trace(A^p) for $p = 4, 6, \ldots, 18$ by the three above methods. The computation with the MATLAB function `trace` break down already for fairly small networks with an "out of memory" error because of the fill-in of the matrix powers. The computing time versus the exponent p for two of the smallest networks are plotted in Figure 4.4. The Lanczos methods become competitive when the exponent increases.

4.2. An application to machine learning

Training a Gaussian process consists of identifying a set of parameters from available data. Specifically, one is interested in maximizing the log *marginal likelihood* of the training data with respect to the variation of the parameters; see, e.g., Ngo et al. [31] and Rasmussen et al. [34]. The solution of this optimization problem involves the repeated computation of trace($T^{-1}S$) for certain Toeplitz matrices T and S. The matrix T is the covariance matrix of a Gaussian distribution, while S is its derivative with respect to one of the parameters to be identified.

In our experiments, we consider the Gaussian matrix

$$[G_\sigma]_{ij} = \sqrt{\frac{\sigma}{2\pi}} \exp\left(-\frac{\sigma}{2}(i-j)^2\right), \quad i, j = 1, \ldots, m,$$

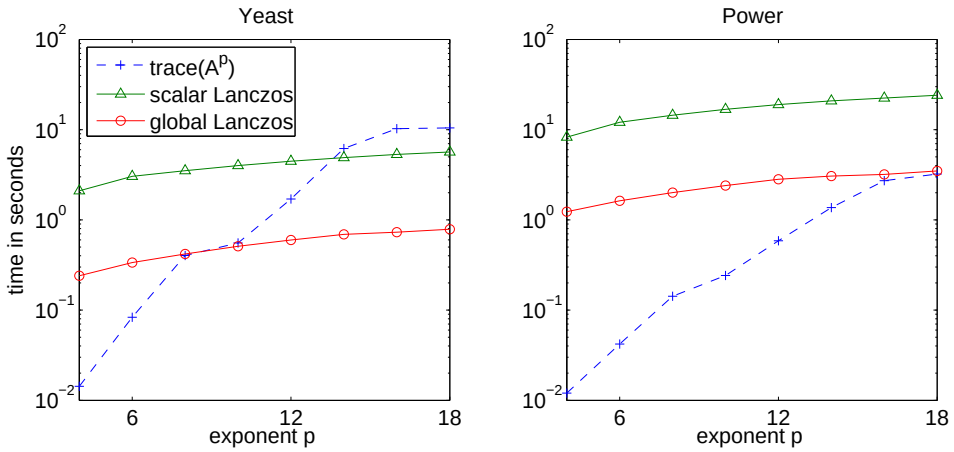

Figure 4.4: Execution times for the two test networks Yeast and Power required to compute the trace of the matrix power with exponent $p = 4, 6, \ldots, 18$. The execution time for the MATLAB function `trace`, the scalar Lanczos method, and the global Lanczos method with block size $k = 50$, versus the exponent p are reported.

and let $T = G_1$ and $S = G_{1/2}$. We approximate $\mathrm{trace}(T^{-1}S)$ using the global Lanczos method, and compare it to the scalar Lanczos method and to the use of the MATLAB command `trace(`$T\backslash S$`)`. This command first computes $T^{-1}S$ with the aid of LU factorization with partial pivoting of T. Thus, T^{-1} is not formed. Having computed $T^{-1}S$, its diagonal entries are summed. The matrices are stored without exploiting their structure.

We set $f(A) = A$ in Algorithm 3.1, so that no scaling strategy is needed, and substitute the matrix product at line 7 by the expression $\widetilde{V} = T^{-1}SV_\ell$. To compute the block $\widetilde{V}$, we solve the linear system of equations

$$T\widetilde{V} = SV_\ell,$$

by the conjugate gradient method, using the optimal circulant preconditioner [11] to accelerate the convergence. We use the MATLAB toolbox `smt` [35] to handle the Toeplitz and circulant matrices and to evaluate matrix-vector products efficiently. Other, more recently developed circulant preconditioners, also could have been applied; see [32, 33] for discussions and references.

To investigate which block size k makes the global Lanczos method most effective, we let $m = 5000$ and record the execution time for $k = 20, 40, \ldots, 200$; see the left-hand side graph of Figure 4.5. The shortest time

is obtained for $k = 160$, but we note that there are only small changes in execution time for $100 \leq k \leq 180$.

The right-hand side graph of Figure 4.5 compares the execution times for matrices of sizes $m = 1000, 2000, \ldots, 10000$. The global Lanczos method is applied with $k = 160$, and is seen to be faster than the scalar Lanczos algorithm. The MATLAB procedure described above is faster for the smaller matrices, but for $m > 2000$ the global Lanczos method is preferable. It is important to point out that Lanczos methods are more convenient to use for large matrices both in terms of speed and storage. Indeed, the evaluation of trace$(T^{-1}S)$ using structure ignoring MATLAB functions requires the allocation of storage for three matrices of size m. Thus, $O(m^2)$ memory locations are needed. On the contrary, only $O(m)$ storage locations are needed to store a Toeplitz matrix and execute Algorithm 2.

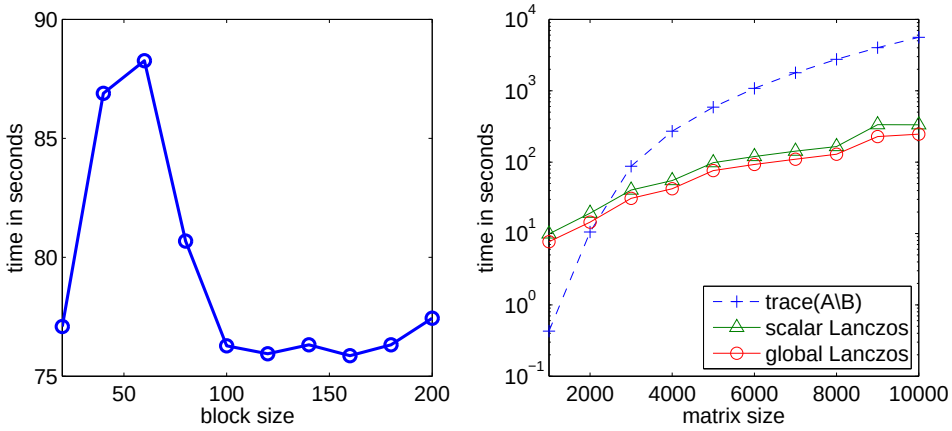

Figure 4.5: Left-hand side graph: execution time for global Lanczos versus the block size. Right-hand side graph: execution time versus the matrix size m for scalar Lanczos, global Lanczos with block size 160, and structure ignoring computations.

5. Conclusion

We describe how the global block Lanczos method determines Gauss and Gauss–Radau quadrature rules that can be applied to compute upper and lower bounds of the trace of the function of a matrix, provided that some derivatives of the function do not change sign in certain intervals. Computed examples illustrate the competitiveness of the approach presented.

Acknowledgment

LR would like to thank Mohammed Bellalij, Giuseppe Rodriguez, and Hassane Sadok for making visits to their universities in Valenciennes, Cagliari, and Calais, respectively, possible and enjoyable. This paper grew out of discussions during these visits. The authors would like to thank the referees for carefully reading the manuscript and for comments that lead to improvements of the presentation. The research of MB benefited from the support of the FMJH Program Gaspard Monge in Optimization and Operation Research, and from the support to this program from EDF. Research by LR is supported in part by NSF grant DMS-1115385.

References

[1] H. Avron, S. Toledo, Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix, J. ACM, 58 (2011), pp. 8:1–8:34.

[2] J. Baglama, C. Fenu, L. Reichel, G. Rodriguez, Analysis of directed networks via partial singular value decomposition and Gauss quadrature, Linear Algebra Appl., 456 (2014) pp. 93–121.

[3] Z. Bai, M. Fahey, G. Golub, Some large-scale matrix computation problems, J. Comput. Appl. Math., 74 (1996), pp. 71–89.

[4] Z. Bai, G. Golub, Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices, Ann. Numer. Math., 4 (1997), pp. 29–38.

[5] B. Beckermann, L. Reichel, Error estimation and evaluation of matrix functions via the Faber transform, SIAM J. Numer. Anal., 47 (2009), pp. 3849–3883.

[6] M. Benzi, P. Boito, Quadrature rule-based bounds for functions of adjacency matrices, Linear Algebra Appl., 433 (2010), pp. 637–652.

[7] F. Bonchi, P. Esfandiar, D.F. Gleich, C. Greif, L. V. S. Lakshmanan, Fast matrix computations for pairwise and columnwise commute times and Katz scores, Internet Math., 8 (2012), pp. 73–112.

[8] C. Brezinski, P. Fika, M. Mitrouli, Estimations of the trace of powers of self-adjoint operators by extrapolation of the moments, Electron. Trans. Numer. Anal., 39 (2012), pp. 144–159.

[9] C. Brezinski, P. Fika, M. Mitrouli, Moments of a linear operator on a Hilbert space, with applications to the trace of the inverse of matrices and the solution of equations, Numer. Linear Algebra Appl., 19 (2012), pp. 937–953.

[10] D. Calvetti, L. Reichel, F. Sgallari, Application of anti-Gauss quadrature rules in linear algebra, in Applications and Computation of Orthogonal Polynomials, W. Gautschi, G. H. Golub, G. Opfer, eds., Birkhäuser, Basel, 1999, pp. 41–56.

[11] T.F. Chan, An optimal circulant preconditioner for Toeplitz systems, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 766–771.

[12] J.A. de la Peña, I. Gutman, J. Rada, Estimating the Estrada index, Linear Algebra Appl., 427 (2007), pp. 70–76.

[13] L. Elbouyahyaoui, A. Messaoudi, H. Sadok, Algebraic properties of the block GMRES and block Arnoldi methods, Electron. Trans. Numer. Anal., 33 (2009), pp. 207–220.

[14] E. Estrada, N. Hatano, M. Benzi, The physics of communicability in complex networks, Phys. Rep., 514 (2012), pp. 89–119.

[15] E. Estrada, D.J. Higham, Network properties revealed through matrix functions, SIAM Rev., 52 (2010), pp. 696–714.

[16] E. Estrada, J.A. Rodríguez-Velázquez, Subgraph centrality in complex networks, Phys. Rev. E, 71 (2005), 056103 (11 pages).

[17] C. Fenu, D. Martin, L. Reichel, G. Rodriguez, Network analysis via partial spectral factorization and Gauss quadrature, SIAM J. Sci. Comput., 35 (2013), pp. A2046–A2068.

[18] C. Fenu, D. Martin, L. Reichel, G. Rodriguez, Block Gauss and anti-Gauss quadrature with application to networks, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1655–1684.

[19] P. Fika, M. Mitrouli, P. Roupa, Estimates for the bilinear form $x^T A^{-1} y$ with applications to linear algebra problems, Electron. Trans. Numer. Anal., 43 (2014), pp. 70–89.

[20] K. Gallivan, M. Heath, E. Ng, B. Peyton, R. Plemmons, J. Ortega, C. Romine, A. Sameh, R. Voigt, Parallel Algorithms for Matrix Computations, SIAM, Philadelphia, 1990.

[21] W. Gautschi, Orthogonal Polynomials: Computation and Approximation, Oxford University Press, Oxford, 2004.

[22] G.H. Golub, G. Meurant, Matrices, moments and quadrature, in Numerical Analysis 1993, D.F. Griffiths, G.A. Watson, eds., Longman, Essex, England, 1994, pp. 105–156.

[23] G.H. Golub, G. Meurant, Matrices, Moments and Quadrature with Applications, Princeton University Press, Princeton, 2010.

[24] M.F. Hutchinson, A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines, Commun. Stat. Simul. Comput., 18 (1989), pp. 1059–1076.

[25] K. Jbilou, A. Messaoudi, H. Sadok, Global FOM and GMRES algorithms for matrix equations, Appl. Numer. Math., 31 (1999), pp. 49–63.

[26] D.P. Laurie, Anti-Gaussian quadrature formulas, Math. Comp., 65 (1996), pp. 739–747.

[27] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods, SIAM, Philadelphia, 1998.

[28] G. López Lagomasino, L. Reichel, L. Wunderlich, Matrices, moments, and rational quadrature, Linear Algebra Appl., 429 (2008), pp. 2540–2554.

[29] G. Meurant, Estimates of the trace of the inverse of a symmetric matrix using the modified Chebyshev algorithm, Numer. Algorithms, 51 (2009), pp. 309–318.

[30] M.E.J. Newman, Networks: An Introduction, Oxford University Press, 2010.

[31] T.T. Ngo, M. Bellalij, Y. Saad, The trace ratio optimization problem, SIAM Rev., 54 (2012), pp. 545–569.

[32] S. Noschese, L. Reichel, Generalized circulant Strang-type preconditioners, Numer. Linear Algebra Appl., 19 (2012), pp. 3–17.

[33] S. Noschese, L. Reichel, A note on superoptimal generalized circulant preconditioners, Appl. Numer. Math., 75 (2014), pp. 188–195.

[34] C.E. Rasmussen, Gaussian processes in machine learning, in Advanced Lectures on Machine Learning, O. Bousquet, U. von Luxburg, G. Rätsch, eds., Springer, Berlin, 2004, pp. 63–71.

[35] M. Redivo–Zaglia, G. Rodriguez, `smt`: a Matlab toolbox for structured matrices, Numer. Algorithms, 59 (2012), pp. 639–659.

[36] J. Tang, Y. Saad, A probing method for computing the diagonal of the matrix inverse, Numer. Linear Algebra Appl., 19 (2012), pp. 485–501.