SIAM J. SCI. COMPUT.
Vol. 35, No. 4, pp. A2046–A2068


# NETWORK ANALYSIS VIA PARTIAL SPECTRAL FACTORIZATION AND GAUSS QUADRATURE*

C. FENU†, D. MARTIN‡, L. REICHEL‡, AND G. RODRIGUEZ†

**Abstract.** Large-scale networks arise in many applications. It is often of interest to be able to identify the most important nodes of a network or to ascertain the ease of traveling between nodes. These and related quantities can be determined by evaluating expressions of the form $\mathbf{u}^T f(A)\mathbf{w}$, where $A$ is the adjacency matrix that represents the graph of the network, $f$ is a nonlinear function, such as the exponential function, and $\mathbf{u}$ and $\mathbf{w}$ are vectors, for instance, axis vectors. This paper describes a novel technique for determining upper and lower bounds for expressions $\mathbf{u}^T f(A)\mathbf{w}$ when $A$ is symmetric and bounds for many vectors $\mathbf{u}$ and $\mathbf{w}$ are desired. The bounds are computed by first evaluating a low-rank approximation of $A$, which is used to determine rough bounds for the desired quantities for all nodes. These rough bounds indicate for which vectors $\mathbf{u}$ and $\mathbf{w}$ more accurate bounds should be computed with the aid of Gauss-type quadrature rules. This hybrid approach is cheaper than only using Gauss-type rules to determine accurate upper and lower bounds in the common situation when it is not known a priori for which vectors $\mathbf{u}$ and $\mathbf{w}$ accurate bounds for $\mathbf{u}^T f(A)\mathbf{w}$ should be computed. Several computed examples, including an application to software engineering, illustrate the performance of the hybrid method.

**Key words.** complex networks, low-rank approximation, Lanczos process, Gauss quadrature, software networks

**AMS subject classifications.** 65F15, 65F60, 05C50, 05C82

**DOI.** 10.1137/130911123

## 1. Introduction.

Let $G$ be an undirected and unweighted graph without loops or multiple edges. We assume that $n$, the number of nodes of $G$, is large and that the number of edges is much smaller than $n^2$. Networks that can be represented by this kind of graph arise in many scientific and industrial applications, including genetics, epidemiology, energy distribution, and telecommunications; see [6, 11, 13, 27]. The adjacency matrix associated with the graph $G$ is a symmetric matrix $A = [A_{ij}] \in \mathbb{R}^{n\times n}$ such that $A_{ij} = 1$ if there is an edge connecting nodes $i$ and $j$, and $A_{ij} = 0$ otherwise.

Given a large graph, it is often useful to extract numerical quantities that describe interesting global properties of the graph, such as the importance of a particular node or the ease of traveling from one node to another. It is not hard to see that for $m \geq 1$, the entry $[A^m]_{ij}$ of $A^m$ equals the number of walks of length $m$ starting at node $i$ and ending at node $j$, where a walk is an ordered list of nodes such that successive nodes are connected. Let the matrix-valued function

$$f(A) = \sum_{m=0}^{\infty} c_m A^m$$

have nonnegative coefficients $c_m$ such that the sum converges. The term $c_0 A^0$ is included for convenience and does not have a particular meaning. Then the entry

*Submitted to the journal's Methods and Algorithms for Scientific Computing section February 26, 2013; accepted for publication (in revised form) June 5, 2013; published electronically August 13, 2013.

http://www.siam.org/journals/sisc/35-4/91112.html

†Dipartimento di Matematica e Informatica, Università di Cagliari, viale Merello 92, 09123 Cagliari, Italy (kate.fenu@unica.it, rodriguez@unica.it). This research was supported in part by MIUR-PRIN grant 20083KLJEZ-003 and by Regione Sardegna grant CRP3_92.

‡Department of Mathematical Sciences, Kent State University, Kent, OH 44242 (dmarti49@kent.edu, reichel@math.kent.edu). This research was supported in part by NSF grant DMS-1115385.

$[f(A)]_{ij}$ with $i \neq j$ can be interpreted as a measure of the ease of traveling between the nodes $i$ and $j$; the entry $[f(A)]_{ii}$ quantifies the importance of node $i$; see below for further details. Generally, the coefficients $c_m$ are chosen to be nonincreasing functions of $m$ because long walks typically are less important than short ones; see, e.g., Estrada and coworkers [11, 13, 14, 15] for discussions. A popular choice [12] is $c_m = 1/m!$, which yields

$$f(A) = \exp(A). \tag{1.1}$$

Estrada and Higham [13] consider this function as well as

$$f(A) = \left(I - \frac{A}{n-1}\right)^{-1}. \tag{1.2}$$

Other matrix functions also may be of interest. For instance, in applications where all walks of length at most $\ell$ are equally useful and no walk of length larger than $\ell$ is meaningful, it is natural to consider the function

$$f(A) = \sum_{m=0}^{\ell} A^m = \left(A^{\ell+1} - I\right)(A - I)^{-1}.$$

The following definitions, which are discussed in [11, 13, 14, 15], are motivated by the discussion above:

- the *degree* of node $i$, given by $[A\mathbf{c}]_i$ with $\mathbf{c} = [1, 1, \ldots, 1]^T$, provides a measure of the importance of node $i$;
- the *$f$-subgraph centrality* of node $i$, given by $[f(A)]_{ii}$, provides a more sophisticated measure of the importance of node $i$ than its degree;
- the *$f$-communicability* between nodes $i$ and $j$, given by $[f(A)]_{ij}$, quantifies the ease of traveling between nodes $i$ and $j$.

Note that $f$-communicability and $f$-subgraph centrality are quantities of the form

$$\mathbf{u}^T f(A)\mathbf{w} \tag{1.3}$$

with $\mathbf{u}$ and $\mathbf{w}$ different or the same axis vectors $\mathbf{e}_j = [0, \ldots, 0, 1, 0, \ldots, 0]^T \in \mathbb{R}^n$. Other quantities of the form (1.3) used to characterize nodes of a graph are described in [5, 11, 13, 16]. For instance, the *$f$-starting convenience* of node $i$,

$$n\frac{\mathbf{e}_i^T f(A)\mathbf{c}}{\mathbf{c}^T f(A)\mathbf{c}},$$

with $\mathbf{c}$ as defined above, quantifies the ease of traveling from node $i$ to anywhere in the network. This is the sum of the communicabilities from node $i$ to all other nodes, scaled so that the average of the quantity over all nodes is one. This quantity is defined in [16], where also the *$f$-ending convenience* of node $i$ is introduced. The latter also can be computed by evaluating expressions of the form (1.3).

When the adjacency matrix $A$ is large, i.e., when the graph $G$ has many nodes, direct evaluation of $f(A)$ generally is not feasible. Benzi and Boito [4] applied pairs of Gauss and Gauss–Radau rules to compute upper and lower bounds for selected entries of $f(A)$. This work is based on the connection between the symmetric Lanczos process, orthogonal polynomials, and Gauss-type quadrature, explored by Golub and his collaborators in many publications; see Golub and Meurant [20] for details and references. A brief review of this technique is provided in section 4. An application of

pairs of block Gauss-type quadrature rules to simultaneously determine approximate upper and lower bounds for several entries of $f(A)$ is described in [16].

The main drawback of quadrature-based methods is that the computational effort is proportional to the number of desired bounds. Quadrature-based methods are attractive to use when bounds for only a few quantities (1.3) are to be computed. However, these methods can be expensive to use when bounds for many expressions are to be evaluated. This situation arises, for instance, when we would like to determine one or a few nodes with the largest $f$-subgraph centrality in a large graph, because this requires the computation of upper and lower bounds for all diagonal entries of $f(A)$.

In this work, we propose to bound quantities of the form (1.3) by using a partial spectral factorization of $A$. After iteratively computing a few leading eigenvalue-eigenvector pairs, bounds for every entry of $f(A)$ can be evaluated with little additional work. For instance, assume that we would like to determine the $m$ most important nodes of a graph with $n \gg m$ nodes, i.e., we would like to find the $m$ nodes with the largest $f$-subgraph centrality. Having computed a partial spectral factorization of $A$, we quickly can evaluate upper and lower bounds for all diagonal entries of $f(A)$. These bounds are used to determine a set of $\ell \geq m$ nodes that contains the $m$ nodes of interest. Finally, we compute tighter upper and lower bounds with the aid of Gauss-type quadrature rules for the $\ell$ nodes in this set to determine which ones of these nodes have the largest subgraph centrality. When $\ell \ll n$, the computational effort required by this hybrid scheme is much smaller than when Gauss quadrature is applied to determine upper and lower bounds for all diagonal entries $[f(A)]_{ii}$, $1 \leq i \leq n$. For many networks, it is possible to single out $\ell \ll n$ nodes for further study by Gauss quadrature; see section 6 for illustrations.

This paper is organized as follows. Section 2 describes how upper and lower bounds for quantities of the form (1.3) can be determined via partial spectral factorization of $A$. In particular, when $\mathbf{u}$ and $\mathbf{v}$ are axis vectors, we obtain upper and lower bounds for all the entries of $f(A)$. The case when upper and lower bounds for all diagonal entries of the matrix exponential (1.1) are desired is considered in section 3. These bounds are used to determine a subset of nodes that contains the node with the largest subgraph centrality. The partial spectral factorization is computed by a block Lanczos method, using the code `irbleigs` [1, 2]. A brief outline of this method is provided in section 3. The application of Gauss-type quadrature rules to the computation of upper and lower bounds for the entries of $f(A)$ is reviewed in section 4. Our hybrid method for bounding expressions (1.3), where we first use the partial spectral factorization to identify nodes that may be of interest and then apply Gauss-type quadrature rules to refine available bounds for these nodes, is described in section 5. Numerical examples reported in section 6 illustrate the competitiveness of the hybrid approach for large networks. Section 7 contains concluding remarks.

**2. Bounds via partial spectral factorization.** We derive bounds for expressions of the kind $\mathbf{u}^T f(A)\mathbf{w}$, with $\|\mathbf{u}\| = \|\mathbf{w}\| = 1$, in terms of a partial spectral factorization of $A$. The function $f$ is assumed to be nondecreasing and nonnegative on the spectrum of $A$. This is true, for example, for the functions (1.1) and (1.2).

Introduce the spectral factorization

$$A = V\Lambda V^T,$$

where the eigenvector matrix $V = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n] \in \mathbb{R}^{n\times n}$ is orthogonal and the eigenvalues in the diagonal matrix $\Lambda = \operatorname{diag}[\lambda_1, \lambda_2, \ldots, \lambda_n] \in \mathbb{R}^{n\times n}$, are ordered

according to $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. By definition,

$$f(A) = Vf(\Lambda)V^T = \sum_{k=1}^{n} f(\lambda_k)\mathbf{v}_k\mathbf{v}_k^T, \tag{2.1}$$

so that

$$f_{\mathbf{u},\mathbf{w}}(A) := \mathbf{u}^T f(A)\mathbf{w} = \sum_{k=1}^{n} f(\lambda_k)\tilde{u}_k\tilde{w}_k,$$

where $\tilde{u}_k = \mathbf{u}^T\mathbf{v}_k$ and $\tilde{w}_k = \mathbf{w}^T\mathbf{v}_k$.

Let the first $N$ eigenpairs $\{\lambda_k, \mathbf{v}_k\}_{k=1}^N$ of $A$ be known. Then $f_{\mathbf{u},\mathbf{w}}(A)$ can be approximated by

$$\mathbf{u}^T f(A)\mathbf{w} \approx F_{\mathbf{u},\mathbf{w}}^{(N)} := \sum_{k=1}^{N} f(\lambda_k)\tilde{u}_k\tilde{w}_k. \tag{2.2}$$

The following result shows how upper and lower bounds for $f_{\mathbf{u},\mathbf{w}}(A)$ can be determined with the aid of the first $N$ eigenpairs of $A$.

THEOREM 2.1. *Let the function $f$ be nondecreasing and nonnegative on the spectrum of $A$ and let $F_{\mathbf{u},\mathbf{w}}^{(N)}$ be defined by* (2.2). *Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$ be the $N$ largest eigenvalues of $A$ and let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$ be associated orthonormal eigenvectors. Then we have*

$$L_{\mathbf{u},\mathbf{w}}^{(N)} \leq f_{\mathbf{u},\mathbf{w}}(A) \leq U_{\mathbf{u},\mathbf{w}}^{(N)}, \tag{2.3}$$

*where*

$$L_{\mathbf{u},\mathbf{w}}^{(N)} := F_{\mathbf{u},\mathbf{w}}^{(N)} - f(\lambda_N)\left(1 - \sum_{k=1}^{N}\tilde{u}_k^2\right)^{1/2}\left(1 - \sum_{k=1}^{N}\tilde{w}_k^2\right)^{1/2},$$

$$U_{\mathbf{u},\mathbf{w}}^{(N)} := F_{\mathbf{u},\mathbf{w}}^{(N)} + f(\lambda_N)\left(1 - \sum_{k=1}^{N}\tilde{u}_k^2\right)^{1/2}\left(1 - \sum_{k=1}^{N}\tilde{w}_k^2\right)^{1/2}.$$

*Proof.* The Cauchy inequality yields

$$\begin{aligned}\left|f_{\mathbf{u},\mathbf{w}}(A) - F_{\mathbf{u},\mathbf{w}}^{(N)}\right| &= \left|\sum_{k=N+1}^{n} f(\lambda_k)\tilde{u}_k\tilde{w}_k\right| \leq f(\lambda_N)\sum_{k=N+1}^{n}|\tilde{u}_k|\,|\tilde{w}_k| \\ &\leq f(\lambda_N)\left(\sum_{k=N+1}^{n}\tilde{u}_k^2\right)^{1/2}\left(\sum_{k=N+1}^{n}\tilde{w}_k^2\right)^{1/2} \\ &= f(\lambda_N)\left(1 - \sum_{k=1}^{N}\tilde{u}_k^2\right)^{1/2}\left(1 - \sum_{k=1}^{N}\tilde{w}_k^2\right)^{1/2},\end{aligned}$$

from which (2.3) follows. ∎

COROLLARY 2.2. *Assume that the conditions of Theorem* 2.1 *hold and let $\mathbf{u} = \mathbf{w}$. Then*

$$F_{\mathbf{u},\mathbf{u}}^{(N)} \leq f_{\mathbf{u},\mathbf{u}}(A) \leq U_{\mathbf{u},\mathbf{u}}^{(N)}. \tag{2.4}$$

*Proof.* We have

$$0 \leq f_{\mathbf{u},\mathbf{u}}(A) - F_{\mathbf{u},\mathbf{u}}^{(N)} = \sum_{k=N+1}^{n} f(\lambda_k)\tilde{u}_k^2 \leq f(\lambda_N) \sum_{k=N+1}^{n} \tilde{u}_k^2 = f(\lambda_N)\left(1 - \sum_{k=1}^{N} \tilde{u}_k^2\right),$$

which implies (2.4). ∎

COROLLARY 2.3. *Let the conditions of Theorem* 2.1 *hold. Then the bounds* (2.3) *satisfy,*

$$\left| f_{\mathbf{u},\mathbf{w}}(A) - F_{\mathbf{u},\mathbf{w}}^{(N+1)} \right| \leq \left| f_{\mathbf{u},\mathbf{w}}(A) - F_{\mathbf{u},\mathbf{w}}^{(N)} \right| \tag{2.5}$$

*and*

$$\begin{aligned} L_{\mathbf{u},\mathbf{w}}^{(N)} - F_{\mathbf{u},\mathbf{w}}^{(N)} &\leq L_{\mathbf{u},\mathbf{w}}^{(N+1)} - F_{\mathbf{u},\mathbf{w}}^{(N+1)} \leq 0, \\ U_{\mathbf{u},\mathbf{w}}^{(N)} - F_{\mathbf{u},\mathbf{w}}^{(N)} &\geq U_{\mathbf{u},\mathbf{w}}^{(N+1)} - F_{\mathbf{u},\mathbf{w}}^{(N+1)} \geq 0 \end{aligned} \tag{2.6}$$

*for* $1 \leq N < n$. *For the bounds* (2.4), *we have*

$$F_{\mathbf{u},\mathbf{u}}^{(N)} \leq F_{\mathbf{u},\mathbf{u}}^{(N+1)}, \qquad U_{\mathbf{u},\mathbf{u}}^{(N)} \geq U_{\mathbf{u},\mathbf{u}}^{(N+1)}, \qquad 1 \leq N < n. \tag{2.7}$$

*Proof.* The monotonic behavior of $N \to F_{\mathbf{u},\mathbf{u}}^{(N)}$ is a consequence of the nonnegativity of $f$ on the spectrum of $A$, and the monotonic behavior of $N \to U_{\mathbf{u},\mathbf{u}}^{(N)}$ follows from the fact that $f$ is a nondecreasing function. The inequalities (2.5) and (2.6) can be shown similarly. ∎

This paper is primarily concerned with the determination of nodes with the largest $f$-subgraph centrality $[f(A)]_{ii} = \mathbf{e}_i^T f(A)\mathbf{e}_i$ of a large network. The inequalities (2.4) and (2.7) are important in this context. For notational convenience, we refer to the lower and upper bounds $F_{\mathbf{u},\mathbf{u}}^{(N)}$ and $U_{\mathbf{u},\mathbf{u}}^{(N)}$ in (2.4) as $L_{ii}^{(N)}$ and $U_{ii}^{(N)}$ when $\mathbf{u} = \mathbf{e}_i$.

The bounds (2.3) and (2.5) are relevant when we seek to determine pairs of nodes with the largest $f$-communicability. Letting $\mathbf{u} = \mathbf{e}_i$, $\mathbf{w} = \mathbf{c}/\|\mathbf{c}\| = n^{-1/2}[1, 1, \ldots, 1]^T$, and multiplying all the bounds by $\|\mathbf{c}\| = \sqrt{n}$, we can apply them to the $f$-starting convenience.

**3. Determining important nodes by partial spectral factorization.** This section describes how knowledge of the $N$ leading eigenpairs $\{\lambda_k, \mathbf{v}_k\}_{k=1}^N$ of $A$ and the bounds (2.4) with $\mathbf{u} = \mathbf{e}_i$ can be used to determine a subset of nodes that contains the nodes with the largest $f$-subgraph centrality $[f(A)]_{ii}$. We refer to the nodes with the largest $f$-subgraph centrality as the most important nodes. The function $f$ is required to be nondecreasing and nonnegative. We will comment on special computational issues that arise when $f$ is the exponential function.

Let $L_{ii}^{(N)}$ and $U_{ii}^{(N)}$ be the lower and upper bounds defined after Corollary 2.3, and let $\mathcal{L}_m^{(N)}$ denote the $m$th largest lower bound $L_{ii}^{(N)}$. Introduce the index sets

$$S_m^{(N)} = \left\{ \; i \; : \; U_{ii}^{(N)} \geq \mathcal{L}_m^{(N)} \right\}, \qquad N = 1, 2, \ldots, n.$$

It is not hard to see that if $i \notin S_m^{(N)}$, then node $i$ cannot be in the subset of the $m$ nodes with the largest $f$-subgraph centrality. Moreover, any node whose index is an element of $S_m^{(N)}$ can be in this subset.

Let $|S_m^{(N)}|$ denote the cardinality of $S_m^{(N)}$. The following relations are useful in the sequel.

COROLLARY 3.1.

$$S_m^{(n)} \subseteq S_m^{(n-1)} \subseteq \cdots \subseteq S_m^{(1)} \quad \text{and} \quad |S_m^{(n)}| \geq m. \tag{3.1}$$

*The set $S_m^{(N)}$ contains the indices for a subset of nodes that contains the set of the $m$ most important nodes. In particular, when $|S_m^{(N)}| = m$, the set $S_m^{(N)}$ contains the indices for the $m$ most important nodes.*

*Proof.* By the definition of the sets $S_m^{(N)}$, each set contains at least $m$ indices. The relations (3.1) now follow from (2.7) with $\mathbf{u} = \mathbf{e}_i$ for $1 \leq i \leq n$. The observation about the situation when $|S_m^{(N)}| = m$ is a consequence of the fact that the set $S_m^{(N)}$ contains the indices for the $m$ nodes with the largest $f$-subgraph centrality. ∎

*Remark* 3.2. The lower bound $L_{ii}^{(N)}$ defined after Corollary 2.3 usually converges to $[f(A)]_{ii}$ much faster than the corresponding upper bound $U_{ii}^{(N)}$ as $N$ increases. Therefore, for $N$ fixed, $L_{ii}^{(N)}$ typically is a better approximation of $[f(A)]_{ii}$ than $\frac{1}{2}(L_{ii}^{(N)} + U_{ii}^{(N)})$. This is due to the fact that the lower bound is a truncation of the expansion (2.1), cf. (2.2) and (2.4), while the upper bound is obtained from the lower bound by adding a sufficiently large and computable quantity.

*Remark* 3.3. A particular ordering of the lower bounds $L_{ii}^{(N)}$, $i \in S_m^{(N)}$, does not have to correspond to the same ordering of the $f$-subgraph centralities $[f(A)]_{ii}$, i.e., an inequality $L_{ii}^{(N)} > L_{jj}^{(N)}$ for $j \in S_m^{(N)} \backslash \{i\}$ does not imply that the $i$th node has the largest $f$-subgraph centrality.

We turn to some computational issues. Evaluation of the bounds (2.3) and (2.4) requires the computation of $f(\lambda_N)$. This may result in overflow when the graph contains many nodes and $f$ is the exponential function (1.1). For instance, when the computations are carried out in "double precision arithmetic," i.e., with about 16 significant decimal digits, we obtain overflow when evaluating $\exp(x)$ for $x \gtrsim 710$. This difficulty can be circumvented by replacing $A$ by $A - \mu I$, where $I$ is the identity matrix and $\mu$ is an estimate of the largest eigenvalue of $A$. We then seek to approximate $[f(A - \mu I)]_{ii}$ instead of $[f(A)]_{ii}$, where we note that

$$[f(A)]_{ii} = \exp(\mu)\, [f(A - \mu I)]_{ii}.$$

Since $A \in \mathbb{R}^{n \times n}$ is an adjacency matrix for an unweighted undirected graph without loops, its spectral radius is bounded by $n - 1$. We therefore may use $\mu = n - 1$. However, since we determine a partial spectral factorization $\{\lambda_k, \mathbf{v}_k\}_{k=1}^N$ of $A$, the largest eigenvalue $\lambda_1$ of $A$ is available. Therefore, we let $\mu = \lambda_1$ in the computed examples reported in section 6.

Another computational difficulty to overcome is that we do not know in advance how the dimension $N$ of the leading invariant subspace $\{\lambda_k, \mathbf{v}_k\}_{k=1}^N$ of $A$ should be chosen in order to obtain useful bounds (2.3) or (2.4). We will use the restarted block Lanczos method `irbleigs` described in [1, 2] to compute invariant subspaces in the examples of section 6. This method computes the leading invariant subspace $\{\lambda_k, \mathbf{v}_k\}_{k=1}^q$ of $A$ of user-chosen dimension $q$. The method applies Leja shifts to damp unwanted eigenvector components in the block Krylov subspaces generated. These shifts are constructed in the same manner as Leja points, which are suitable interpolation points when approximating analytic functions by an interpolating polynomial in a region in the complex plane. If the computed bounds (2.3) or (2.4) are

not tight enough, e.g., if $|S_m^{(q)}|$ is larger than $m$, then we restart the computations with `irbleigs` to determine the next $q$ eigenpairs $\{\lambda_k, \mathbf{v}_k\}_{k=q+1}^{2q}$ of $A$ to obtain the leading invariant subspace $\text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{2q}\}$ of $A$. This can be done by passing the already available $q$ eigenvectors $\{\mathbf{v}_k\}_{k=1}^{q}$ of $A$ to `irbleigs` using the option `opts.eigvec`. If $|S_m^{(N)}| = m$, for $N \leq 2q$, then we are done, otherwise we compute the next $q$ eigenpairs $\{\lambda_k, \mathbf{v}_k\}_{k=2q+1}^{3q}$ of $A$ with `irbleigs`, and so on. The approach outlined requires that all already computed eigenvectors be stored when determining the next batch of $q$ eigenvectors. This method for determining an invariant subspace of desired dimension is attractive when the computer used allows storage of an orthonormal basis for the already computed subspace. A comparison of `irbleigs` and the MATLAB function `eigs`, which is based on ARPACK [23], is reported in [1] and shows the former method to be competitive. The use of `irbleigs` is particularly advantageous when only few auxiliary vectors can be stored. This often is the case for large-scale problems. Moreover, `irbleigs` is a block method, while `eigs` implements a standard restarted Lanczos method. Block methods may perform more efficiently on many computers; see, e.g., Gallivan et al. [17].

When the adjacency matrix is very large and the dimension $N$ of a leading invariant subspace $\{\lambda_k, \mathbf{v}_k\}_{k=1}^{N}$ such that $|S_m^{(N)}| = m$ is fairly large, the storage requirement for a basis for this subspace may be problematic. We describe an approach to handle this situation. Note that the evaluation of the bounds (2.3) and (2.4) does not require simultaneous access to all computed eigenvectors. We therefore may reduce the storage requirement by limiting the number of eigenvectors passed to `irbleigs`. This can be achieved by calling `irbleigs` with the option `opts.sigma` as follows. This option determines eigenvalues of $A$ close to a specified value. We choose this value to be the smallest eigenvalue of the invariant subspace already computed. The option `opts.eigvec` is used to pass available eigenvectors associated with eigenvalues closest to the smallest computed eigenvalue. The number of eigenvectors passed, $M_{\max}$, is close to the largest possible number of eigenvectors that fit into fast computer memory. For definiteness, assume that $q$ new eigenpairs with eigenvalues smaller than the smallest of the already computed eigenvalues are desired. Then this approach typically gives some new eigenpairs, which are used to update the bounds (2.3) and (2.4). Also a few previously already computed eigenpairs may be recomputed. The outlined computations only require storage of $M_{\max} + q$ eigenvectors. The memory requirement therefore is modest also when $N$ is fairly large. However, typically this approach requires more matrix-vector product evaluations with $A$ than when all computed eigenvectors are stored.

We may compute more and more eigenpairs of $A$ until $N$ is such that

$$|S_m^{(N)}| = m. \tag{3.2}$$

This stopping criterion is referred to as the *strong convergence condition*. By Corollary 3.1, the set $S_m^{(N)}$ contains the indices of the $m$ nodes with the largest $f$-subgraph centrality.

The criterion (3.2) for choosing $N$ is useful if the required value of $N$ is not too large. We introduce the *weak convergence criterion* to be used for problems for which the large size of $N$ required to satisfy (3.2) makes it impractical to compute the associated bounds (2.4) with $\mathbf{u} = \mathbf{e}_i$. The weak convergence criterion is well suited for use with the hybrid algorithm for determining the most important nodes described in section 5. This criterion is designed to stop increasing $N$ when the lower bounds $L_{ii}^{(N)}$ do not increase significantly with $N$. Specifically, we stop increasing $N$, when the

average increment of the lower bounds $L_{ii}^{(N)}$, $1 \le i \le n$, is small when including the $N$th eigenpair $\{\lambda_N, \mathbf{v}_N\}$ in the bounds. The average contribution of this eigenpair to the bounds $L_{ii}^{(N)}$, $1 \le i \le n$, is

$$\frac{1}{n}\sum_{i=1}^{n} f(\lambda_N) v_{i,N}^2 = \frac{1}{n} f(\lambda_N),$$

and we stop increasing $N$ when

$$\frac{1}{n} f(\lambda_N) \le \tau \cdot \mathcal{L}_m^{(N)} \tag{3.3}$$

for a user-specified tolerance $\tau$. We use $\tau = 10^{-3}$ in the computed examples. Note that when this criterion is satisfied, but not (3.2), the nodes with index in $S_m^{(N)}$ and with the largest lowest bounds $L_{ii}^{(N)}$ are not guaranteed to be the nodes with the largest $f$-subgraph centrality.

The weak convergence criterion (3.3) may yield a set $S_m^{(N)}$ with many more indices than $m$. In particular, we may not want to compute accurate bounds for the $f$-subgraph centrality using the approach of section 4 for all nodes with index in $S_m^{(N)}$. We therefore describe how to determine a smaller index set $\mathcal{J}$, which is likely to contain the indices of the $m$ nodes with the largest $f$-subgraph centrality. Since $L_{ii}^{(N)}$ generally is a better approximation of $[f(A)]_{ii}$ than $U_{ii}^{(N)}$ (cf. Remark 3.2), we discard from the set $S_m^{(N)}$ indices for which $L_{ii}^{(N)}$ is much smaller than $\mathcal{L}_m^{(N)}$. Thus, for a user-chosen parameter $\rho > 0$, we include in the set $\mathcal{J}$ all indices $i \in S_m^{(N)}$ such that

$$\mathcal{L}_m^{(N)} - L_{ii}^{(N)} < \rho \cdot \mathcal{L}_m^{(N)}. \tag{3.4}$$

In the computed examples, we use $\rho = 10^{-1}$.

The following algorithm describes the determination of the dimension $N$ of the leading subspace and of the index set $S_m^{(N)}$. The function $f$ is the exponential function (1.1). This function may be replaced by some other nonnegative nondecreasing function such as (1.2). The algorithm requires the adjacency matrix $A$, its order $n$, and the number of nodes with largest $f$-subgraph centrality desired, $m$. We remark that the adjacency matrix $A$ does not have to be explicitly stored, only a function for evaluating matrix-block-vector products with $A$ is required. In addition, the following parameters have to be provided:

- $N_{\max}$, maximum number of iterations performed;
- $q$, number of eigenvalues computed at each `irbleigs` call;
- $M_{\max}$, maximum number of eigenvector kept in memory;
- $\tau$, tolerance used to detect *weak* convergence;
- $\rho$, tolerance used to constructed an extended list of nodes in case of weak convergence; cf. (3.4).

We comment on the computations of part one of the algorithm below.

Algorithm 1 first initializes the vectors $\boldsymbol{\ell}$, $\mathbf{u}$, and $\mathbf{s}$, whose components, at each iteration $N$, are given by

$$\ell_i = L_{ii}^{(N)}, \quad u_i = U_{ii}^{(N)}, \quad s_i = \sum_{k=1}^{N} v_{ik}^2.$$

Then, `irbleigs` is called to compute the first batch of $q$ eigenpairs and the main loop is entered. The Boolean variable "flag" is used to detect either strong or weak convergence. The parameter $N_{\max}$ specifies the maximum number of iterations, i.e.,

ALGORITHM 1. LOW-RANK APPROXIMATION, PART 1.

1: **Input:** matrix $A$ of size $n$, number $m$ of nodes to be identified,
2: tuning constants: $N_{\max}$, $q$, $M_{\max}$, $\tau$, $\rho$
3: **for** $i = 1$ **to** $n$ **do** $\ell_i, s_i = 0$ **end**
4: **call irbleigs** to compute $\{\lambda_i, \mathbf{v}_i\}_{i=1}^{q}$ such that $\lambda_i \geq \lambda_{i+1}$
5: **if** spectrum shift is active **then** $\mu = \lambda_1$ **else** $\mu = 0$ **end**
6: $N = 0$, $\overline{N} = 0$, flag = **true**
7: **while** flag **and** $(N < \min\{N_{\max}, n\})$ **and** $(\overline{N} < q)$
8: $N = N + 1$, $\overline{N} = \overline{N} + 1$
9: $f_\lambda = \exp(\lambda_N - \mu)$
10: **for** $i = 1$ **to** $n$ **do** $w_i = v_{iN}^2$ **end**
11: $\mathbf{s} = \mathbf{s} + \mathbf{w}$
12: $\boldsymbol{\ell} = \boldsymbol{\ell} + f_\lambda \cdot \mathbf{w}$
13: $\mathbf{u} = \boldsymbol{\ell} + f_\lambda(1 - \mathbf{s})$
14: let $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_n]$ be an index permutation such that $\ell_{\sigma_i} \geq \ell_{\sigma_{i+1}}$
15: $L_{\max} = \ell_{\sigma_m}$
16: $S^{(N)} = \{i : u_i \geq L_{\max}\}$
17: flag = $(|S^{(N)}| > m)$ **and** $(\frac{1}{n} f_\lambda > \tau \cdot L_{\max})$
18: **if** $\overline{N} = q$
19: **if** $N < M_{\max}$
20: **call irbleigs** to compute $\{\lambda_{N+i}, \mathbf{v}_{N+i}\}_{i=1}^{q}$ such that $\lambda_{N+i} \geq \lambda_{N+i+1}$
21: $\overline{N} = 0$
22: **else**
23: **call irbleigs** to compute e-values $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_q$ closest to $\lambda_N$
24: $r = \arg\min_i |\nu_i - \lambda_N|$
25: $\lambda_{N+i} = \nu_{r+i}$, $i = 1, \ldots, q - r$; $\mathbf{v}_{N+i}$ are associated eigenvectors
26: $\overline{N} = r$
27: **end if**
28: **end if**
29: **end while**

the maximum number of times the loop made up of lines 7–29 is executed. We found it beneficial to introduce the auxiliary parameter $\overline{N}$ to keep track of how many eigenvectors from the current batch are being used. When $\overline{N} = q$, a new batch of eigenpairs is computed.

The bounds (2.4) with $\mathbf{u} = \mathbf{e}_i$ are computed in lines 8–13; the vector of indices $\boldsymbol{\sigma}$ contains a permutation which yields the lower bounds $\ell_i$ in decreasing order. In line 16 the set $S_m^{(N)}$ is constructed; subsequently, the exit condition is checked. Lines 18–28 give a new batch of eigenpairs. Either one of the two kinds of restarts discussed above may be applied. If $N$ is smaller than $M_{\max}$, then we compute the next $q$ eigenpairs and orthogonalize the new eigenvectors against the available eigenspace. If, instead, $N \geq M_{\max}$, then we seek to determine the $q$ eigenvalues close to the smallest available eigenvalue $\lambda_N$, and we select those that are smaller than $\lambda_N$.

Algorithm 2 describes the continued computations. The computations of the algorithm are commented on below. Lines 30–42 of the algorithm determine whether weak or strong convergence has been achieved. The variable "info" contains this information. A list of the desired nodes $\mathcal{N}$ is formed in lines 43–46, where the subgraph

ALGORITHM 2. LOW-RANK APPROXIMATION, PART 2.

```
30: if flag
31:     j = |S^(N)| − m
32:     info = 2     % no convergence
33: else
34:     if |S^(N)| > m
35:         J = {i : i > m, L_max − ℓ_σi < ρ · L_max}
36:         j = min(|J|, 100), j = max(j, 5)
37:         info = 1     % weak convergence
38:     else
39:         j = 0
40:         info = 0     % strong convergence
41:     end if
42: end if
43: for i = 1 to m + j
44:     N_i = σ_i
45:     V_i = e^μ · ℓ_σi
46: end if
47: Output: list of nodes N, subgraph centralities V,
48:             spectrum shift μ, iterations N, info
```

centralities also are updated, keeping in mind the spectrum shift at line 9 of Algorithm 1. We remark that the MATLAB implementation of Algorithm 2 contains some features not described in the algorithm. For instance, we only apply the correction due to the spectrum shift when this does not cause overflow.

**4. Bounds via Gauss quadrature.** This section reviews a technique proposed by Golub and his collaborators to bound certain matrix functionals with the aid of Gauss-type quadrature rules; see, e.g., [7, 19, 21]. A nice overview is presented in [20]. Applications to network analysis have recently been described in [4]. We apply the technique of this section to improve the bounds obtained by partial spectral factorization for nodes that are deemed to be of interest.

Let $\mathbf{w} = \mathbf{u}$ in (1.3) and assume that $\mathbf{u}$ is a unit vector. Then, using (2.1), the expression can be written as a Stieltjes integral

$$\mathbf{u}^T f(A)\mathbf{u} = \sum_{i=1}^{n} f(\lambda_i)\omega_i^2 = \int f(t)d\omega(t), \tag{4.1}$$

where $\omega$ is a piecewise constant distribution function with jumps at the eigenvalues $\lambda_i$ of $A$. The integral can be approximated by a $k$-point Gauss quadrature rule,

$$\mathcal{G}_k f = \sum_{i=1}^{k} f(\theta_i^{(k)})\omega_i^{(k)}. \tag{4.2}$$

Consider the application of $k$ steps of the Lanczos method to the matrix $A$ with initial vector $\mathbf{u}$. Generically, this yields the decomposition

$$AU_k = U_k T_k + t_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^T, \tag{4.3}$$

where the matrix $U_k \in \mathbb{R}^{n\times k}$ has orthonormal columns with $U_k\mathbf{e}_1 = \mathbf{u}$, $T_k \in \mathbb{R}^{k\times k}$ is symmetric and tridiagonal, $\mathbf{u}_{k+1} \in \mathbb{R}^n$ is a unit vector that is orthogonal to the

columns of $U_k$, $t_{k+1,k}$ is a positive scalar, and $\mathbf{e}_j$ denotes the $j$th axis vector of appropriate dimension. Then the Gauss rule (4.2) can be written as

$$\mathcal{G}_k f = \mathbf{e}_1^T f(T_k)\mathbf{e}_1; \tag{4.4}$$

see, e.g., [19, 20] for a proof.

Let $\theta_0$ be an upper bound for the largest eigenvalue of $A$, i.e., $\theta_0 \geq \lambda_1$. Consider the $(k+1)$-point Gauss–Radau quadrature rule with a fixed node at $\theta_0$ for approximating the Stieltjes integral (4.1). It can be expressed as

$$\hat{\mathcal{G}}_{k+1} f = \mathbf{e}_1^T f(\hat{T}_{k+1})\mathbf{e}_1, \tag{4.5}$$

where $\hat{T}_{k+1} \in \mathbb{R}^{(k+1)\times(k+1)}$ is a symmetric tridiagonal matrix with leading $k \times k$ principal submatrix $T_k$ and $\mathbf{e}_{k+1}^T \hat{T}_{k+1}\mathbf{e}_k = t_{k+1,k}$. The last diagonal entry of $\hat{T}_{k+1}$ is determined so that the matrix has the eigenvalue $\theta_0$. This entry can be computed in only $\mathcal{O}(k)$ arithmetic floating point operations; see [18, 19, 20] for details.

Let $f$ be the exponential function (1.1) and $\mathbf{u} = \mathbf{e}_i$. Then the remainder formulas for Gauss and Gauss–Radau quadrature rules yield that, generically,

$$\mathcal{G}_{k-1} f < \mathcal{G}_k f < [f(A)]_{ii} < \hat{\mathcal{G}}_{k+1} f < \hat{\mathcal{G}}_k f;$$

see, e.g., [20, 24] for details.

In section 6, we let $\mathbf{u} = \mathbf{w} = \mathbf{e}_i$ in (1.3) and apply pairs of Gauss and Gauss–Radau rules to determine bounds for the $f$-subgraph centrality for the nodes in the set $\mathcal{J}$ determined by Algorithms 1 and 2. We compute a Lanczos decomposition (4.3) for each index in $\mathcal{J}$ and choose the number of Lanczos steps $k$ as small as possible so that the bound

$$\frac{|\mathcal{G}_k f - \hat{\mathcal{G}}_{k+1} f|}{|\mathcal{G}_k f|} \leq \tau_{\mathcal{G}}$$

holds for a user-specified tolerance $\tau_{\mathcal{G}}$. In the computed examples, we use $\tau_{\mathcal{G}} = 10^{-3}$.

We evaluate the matrix exponentials in (4.4) and (4.5) by using the spectral factorizations of the tridiagonal matrices. For instance,

$$T_k = W \operatorname{diag}[\theta_1, \theta_2 \ldots, \theta_k] W^T,$$

with $W \in \mathbb{R}^{k\times k}$ and $W^T W = I$, yields

$$\exp(T_k - \mu I) = W \operatorname{diag}[\exp(\theta_1 - \mu), \exp(\theta_2 - \mu), \ldots, \exp(\theta_k - \mu)] W^T. \tag{4.6}$$

Shifting the spectrum by $\mu = \lambda_1$ avoids overflow.

The Lanczos method is said to break down when $t_{k+1,k}$ in (4.3) vanishes. Then the spectrum of $T_k$ is a subset of the spectrum of $A$ and the Gauss rule (4.4) yields the exact $f$-subgraph centrality. The Lanczos method is terminated when $t_{k+1,k} \geq 0$ is "tiny."

We conclude this section by noting that pairs of Gauss and Gauss–Radau quadrature rules can be applied to compute bounds for expressions of the form (1.3) for any functions that are analytic on the convex hull of support of the measure and whose derivatives are of constant sign on this set. The situation when $\mathbf{u} \neq \mathbf{w}$ can be handled by writing (1.3) in the form

$$\mathbf{u}^T f(A)\mathbf{w} = \frac{1}{4}\left((\mathbf{u}+\mathbf{w})^T f(A)(\mathbf{u}+\mathbf{w}) - (\mathbf{u}-\mathbf{w})^T f(A)(\mathbf{u}-\mathbf{w})\right). \tag{4.7}$$

The choice $\mathbf{u} = \mathbf{e}_i$ and $\mathbf{w} = \mathbf{e}_j$ allows the computation of upper and lower bounds for the $f$-communicability between the nodes $i$ and $j$. Computations with these and other choices of vectors $\mathbf{u} \neq \mathbf{w}$ are illustrated in [4, 16].

**5. The hybrid method.** We summarize the computations with our hybrid method. The method first computes a partial spectral factorization of the adjacency matrix $A$ and applies it to determine which nodes might be the most interesting ones with respect to the criterion chosen. Section 3 discussed the situation when we would like to determine the nodes of a large network with the largest $f$-subgraph centrality. We also may be interested in determining the nodes with the largest $f$-communicability. The partial spectral factorization helps us find a set of candidate nodes that contains the nodes of interest. More accurate upper and lower bounds for expressions of the form (1.3) for the candidate nodes are then computed with the aid of Gauss quadrature rules. For example, this approach allows us to compute the node with the largest $f$-subgraph centrality of a large graph without evaluating pairs of Gauss and Gauss–Radau rules for every node of the network, i.e., for every diagonal entry of the adjacency matrix. The evaluation of Gauss-type rules for every diagonal entry can be expensive for large graphs. The computations with our hybrid method typically are considerably cheaper. This is illustrated in the following section. Similarly, if we are interested in finding the nodes with the largest $f$-communicability, then we use (4.7) and compute more accurate upper and lower bounds for the candidate nodes with Gauss and Gauss–Radau rules.

**6. Computed examples.** This section presents a few examples that illustrate the performance of the methods discussed in the paper. All computations were conducted in MATLAB version 7.11 (R2010b) 64-bit for Linux, in double precision arithmetic, on an Intel Core i7-860 computer, with 8 Gb RAM. The function $f$ is the exponential function (1.1).

The examples fall into three categories. The first set consists of synthetic networks which recently have been used by many authors to test the performance of computational methods. These examples are not based on real data, but reproduce typical properties of complex networks, e.g., small world effect, power law degree distribution, etc.; see [31]. They allow us to choose the dimension of the network as well as the probability value upon which they depend. The second set consists of five networks that arise in real-world applications and have publicly available data sets. Finally, we analyze a new application, which is gaining an increasing interest in software engineering, namely, networks describing the dependency between modules in large software projects.

**6.1. Synthetic networks.** In the initial examples, we consider nine classes of random symmetric adjacency matrices that were generated with the functions erdrey, geo, gilbert, kleinberg, lockandkey, pref, renga, smallw, and sticky in the MATLAB package CONTEST by Taylor and Higham [31]. For instance, the function geo [29] generates a random symmetric adjacency matrix $A$ as follows: after determining $n$ randomly distributed points $\mathbf{x}_i$ on the unit square, an edge is inserted between nodes $i$ and $j$ if $\|\mathbf{x}_i - \mathbf{x}_j\| < r$, where by default $r = \sqrt{1.44/n}$; see [31] for details on the adjacency matrices produced by the functions. The parameters for each of these functions are chosen to be their default values. Figure 6.1 shows the sparsity pattern of a typical adjacency matrix of order $1000 \times 1000$ for each kind of graph generated, as well as the number of nonzero entries of each matrix. In all computed examples, we choose the parameters $q = 20$, $N_{\max} = 300$, and $M_{\max} = 200$ for Algorithms 1 and 2.

Our first experiment is concerned with determining the node of a graph with the largest subgraph centrality, or a small set of nodes containing this node. We refer to the node with the largest subgraph centrality as the most important node. For each

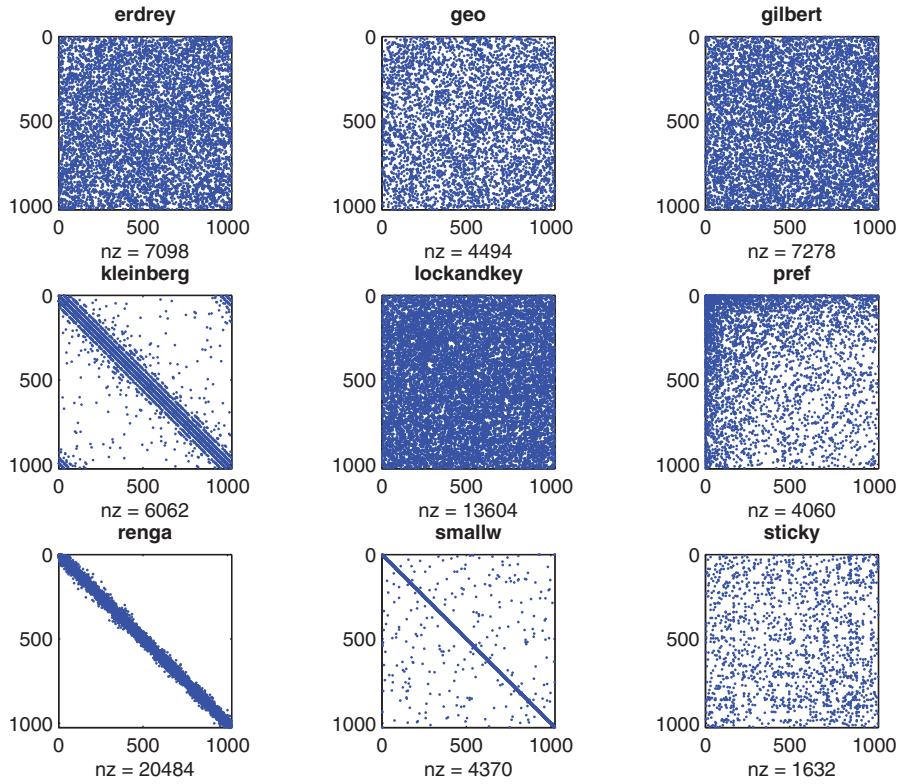


FIG. 6.1. *The sparsity pattern of a typical* $1000 \times 1000$ *matrix for each one of the nine kinds of random adjacency matrices used in the computed examples. The parameter nz shows the number of nonzero entries of each matrix.*

type of adjacency matrix, we used the bounds (2.4) with $\mathbf{u} = \mathbf{e}_i$ for $1 \leq i \leq n$ to determine sets of $M = 2^k$ nodes that contain the most important node of the graph for $k = 0, 1 \ldots, 9$. Table 6.1 reports, for each type of adjacency matrix and for each value of $M$, the average and standard deviation of the number of eigenpairs required to achieve $|S^{(N)}| \leq M$ for a sample of 100 randomly generated adjacency matrices of each kind.

For the matrix classes lockandkey, pref, renga, and sticky, the most important node in the network is correctly identified using only a fairly small number of eigenpairs; on average 12, 2, 25, and 4, respectively. For this kind of adjacency matrix, the approach of section 3 provides a powerful method for determining the most important node or set of nodes in a large graph and for estimating subgraph centralities. The method also can be used to bound communicabilities of interest.

Turning to the adjacency matrices for graphs of the type geo, the partial spectral factorization method was able to determine a set of four nodes that contained the most important node using a modest number of eigenpairs, on average 37 eigenpairs were needed. However, successful identification of the most important node required on average the much larger number of 228 eigenpairs. We made similar observations for adjacency matrices of the types erdrey, gilbert, kleinberg, and smallw. This illustrates that the low-rank approximation approach of section 3 might not be able to identify the most important node(s) with a fairly small computational effort. However, the low-rank approximation approach may be useful for generating a short list of

TABLE 6.1

*Mean and standard deviation (sdev) for nine classes of random adjacency matrices of order $n = 1024$ of the number $N$ of eigenpairs required to achieve $|S^{(N)}| \leq M$ for $M = 2^k$, $0 \leq k \leq 9$. The sample size is 100 matrices of each kind.*

| Test matrix | $M$ | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| erdrey | mean | 65 | 69 | 75 | 82 | 92 | 104 | 121 | 147 | 187 | 261 |
| | sdev | 12 | 13 | 15 | 18 | 21 | 25 | 32 | 45 | 63 | 128 |
| geo | mean | 11 | 11 | 11 | 12 | 13 | 17 | 24 | 37 | 97 | 228 |
| | sdev | 6 | 6 | 6 | 7 | 9 | 12 | 14 | 22 | 194 | 350 |
| gilbert | mean | 65 | 69 | 75 | 83 | 93 | 106 | 125 | 151 | 200 | 287 |
| | sdev | 13 | 14 | 16 | 19 | 23 | 29 | 40 | 52 | 92 | 170 |
| kleinberg | mean | 113 | 121 | 130 | 140 | 152 | 167 | 185 | 212 | 263 | 337 |
| | sdev | 10 | 11 | 14 | 17 | 20 | 27 | 35 | 49 | 94 | 139 |
| lockandkey | mean | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 12 |
| | sdev | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 4 | 27 |
| pref | mean | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | sdev | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| renga | mean | 17 | 17 | 18 | 18 | 19 | 20 | 20 | 21 | 23 | 25 |
| | sdev | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 4 |
| smallw | mean | 224 | 227 | 243 | 245 | 248 | 266 | 312 | 342 | 406 | 607 |
| | sdev | 18 | 19 | 25 | 25 | 26 | 47 | 98 | 134 | 196 | 273 |
| sticky | mean | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 4 |
| | sdev | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 3 | 6 |

TABLE 6.2

*Results obtained by the low-rank approximation method with termination due to strong or weak convergence. The table shows the number of failures, the number $N$ of eigenpairs required to satisfy the specified termination criterion, the execution time in seconds, and in the case of weak convergence the cardinality of the list $\mathcal{N}$ of candidate nodes. Each test was repeated 10 times with $4096 \times 4096$ adjacency matrices.*

| | Strong convergence | | | Weak convergence | | |
|---|---|---|---|---|---|---|
| Test matrix | Fail | $N$ | Time | $N$ | Time | $\lvert\mathcal{N}\rvert$ |
| erdrey | 3 | 304 | 4.2e+01 | 54 | 2.7e+00 | 10 |
| geo | 1 | 198 | 1.2e+01 | 4 | 2.3e-01 | 10 |
| gilbert | 4 | 345 | 5.0e+01 | 50 | 2.8e+00 | 10 |
| kleinberg | 9 | 346 | 4.1e+01 | 150 | 9.7e+00 | 11 |
| lockandkey | 0 | 35 | 3.4e+00 | 2 | 5.1e-01 | 8 |
| pref | 0 | 6 | 2.6e-01 | 2 | 2.8e-01 | 10 |
| renga | 0 | 112 | 2.0e+00 | 45 | 8.6e-01 | 13 |
| smallw | 9 | 451 | 7.7e+01 | 288 | 4.8e+01 | 1654 |
| sticky | 1 | 4 | 2.8e-01 | 2 | 2.6e-01 | 8 |

candidate nodes, whose subgraph centralities can be accurately determined by Gauss quadrature.

Table 6.2 shows the performance of the low-rank approximation method (Algorithms 1 and 2) when applied to test matrices of order $n = 4096$ from CONTEST. Each type of matrix was randomly generated 10 times. We wanted to identify the five most important nodes in the correct order. In the first set of experiments, we terminated the computations when the strong convergence criterion, $|S_5^{(N)}| = 5$, was satisfied. By Corollary 3.1, the five indices of $S_5^{(N)}$ are for the five nodes with largest $f$-subgraph centrality. The column "fail" reports the number of times the relative size of the lower bounds $L_{ii}^{(N)}$ does not convey the correct relative size of the $f$-subgraph centrality of the nodes. Table 6.2 also shows the number $N$ of eigenpairs required to satisfy the strong convergence criterion and the computation time required. Both the value of $N$ and the computation time are averages over 10 runs with each kind of

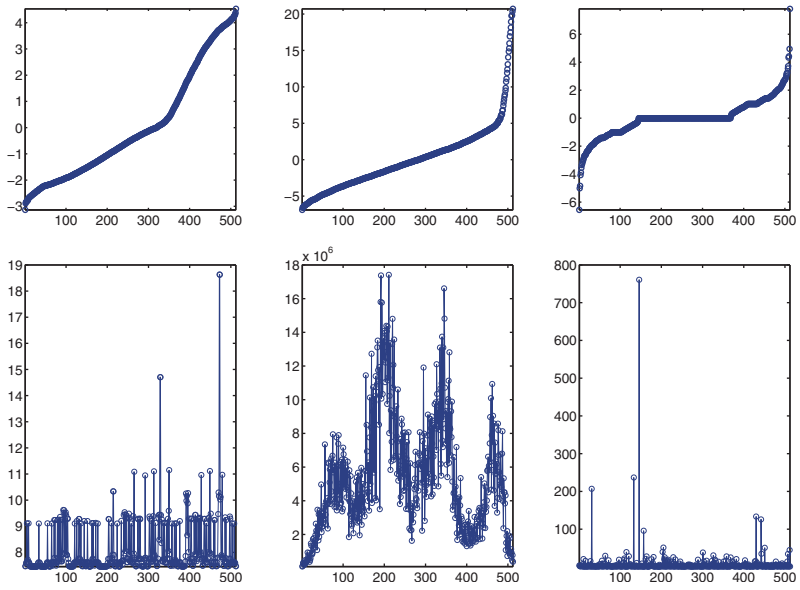


FIG. 6.2. *From right to left: Eigenvalue distribution (top) and subgraph centralities (bottom) for the networks* smallw, renga, *and* sticky *with* $n = 512$ *nodes.*

matrix. The timings are for irbleigs with convergence tolerance $1 \cdot 10^{-3}$. This tolerance is used in all computations for this section.

The last three columns of Table 6.2 are obtained when terminating the low-rank approximation method with the weak convergence criterion (3.3). The columns display, in order, the number of eigenpairs computed, the execution time, and the number of candidate nodes included in the resulting list $\mathcal{N}$. The table shows that for many of the graphs, a large number of eigenpairs is required to satisfy the strong convergence criterion, while the low-rank method for all graphs succeeds in determining a small list of candidate nodes in fairly short time, with the possible exception of the smallw graphs. The latter graphs are particularly difficult for the low-rank method. We believe that this is due to the fact that almost all the nodes of a typical smallw network have quantized subgraph centralities, in the sense that each value is very close to another one among a small number of possible values. This leads to that a large number $N$ of eigenpairs are required to satisfy the strong or weak convergence criteria. This is illustrated in Figure 6.2, which displays the eigenvalues and the subgraph centralities for the nodes of three test networks with 512 nodes each. It is immediate to observe that the eigenvalue decay alone is not sufficient to predict the speed of convergence of the algorithm.

Columns 2 and 3 of Table 6.3 report the number of matrix-vector product evaluations and the computational time (in seconds) required when determining the five nodes with the largest $f$-subgraph centrality for graphs with $n = 4096$ nodes of the type indicated, by evaluating pairs of Gauss and Gauss–Radau quadrature rules as described in section 4. A matrix-vector product with a block vector with $k$ columns is counted as $k$ matrix-vector product evaluations. The performance of the hybrid method described in section 5 is shown in columns 4 and 5. The hybrid method can be seen to require fewer matrix-vector product evaluations and shorter execution time. The number of matrix-vector products and the execution times are averages over 10 runs with randomly generated graphs of the appropriate kind. Table 6.3 compares

TABLE 6.3

*Comparison of Gauss quadrature and the hybrid algorithm. Each test is repeated* 10 *times with* $n = 4096$. *For each kind of adjacency matrix, we report the average number of matrix-vector product evaluations required (mvp) and the average execution time in seconds.*

| | Gauss | | Hybrid | | `expm` | `eig` |
|---|---|---|---|---|---|---|
| Test matrix | mvp | Time | mvp | Time | Time | Time |
| erdrey | 28663 | 1.5e+01 | 4875 | 3.0e+00 | 5.5e+02 | 1.7e+02 |
| geo | 19913 | 9.6e+00 | 496 | 2.3e-01 | 7.6e+01 | 1.5e+02 |
| gilbert | 28663 | 1.5e+01 | 4558 | 2.7e+00 | 5.6e+02 | 1.7e+02 |
| kleinberg | 22343 | 1.1e+01 | 11710 | 1.0e+01 | 2.7e+01 | 1.7e+02 |
| lockandkey | 32766 | 2.0e+01 | 1054 | 5.6e-01 | 5.2e+02 | 1.7e+02 |
| pref | 36607 | 1.8e+01 | 587 | 2.7e-01 | 4.6e+02 | 1.6e+02 |
| renga | 36087 | 2.2e+01 | 1591 | 9.5e-01 | 9.7e+01 | 1.6e+02 |
| smallw | 19266 | 8.6e+00 | 77979 | 5.5e+01 | 9.5e+00 | 1.6e+02 |
| sticky | 21792 | 1.0e+01 | 650 | 2.9e-01 | 1.3e+02 | 1.5e+02 |

these approaches with two classical methods, namely, the evaluation of the matrix exponential using the MATLAB function `expm`, which is based on Padé approximation, and complete spectral factorization with spectrum shift of the adjacency matrix using the MATLAB function `eig`; see (4.6). All methods compared in Table 6.3 correctly identified the five most important nodes of each network. The table shows the Gauss quadrature approach to be faster than `expm` and `eig`, and the hybrid method to be the fastest, except for the smallw network. The execution times for the Gauss quadrature and hybrid algorithms, as well as for `expm`, change significantly with the network. We note that the Gauss quadrature and hybrid algorithms require far less storage space than the `expm` function, which needs to allocate up to six matrices of the same size as the input matrix.

We also applied the hybrid method to smaller networks of the same kind as used for Table 6.3. Specifically, we generated networks with 512, 1024, and 2048 nodes. The hybrid method successfully determined the five nodes with the largest $f$-subgraph centrality for these graphs.

Having at our disposal a set of synthetic examples, whose size can be chosen at will, makes it possible to investigate the scalability of the methods considered. We let $n = 100, 200, \ldots, 2000$ and, for each of the CONTEST networks, we measured the execution time corresponding to the four algorithms of Table 6.3. There are essentially four classes of results, depicted in Figure 6.3. Most of the examples behave similarly to the lockandkey network: the `expm` function is convenient only for small sizes and it is generally slower than the approach based on eigenvalues (`eig`). Gauss quadrature is faster than the first two methods when $n$ gets large enough. The hybrid method is the fastest for almost all large and small networks.

In the kleinberg example the performance of the four methods is quite similar, but still the hybrid approach is slightly faster. In the renga network the `expm` function is faster than `eig` and Gauss quadrature, but all of them are slower than `hybrid`. Similar results hold for geo. The smallw results are totally different from the others: the hybrid method behaves like `eig`, and `expm` is the fastest method.

The block Lanczos method implemented by the MATLAB function `irbleigs` requires a user to select block size. Figure 6.4 shows the influence of the block size $k$ on the execution time required by the hybrid method. The figure displays timings for block sizes $k = 1, 2, \ldots, 10$, for a particular realization of four kinds of adjacency matrices of order 4096. The effect of the block size on the execution time is further illustrated in Table 6.4, where the performance of the hybrid method for block size

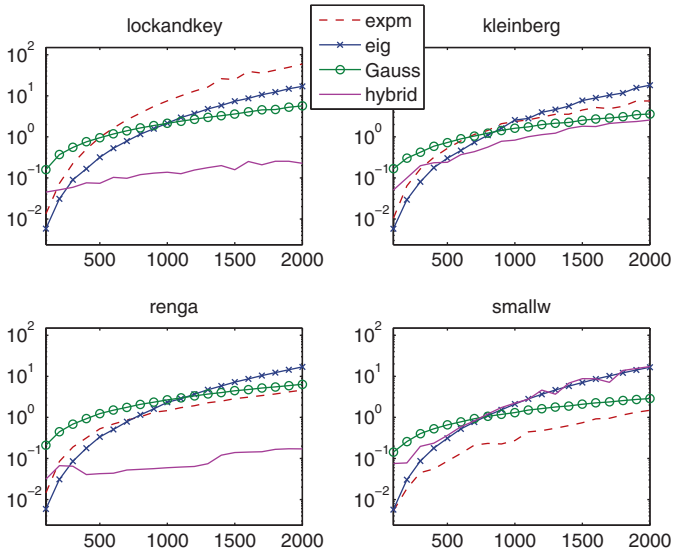


FIG. 6.3. *Execution time for the algorithms considered when the size of the network varies.*

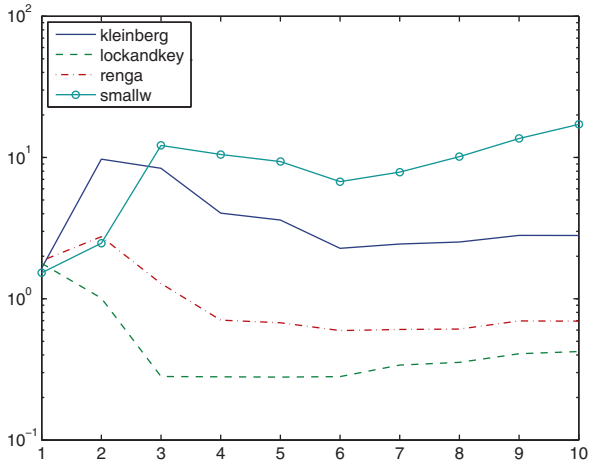


FIG. 6.4. *Hybrid algorithm: execution time versus block size for four classes of adjacency matrices,* $n = 4096$.

$k = 1$ is compared with the results in Table 6.3, obtained for block size $k = 5$. We observed in our experiments that for most matrices the execution times were the smallest for block size about 5 or 6, and only in a few cases the computation was faster with $k = 1$; see the matrix smallw in Figure 6.4 and Table 6.4. For this reason, we used block size $k = 5$ in the computed examples reported above.

**6.2. Real-world networks.** The following examples come from real-world applications. The first network (2114 nodes, 4480 edges) describes the protein interaction network for yeast: each edge represents an interaction between two proteins [22, 30]. The data set was originally included in the Notre Dame Networks Database and is now available at [3]. The eigenvalue distribution of the adjacency matrix is shown in Figure 6.5 (left).

TABLE 6.4

*Comparison between hybrid algorithms with block sizes $k = 5$ and $k = 1$. Each test is repeated 10 times, with $n = 4096$.*

| Test matrix | Hybrid, $k = 5$ mvp | Hybrid, $k = 5$ Time | Hybrid, $k = 1$ mvp | Hybrid, $k = 1$ Time | expm Time | eig Time |
|---|---|---|---|---|---|---|
| erdrey | 4875 | 3.0e+00 | 2190 | 1.7e+00 | 5.5e+02 | 1.7e+02 |
| geo | 496 | 2.3e-01 | 1044 | 7.1e-01 | 7.6e+01 | 1.5e+02 |
| gilbert | 4558 | 2.7e+00 | 2181 | 1.6e+00 | 5.6e+02 | 1.7e+02 |
| kleinberg | 11710 | 1.0e+01 | 2271 | 1.6e+00 | 2.7e+01 | 1.7e+02 |
| lockandkey | 1054 | 5.6e-01 | 2106 | 1.7e+00 | 5.2e+02 | 1.7e+02 |
| pref | 587 | 2.7e-01 | 856 | 5.6e-01 | 4.6e+02 | 1.6e+02 |
| renga | 1591 | 9.5e-01 | 2208 | 1.7e+00 | 9.7e+01 | 1.6e+02 |
| smallw | 77979 | 5.5e+01 | 2271 | 1.4e+00 | 9.5e+00 | 1.6e+02 |
| sticky | 650 | 2.9e-01 | 662 | 4.2e-01 | 1.3e+02 | 1.5e+02 |

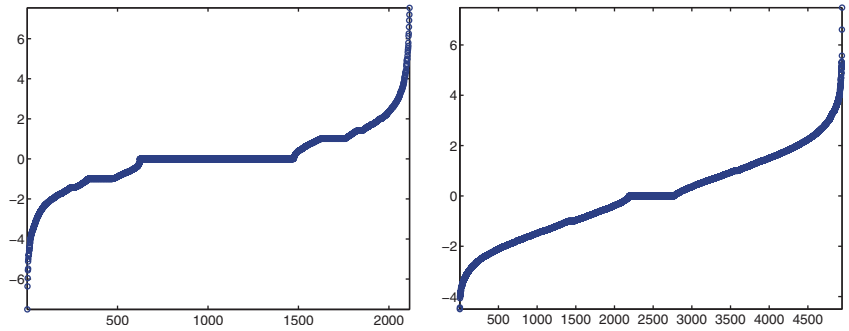


FIG. 6.5. *Eigenvalues of the adjacency matrix of the* yeast *(left) and* power *(right) networks.*

The second network (4941 nodes, 13188 edges) is an undirected unweighted representation of the topology of the western states power grid of the United States, compiled by Watts and Strogatz [33], and is now available at [28]. Figure 6.5 (right), reports the eigenvalues of the adjacency matrix.

The third example is a symmetrized snapshot of the structure of the Internet at the level of autonomous systems (22963 nodes, 96872 edges), reconstructed from BGP (Border Gateway Protocol) tables posted by the University of Oregon Route Views Project. This snapshot was created by Newman from data for July 22, 2006, and is available at [28].

The fourth example is the collaboration network of scientists posting preprints on the condensed matter archive at www.arXiv.org [26], and consists of 40421 nodes and 351304 edges. This version is based on preprints posted to the archive between January 1, 1995 and March 31, 2005. The original network is weighted; here we consider an unweighted version. The data set is available at [28].

The fifth and largest example describes all the user-to-user links (*friendship*) from the Facebook New Orleans networks. Each edge between two users means that the second user appeared in the first user's friend list. The network (63731 nodes, 1545686 edges) was studied in [32], and the data set is available at [25]. We symmetrized the adjacency matrix since the friendship relation in Facebook is reflexive.

Table 6.5 shows, for each of the above networks, the results obtained by the low rank approximation, either with strong or weak convergence test, when identifying the five most important nodes. In the two smaller examples the strong convergence criterion requires a large number of eigenpairs, but the weak convergence test is satis-

TABLE 6.5

*Results obtained by the low-rank approximation algorithm with both strong and weak convergence criteria. The table reports the number of failures, the number $N$ of eigenpairs required to reach convergence, the execution time, and, in the case of weak convergence, the cardinality of the list $\mathcal{N}$ of candidate nodes.*

| | | | Strong convergence | | | Weak convergence | | |
|---|---|---|---|---|---|---|---|---|
| Network | Nodes | Edges | Fail | $N$ | Time | $N$ | Time | $\lvert\mathcal{N}\rvert$ |
| yeast | 2114 | 4480 | 0 | 66 | 1.5e+00 | 7 | 1.2e-01 | 10 |
| power | 4941 | 13188 | 0 | 230 | 3.4e+01 | 3 | 5.8e-01 | 10 |
| internet | 22963 | 96872 | – | 2 | 1.6e+00 | 2 | 1.7e+00 | 5 |
| collaborations | 40421 | 351304 | – | 2 | 3.8e+00 | 2 | 3.6e+00 | 5 |
| facebook | 63731 | 1545686 | – | 2 | 1.5e+01 | 2 | 1.4e+01 | 5 |

TABLE 6.6

*Comparison of Gauss quadrature and hybrid algorithms. For each matrix, we report the number of matrix-vector product evaluations (mvp) and the execution time in seconds.*

| | | Gauss | | Hybrid | | expm | eig |
|---|---|---|---|---|---|---|---|
| Network | Nodes | mvp | Time | mvp | Time | Time | Time |
| yeast | 2114 | 9028 | 3.0e+00 | 367 | 1.4e-01 | 1.2e+01 | 1.5e+01 |
| power | 4941 | 23317 | 1.5e+01 | 759 | 6.4e-01 | 3.3e+01 | 2.6e+02 |
| internet | 22963 | 236345 | 3.8e+02 | 679 | 2.4e+00 | – | – |
| collaborations | 40421 | 431146 | 1.2e+03 | 835 | 4.5e+00 | – | – |
| facebook | 63731 | 801960 | 7.9e+03 | 859 | 1.2e+01 | – | – |

fied when the candidate list contains just 10 nodes. On the contrary, only 2 eigenpairs suffice to identify the five most important nodes of the three largest networks. We do not specify a value in the column labeled "fail" of Table 6.5 for the largest networks, because we cannot evaluate expm in reasonable time when $n > 10^4$ and compare with the ordering obtained when using this function. However, we note that both the strong and weak convergence of Algorithms 1 and 2, as well as Gauss quadrature and the hybrid algorithms, produce the same five nodes in the same order.

In Table 6.6, the number of matrix-vector product evaluations and the execution time corresponding to Gauss quadrature, the hybrid method, `expm`, and `eig`, are compared. The computations required for the Gauss quadrature method and for the MATLAB functions expm and eig can be seen to be quite time consuming for large matrix sizes. Results for expm and eig for the largest networks therefore are not reported when $n > 10^4$. The results achieved with the hybrid method are very encouraging and illustrate the possibility of applying hybrid methods to large-scale problems.

**6.3. Software networks.** One of the main issues in modern software engineering is to apply certain metrics to software packages in order to gain information about their properties. Concas et al. [8, 9] associate a graph to a software package and study the connection between properties of the graph and the occurrence of bugs during the package development. We considered the software package Netbeans (http://www.netbeans.org/), an integrated development environment. This choice is motivated by the fact that for this package, the source code for several versions is available and so is complete information about the localization of bugs in different versions and software modules. The Netbeans system is written in Java and its classes are contained in source files referred to as compilation units (CU). A CU generally contains just one class, but may occasionally contain two or more classes. A software network is a graph, in which the CUs are nodes and the directed edges correspond to a CU referencing another one. The resulting network is directed, not connected, and

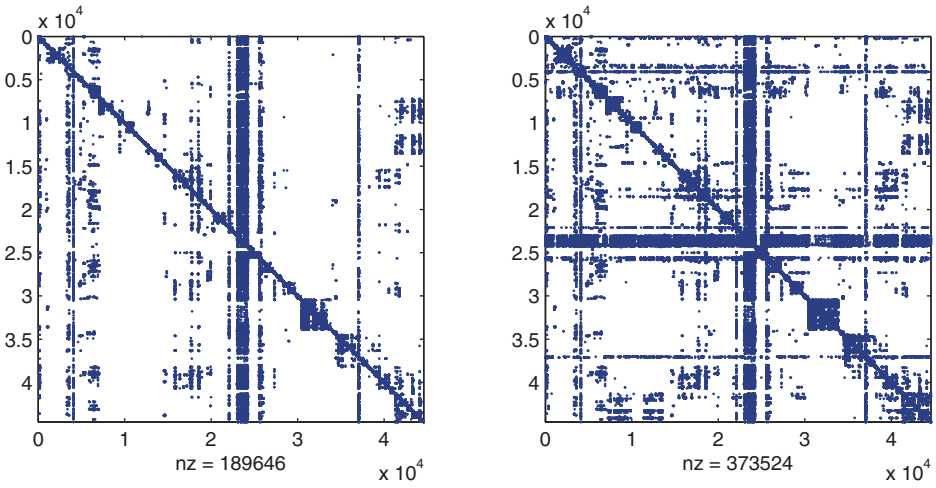


FIG. 6.6. *The sparsity pattern of the Netbeans network: original adjacency matrix (left), symmetrized version (right). The parameter nz shows the number of nonzero entries of each matrix.*

TABLE 6.7

*Results obtained by the low-rank approximation algorithm with both strong and weak convergence criteria. The table reports the number of failures, the number $N$ of eigenpairs required to reach convergence, the execution time, and, in the case of weak convergence, the cardinality of the list $\mathcal{N}$ of candidate nodes.*

| | | | Strong convergence | | | Weak convergence | | |
|---|---|---|---|---|---|---|---|---|
| Network | Nodes | Edges | Fail | $N$ | Time | $N$ | Time | $\lvert\mathcal{N}\rvert$ |
| profiler | 1231 | 4161 | 0 | 2 | 3.3e-01 | 2 | 1.1e-01 | 5 |
| j2ee | 2100 | 4642 | 0 | 3 | 1.1e-01 | 2 | 1.5e-01 | 10 |
| uml | 3462 | 13130 | 0 | 2 | 1.6e-01 | 2 | 2.1e-01 | 5 |
| enterprise | 13548 | 15604 | – | 5 | 6.5e-01 | 2 | 7.3e-01 | 10 |
| netbeans | 44581 | 189646 | – | 2 | 4.9e+00 | 2 | 4.3e+00 | 5 |

has self-edges, i.e., there may be edges connecting a node to itself. The network is quite large, with 44581 nodes and 189646 links.

Following [8, 9], we symmetrize the adjacency matrix by considering each edge without orientation. This allows us to test the performance of the algorithms discussed in this paper, and to perform an initial analysis of the network. The sparsity pattern of the original Netbeans network and of the symmetrized version is displayed in Figure 6.6.

Tables 6.7 and 6.8 present results for several subnetworks of Netbeans as well as for the entire network. Each subnetwork corresponds to a subsystem of the whole software package. Our task was to determine the five most important nodes of the entire network as well as of each subnetwork. It is quite remarkable that only 2 to 5 iterations with Algorithms 1 and 2 suffice to satisfy the strong convergence criterion and only 2 iterations are needed to satisfy the weak convergence criterion. It is also interesting that Algorithms 1 and 2 require about the same number of iterations $N$ for each subnetwork as well as for the entire network. As explained above, we did not apply the `expm` and `eig` functions to matrices of order larger than $10^4$.

Using available information about bugs in the CUs and the computed values for the $f$-subgraph centrality and starting convenience gives Table 6.9. The entries of

TABLE 6.8

*Comparison of Gauss quadrature and hybrid algorithms. For each matrix, we report the number of matrix-vector product evaluations (mvp) and the execution time in seconds.*

| | | Gauss | | Hybrid | | expm | eig |
|---|---|---|---|---|---|---|---|
| Network | Nodes | mvp | Time | mvp | Time | Time | Time |
| profiler | 1231 | 10265 | 3.0e+00 | 429 | 1.1e-01 | 1.3e+01 | 3.7e+00 |
| j2ee | 2100 | 15654 | 4.9e+00 | 386 | 1.1e-01 | 2.7e+01 | 1.6e+01 |
| uml | 3462 | 33541 | 1.7e+01 | 655 | 2.7e-01 | 2.8e+02 | 8.1e+01 |
| enterprise | 13548 | 55136 | 4.1e+01 | 640 | 8.3e-01 | – | – |
| netbeans | 44581 | 450078 | 1.3e+03 | 679 | 3.7e+00 | – | – |

TABLE 6.9

*Contingency table reporting the frequency of bugs with respects to either subgraph centrality or starting convenience.*

| | Subgraph centrality | |
|---|---|---|
| Bugs | Smaller than average | Larger than average |
| no | 33563 | 4501 |
| yes | 4004 | 2513 |
| | Starting convenience | |
| Bugs | Smaller than average | Larger than average |
| no | 30028 | 8035 |
| yes | 3053 | 3465 |

the top table represent the number of CUs without and with bugs, with subgraph centrality either smaller or larger than the average. The bottom table reports similar information for the starting convenience. The tables yield the chi-square test statistic $\chi^2 = 2997$ and $\chi^2 = 3503$, respectively. Both values are much larger than the critical value $\chi^2_\alpha = 7.88$ from the chi-square distribution for $\alpha = 5 \cdot 10^{-3}$. Since we are testing the hypothesis of stochastic independence, we are led to accept, with probability 99.5%, the hypothesis that the presence of bugs in a CU depends on both the subgraph centrality and the starting convenience. The results of Table 6.9 suggest that there is a low probability of finding bugs in CUs characterized by a low value of the computed metrics. This example illustrates that the hybrid method can be applied to large-scale problems. Both the storage and execution time of the method are fairly small.

**7. Conclusion and extensions.** The numerical examples illustrate the competitiveness of the hybrid method of this paper when applied to the analysis of large networks. Hybrid methods may be the only feasible approach for determining the most important node(s) of a large undirected graph. Of course, the feasibility of the hybrid method depends on the distribution of the eigenvalues of the adjacency matrix. Numerical experiments, some of which are reported in section 6, indicate that for many networks of interest as well as for synthetic networks, the adjacency matrix has sufficiently few large eigenvalues to make the hybrid method of the present paper competitive.

This paper considers undirected graphs. The symmetry of the adjacency matrix for this kind of graph leads to some simplifications of the computations. However, there are many networks that give rise to directed graphs; see [5, 8, 10, 11, 14, 15, 16, 27] for discussions and illustrations. Hybrid methods also can be developed for directed graphs. We will discuss such methods in forthcoming work.

**Acknowledgments.** The authors would like to thank Michele Marchesi and Roberto Tonelli for providing us the data for the last numerical experiment. Com-

ments by the referees lead to improvements of the presentation. LR would like to thank Giuseppe Rodriguez and Sebastiano Seatzu for an enjoyable visit to the University of Cagliari, during which this work was initiated.

## REFERENCES

[1] J. Baglama, D. Calvetti, and L. Reichel, *IRBL: An implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems*, SIAM J. Sci. Comput., 24 (2003), pp. 1650–1677.

[2] J. Baglama, D. Calvetti, and L. Reichel, *Algorithm* 827*: irbleigs: A MATLAB program for computing a few eigenpairs of a large sparse Hermitian matrix*, ACM Trans. Math. Software, 29 (2003), pp. 337–348.

[3] V. Batagelj and A. Mrvar, *Pajek datasets*, http://vlado.fmf.uni-lj.si/pub/networks/data/ (2006).

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

[5] M. Benzi, E. Estrada, and C. Klymko, *Ranking hubs and authorities using matrix functions*, Linear Algebra Appl., 431 (2013), pp. 2447–2474.

[6] D. A. Bini, G. M. Del Corso, and F. Romani, *Evaluating scientific products by means of citation-based models: A first analysis and validation*, Electron. Trans. Numer. Anal., 33 (2008), pp. 1–16.

[7] D. Calvetti, G. H. Golub, and L. Reichel, *Estimation of the L-curve via Lanczos bidiagonalization*, BIT, 39 (1999), pp. 603–619.

[8] G. Concas, M. Marchesi, A. Murgia, and R. Tonelli, *An empirical study of social networks metrics in object oriented software*, Adv. Software Eng., 2010 (2010), 729826.

[9] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, *On the distribution of bugs in the Eclipse system*, IEEE Trans. Software Eng., 37 (2011), pp. 872–877.

[10] J. J. Croft, E. Estrada, D. J. Higham, and A. Taylor, *Mapping directed networks*, Electron. Trans. Numer. Anal., 37 (2010), pp. 337–350.

[11] E. Estrada, *The Structure of Complex Networks*, Oxford University Press, Oxford, 2012.

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

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

[14] E. Estrada and J. A. Rodríguez-Velázquez, *Subgraph centrality in complex networks*, Phys. Rev. E, 71 (2005), 056103.

[15] E. Estrada and J. A. Rodríguez-Velázquez, *Subgraph centrality and clustering in complex hyper-networks*, Phys. A, 364 (2006), pp. 581–594.

[16] C. Fenu, D. Martin, L. Reichel, and G. Rodriguez, *Block Gauss and anti-Gauss quadrature with application to networks*, SIAM J. Matrix Anal. Appl., to appear.

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

[18] G. H. Golub, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.

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

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

[21] G. H. Golub and Z. Strakoš, *Estimates in quadratic formulas*, Numer. Algorithms, 8 (1994), pp. 241–268.

[22] H. Jeong, S. Mason, A.-L. Barabási, and Z. N. Oltvai, *Lethality and centrality of protein networks*, Nature, 411 (2001), pp. 41–42.

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

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

[25] Max Plank Institute for Software Systems, *Online Social Networks Research*, http://socialnetworks.mpi-sws.org/data-wosn2009.html.

[26] M. E. J. Newman, *The structure of scientific collaboration networks*, Proc. Natl. Acad. Sci. USA, 98 (2001), pp. 404–409.
[27] M. E. J. Newman, *Networks: An Introduction*, Oxford University Press, Oxford, 2010.
[28] M. E. J. Newman, *Network Data*, http://www-personal.umich.edu/~mejn/netdata/.
[29] M. Penrose, *Geometric Random Graphs*, Oxford University Press, Oxford, 2003.
[30] S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen, *Topological structure analysis of the protein-protein interaction network in budding yeast*, Nucleic Acids Res., 31 (2003), pp. 2443–2450.
[31] A. Taylor and D. J. Higham, *CONTEST: A controllable test toolbox for MATLAB*, ACM Trans. Math. Software, 35 (2009), pp. 1–17.
[32] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, *On the evolution of user interaction in Facebook*, Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09), Barcelona, Spain, 2009, ACM, New York, 2009, pp. 37–42.
[33] D. J. Watts and S. H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442.