



A Fast Monte Carlo Algorithm for Evaluating Matrix Functions with Application in Complex Networks

Nicolas L. Guidotti¹ · Juan A. Acebrón^{1,2} · José Monteiro¹

Received: 28 July 2023 / Revised: 23 February 2024 / Accepted: 25 February 2024 /
Published online: 1 April 2024
© The Author(s) 2024

Abstract

We propose a novel stochastic algorithm that randomly samples entire rows and columns of the matrix as a way to approximate an arbitrary matrix function using the power series expansion. This contrasts with existing Monte Carlo methods, which only work with one entry at a time, resulting in a significantly better convergence rate than the original approach. To assess the applicability of our method, we compute the subgraph centrality and total communicability of several large networks. In all benchmarks analyzed so far, the performance of our method was significantly superior to the competition, being able to scale up to 64 CPU cores with remarkable efficiency.

Keywords Monte Carlo methods · Randomized algorithms · Matrix functions · Network analysis

Mathematics Subject Classification 65C05 · 68W20 · 65F50 · 05C90 · 68W10

1 Introduction

Complex networks have become an essential tool in many scientific areas [36, 69, 70] for studying the interaction between different elements within a system. From these networks, it is possible, for instance, to identify the most important elements in the system, such as the key proteins in a protein interaction network [35], the keystone species in an ecological network [59], vulnerable infrastructures [4] or the least resilient nodes in a transportation network [9].

✉ Nicolas L. Guidotti
nicolas.guidotti@tecnico.ulisboa.pt

Juan A. Acebrón
juan.acebron@ist.utl.pt

José Monteiro
jcm@inesc-id.pt

¹ INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

² Department of Mathematics, Carlos III University of Madrid, Leganés, Spain

Measures of node importance are referred to as *node centrality* and many metrics have been proposed over the years [36, 70]. Here, we consider the family of centrality measures defined in terms of matrix functions [38], which classify the nodes according to how well they can spread information to other nodes in the network. Both the Katz centrality [60] and subgraph centrality [39] belong to this family. In most cases, the node centrality is computed based on the matrix resolvent $(\mathbf{I} - \gamma\mathbf{A})^{-1}$ and the exponential $\exp(\mathbf{A})$, but other functions [11, 17] can be used as well. Here $\mathbf{A} \in \mathbb{R}^{n \times n}$ denotes the network's adjacency matrix. In this paper, we focus on the subgraph centrality [39] (i.e., $\text{diag}(f(\mathbf{A}))$) and total communicability [15] (i.e., $f(\mathbf{A})\mathbf{1}$), where $f(\mathbf{A})$ is a given matrix function.

Although these centrality measures have been successfully used in many problems [17, 38, 39, 41], computing a matrix function for large networks can be very demanding using the current numerical methods. Direct methods, such as `expm` [5, 53] or the Schur-Parlett algorithm [26], have a computational cost of $O(n^3)$ and yield a full dense matrix $f(\mathbf{A})$, hence are only feasible for small matrices. Methods based on Gaussian quadrature rules [14, 42, 46] can estimate the diagonal entries of $f(\mathbf{A})$ without evaluating the full matrix function, but are prone to numerical breakdown when \mathbf{A} is sparse and large (which is often the case with real networks), and thus, are often employed to determine only the most important nodes in the network. Krylov-based methods [3, 33, 51, 52] can efficiently compute $f(\mathbf{A})\mathbf{v}$, for $\mathbf{v} \in \mathbb{R}^n$, provided that \mathbf{A} is well conditioned. Otherwise, their convergence rate can be very slow or even stagnate since a general and well-established procedure to precondition the matrix \mathbf{A} does not exist. Rational Krylov methods are often more resilient to the condition number and provide a better approximation to $f(\mathbf{A})\mathbf{v}$ than polynomial ones, but require solving a linear system for each vector of the basis. Moreover, the stopping criteria for these methods remains an open issue [52].

Monte Carlo methods [18, 28, 29, 43, 58] provide an alternative way to calculate matrix functions, primarily for solving linear systems. In essence, these methods construct a discrete-time Markov chain whose underlying random paths evolve through the different indices of the matrix, which can be formally understood as the Monte Carlo sampling of the corresponding Neumann series. Their convergence has been rigorously established in [18, 30, 58]. Recently, [1, 2, 48] extended these methods for the evaluation of the matrix exponential and Mittag-Leffler functions.

Another strategy is to construct a random *sketch* (i.e., a probabilistic representation of the matrix) and then use it to approximate the desired operation. This is a basic idea in contemporary numerical linear algebra [64, 66]. Some recent studies have shown that a polynomial Krylov method can be accelerated using randomization techniques [25, 49]. In this paper, we propose a new stochastic algorithm that randomly samples full rows and columns of the matrix as a way to approximate the target function using the corresponding power series expansion. Through an extensive set of numerical experiments, we show that our approach converges much faster than the original Monte Carlo method and that it is particularly effective for estimating the subgraph centrality and total communicability of large networks. We also compare our method against other classical and randomized methods considering very large matrices.

The paper is organized as follows. Section 2 presents a brief overview of the centrality measures defined in terms of matrix functions. Section 3 describes our randomized algorithm and how it can be implemented efficiently. In Sect. 4, we evaluate the performance and accuracy of our method by running several benchmarks with both synthetic and real networks. We also compare our method against several other algorithms. In the last section, we conclude our paper and present some future work.

2 Background

In this section, we introduce some definitions and ideas from graph theory that will be used throughout the paper.

2.1 Graph Definitions

A graph or network $G = (V, E)$ is composed of a set $V = \{1, 2, \dots, n\}$ of nodes (or vertices) and a set $E = \{(u, v) : u, v \in V\}$ of edges between them [70]. A graph is *undirected* if the edges are bidirectional and *directed* if the edges are unidirectional. A *walk* of length k over the graph G is a sequence of vertices v_1, v_2, \dots, v_{k+1} such that $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \dots, k$. A *closed walk* is a walk that starts and ends at the same vertex, i.e., $v_1 = v_{k+1}$. An edge from a node to itself is called a *loop*. A *subgraph* of graph G is a graph created from a subset of nodes and edges of G . The *degree* of a node is defined as the number of edges entering or exiting the node. In directed graphs, the *in-degree* counts the number of incoming edges and *out-degree*, the number of outgoing edges.

A graph G can be represented through an *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$:

$$\mathbf{A} = (a_{ij}); \quad a_{ij} = \begin{cases} w_{ij}, & \text{if edge } (i, j) \text{ exists in graph } G \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where w_{ij} is the weight of the edge (i, j) . In this paper, we focus our attention on graphs that are undirected and *unweighted* (i.e., $w_{ij} = 1$ for all edges (i, j)) and do not contain loops or have multiple edges between nodes. Consequently, all matrices in this paper will be symmetric, binary, and with zeros along the diagonal. Notwithstanding, it is worth mentioning that our method is general and can be applied to other classes of matrices.

2.2 Centrality Measures

There are many *node centrality* measures, and the simplest one must be the *degree centrality* [44]. The degree of a node provides a rough estimation of its importance, yet it fails to take into consideration the connectivity of the immediate neighbours with the rest of the network. Instead, let us consider the flow of information in the network. An important node must be part of routes where the information can flow through, and thus, be able to spread information very quickly to the rest of the network. These information routes are represented as walks over the network. This is the main idea behind walk-based centralities and was formalized by Estrada and Higham [38].

In graph theory, it is well known that the entry $(\mathbf{A}^k)_{ij}$ counts the number of walks of length $k \geq 1$ over graph G that starts at node i and end at node j . Then, the entry f_{ij} of the matrix function $f(\mathbf{A})$ defined as

$$f(\mathbf{A}) = \sum_{k=0}^{\infty} \zeta_k \mathbf{A}^k \quad (2)$$

measures how easily the information can travel from node i to node j . The entry \mathbf{A}^k is scaled by a coefficient ζ_k , such that $\zeta_k \geq \zeta_{k+1} \geq 0$ and $\zeta_k \rightarrow 0$ when k is large, in order to penalize the contribution of longer walks and ensure the convergence of the series. The two most common choices for $f(\mathbf{A})$ are the matrix exponential $e^{\mathbf{A}}$ and resolvent $(I - \gamma \mathbf{A})^{-1}$ [38], but other matrix functions can be used as well [10, 11, 17].

Algorithm 1 A Monte Carlo method adapted from [18] for computing a matrix \mathbf{F} as an approximation of $f(\mathbf{A})$. N_s represents the number of random walks for approximating each row and W_c is the weight cutoff.

```

1: function MC(A, Ns, Wc)
2:   F = 0
3:   T = { tij =  $\frac{|a_{ij}|}{\sum_k |a_{ik}|}$  }
4:   for i = 1, 2, ..., n do ▷ for each row in A
5:     for s = 1, 2, ..., Ns do ▷ for each random walk
6:       ℓ0 = i; W(0) =  $\frac{1}{N_s}$ ; k = 0
7:       while W(k) > WcW(0) do ▷ compute the k-th step
8:         fi ℓk = fi ℓk + ζkW(k)
9:         ℓk+1 = SELECTNEXTSTATE(T, ℓk)
10:        W(k+1) = W(k)  $\frac{a_{\ell_k \ell_{k+1}}$ 
11:           $t_{\ell_k \ell_{k+1}}$ 
12:        k = k + 1
13:       end while
14:     end for
15:   return F
16: end function

```

From the power series (2), Estrada defined *f-subgraph centrality* [38, 39] as the diagonal of $f(\mathbf{A})$, that is $fSC(i) = (f(\mathbf{A}))_{ii}$, and measures the importance of this node based on its participation in all subgraphs in the network. The sum over all nodes of the subgraph centrality has become known as the *Estrada Index* [27, 34], which was first introduced to quantify the degree of folding in protein chains [34], but later extended to characterize the global structure of general complex networks [37, 40].

Later, Benzi and Klymko [15, 17] introduced the concept of *f-total communicability* based on the row sum of $f(\mathbf{A})$, ranking the nodes according to how well they can communicate with the rest of the network. Formally, the *f-total communicability* is expressed as

$$fTC(i) = (f(\mathbf{A})\mathbf{1})_i, \tag{3}$$

where $\mathbf{1}$ is a vector of length n with all entries set to 1. If we consider the matrix resolvent $f(\mathbf{A}) = (I - \gamma\mathbf{A})^{-1}$, the total communicability of a node corresponds to the well-known Katz's Centrality [55, 60].

In the context of network science, it is common to introduce a weighting parameter $\gamma \in (0, 1)$ and work with the parametric matrix function $f(\gamma\mathbf{A})$. The parameter γ can be interpreted as the *inverse temperature* and accounts for external disturbances on the network [41]. Furthermore, the value of γ is often chosen in such a way that the terms $\zeta_k(\gamma\mathbf{A})^k$ in (2) are monotonically decreasing in order to preserve the notion that the information travels faster to nearby nodes compared to those that are farther away.

3 Randomized Algorithm for Matrix Functions

Ulam and von Neumann [43] were the first to propose a Monte Carlo method for computing the matrix inverse as a way to solve linear systems, which was later refined by [18, 28, 30]. It consists of generating random walks over the matrix \mathbf{A} to approximate each power in the corresponding series (2). Starting from a row ℓ_0 , a random walk consists of a random variable

$W^{(k)}$ and a sequence of states $\ell_0, \ell_1, \dots, \ell_k$, which are obtained by randomly jumping from one row to the next. At each step k , the program updates $W^{(k)}$ and add the results to entry $f(\mathbf{A})_{i\ell_k}$ as an approximation for the k -th term in the series (2).

The full procedure is described in Algorithm 1. The `SelectNextState` routine randomly selects an entry in row ℓ_k to determine which row to jump to in the next step of the random walk. The probability of choosing an entry j is equal to $\mathbb{P}(\ell_{k+1} = j \mid \ell_k = i) = t_{ij}$, where t_{ij} is an entry of a transition probability matrix \mathbf{T} .

The main limitation of this method is that each random walk only updates a single entry of $f(\mathbf{A})$ at a time, requiring a large number of walks just to estimate a single row with reasonable accuracy. Therefore, our objective is to modify this algorithm such that it samples entire rows and columns of \mathbf{A} when approximating each term in the series (2), drastically reducing the number of walks necessary to achieve the desired precision.

3.1 Mathematical Description of the Method

In the following, we discuss how to extend the randomized matrix product algorithm proposed in [31, 32] to compute an arbitrary matrix power.

Lemma 1 *Let R_i and C_j denote the i -th row and j -th column of $\mathbf{A} \in \mathbb{R}^{n \times n}$. The matrix power \mathbf{A}^p with $p \in \mathbb{N}$ and $p \geq 2$ can be evaluated as*

$$\mathbf{A}^p = \sum_{i_2=1}^n \sum_{i_3=1}^n \cdots \sum_{i_p=1}^n C_{i_2} a_{i_2 i_3} a_{i_3 i_4} \cdots a_{i_{p-1} i_p} R_{i_p}. \tag{4}$$

Proof Recall that the matrix product \mathbf{AB} with $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$ can be expressed as a sum of outer products [32]:

$$\mathbf{AB} = \sum_{k=1}^n C_k B_k,$$

where C_k is the k -th column of \mathbf{A} and B_k is the k -th row of \mathbf{B} . Therefore, a power p of a square matrix \mathbf{A} can be written as

$$\mathbf{A}^p = \sum_{k=1}^n C_k R_k^{(p-1)}, \tag{5}$$

where $R_k^{(p-1)}$ is the k -th row of \mathbf{A}^{p-1} . For a single row, (5) is reduced to

$$R_i^{(p)} = \sum_{k=1}^n a_{ik} R_k^{(p-1)}. \tag{6}$$

Recursively applying (6) for the powers $p, p-1, \dots, 2$ and then substituting the expansion in (5) leads to the expression in (4). □

Corollary 1 *Let $f(\mathbf{A}) : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n}$ be a matrix function defined by the power series*

$$f(\mathbf{A}) = \zeta_0 \mathbf{I} + \zeta_1 \mathbf{A} + \sum_{k=2}^{\infty} \zeta_k \mathbf{A}^k = \mathbf{H} + \mathbf{U}, \tag{7}$$

where $\mathbf{H} = \zeta_0 \mathbf{I} + \zeta_1 \mathbf{A}$. Concerning matrix \mathbf{U} , it can be written as the following sum of rank-one matrices:

$$\mathbf{U} = \sum_{k=2}^{\infty} \sum_{i_2=1}^n \sum_{i_3=1}^n \cdots \sum_{i_k=1}^n \zeta_k C_{i_2} a_{i_2 i_3} a_{i_3 i_4} \cdots a_{i_{k-1} i_k} R_{i_k}. \tag{8}$$

The multiple sums appearing in the definition of matrix \mathbf{U} in (8) can be exploited in practice to construct a probabilistic algorithm similar to [18, 28], which was originally created for computing the inverse matrix. In fact, the formal procedure is analogous, but instead of generating random scalar variables our goal here consists of generating randomly rank-one matrices governed by the following Markov chain.

Definition 1 Let $\{X_k : k \geq 0\}$ be a Markov chain taking values in the state space $S = \{1, 2, \dots, n\}$ with initial distribution and transition probability matrix given by

$$(i) \quad \mathbb{P}(X_0 = \ell_0) = p_{\ell_0} = \frac{\|C_{\ell_0}\|_2}{\sum_{k=1}^n \|C_k\|_2} \tag{9}$$

$$(ii) \quad \mathbf{T} = (t_{ij}); \quad t_{ij} = \mathbb{P}(\ell_{k+1} = j \mid \ell_k = i) = \frac{|a_{ij}|}{\sum_{k=1}^n |a_{ik}|}. \tag{10}$$

Here, the indices ℓ_m denote the corresponding state reached by the Markov chain after m -steps. We use the initial state ℓ_0 of the Markov chain to choose randomly a column of matrix \mathbf{A} , C_{ℓ_0} . After k -steps of the Markov chain, the state is ℓ_k , which is used to select the corresponding row of the matrix \mathbf{A} , R_{ℓ_k} . During this random evolution of the Markov chain different states are visited according to the corresponding transition probability, and along this process a suitable multiplicative random variable $W^{(k)}$ is updated conveniently. Finally, matrix \mathbf{U} can be computed through the expected value of a given functional, as proved formally by the following Lemma.

Lemma 2 Let $\mathbf{Z}(\omega)$ be a realization of a random matrix at a point ω of the discrete sample space, defined as

$$\mathbf{Z}(\omega) = \sum_{k=0}^{\infty} \zeta_{k+2} C_{X_0(\omega)} W^{(k)} R_{X_k(\omega)}. \tag{11}$$

Here $W^{(k)}$ is a multiplicative random variable defined recursively as

$$W^{(0)} = \frac{1}{p_{\ell_0}} \quad W^{(k)} = W^{(k-1)} \frac{a_{\ell_{k-1} \ell_k}}{t_{\ell_{k-1} \ell_k}}. \tag{12}$$

Then, it holds that

$$\mathbf{U} = \mathbb{E}[\mathbf{Z}]. \tag{13}$$

Proof Note that $\mathbf{Z}(\omega)$ is obtained from Eq. (11) as a sum of independent random matrices $\mathbf{Y}^{(k)}(\omega)$:

$$\mathbf{Z}(\omega) = \sum_{k=0}^{\infty} \zeta_{k+2} \mathbf{Y}^{(k)}(\omega), \tag{14}$$

where

$$\mathbf{Y}^{(k)}(\omega) = C_{X_0(\omega)} W^{(k)} R_{X_k(\omega)} = C_{X_0(\omega)} \frac{a_{\ell_0 \ell_1} a_{\ell_1 \ell_2} \cdots a_{\ell_{k-1} \ell_k}}{p_{\ell_0} t_{\ell_0 \ell_1} t_{\ell_1 \ell_2} \cdots t_{\ell_{k-1} \ell_k}} R_{X_k(\omega)}.$$

Let $P^{(k)}(\omega)$ be the probability of occurring an event ω consisting in a transition from ℓ_0 to ℓ_k after k steps. This probability turns out to be $p_{\ell_0\ell_0}t_{\ell_0\ell_1}t_{\ell_1\ell_2}\dots t_{\ell_{k-1}\ell_k}$. Therefore, the expected value of the random matrix $\mathbf{Y}^{(k)}(\omega)$ is given by,

$$\begin{aligned} \mathbb{E}[\mathbf{Y}^{(k)}] &= \sum_{\omega} P^{(k)}(\omega)\mathbf{Y}^{(k)}(\omega) \\ &= \sum_{\omega} C_{X_0(\omega)}a_{\ell_0\ell_1}a_{\ell_1\ell_2}\dots a_{\ell_{k-1}\ell_k}R_{X_k(\omega)}. \end{aligned}$$

Note that every event ω can be described by different values of $k + 1$ integer indices, running from 1 to n , then

$$\mathbb{E}[\mathbf{Y}^{(k)}] = \sum_{i_0=1}^n \sum_{i_1=1}^n \dots \sum_{i_k=1}^n C_{i_0}a_{i_0i_1}a_{i_1i_2}\dots a_{i_{k-1}i_k}R_{i_k}.$$

Therefore, from (8) we conclude that $\mathbb{E}[\mathbf{Y}^{(k)}] = \mathbf{A}^{k+2}$. Finally after summing all contributions coming from any number of steps, using (14) and by linearity of the expected value operator we obtain

$$\mathbb{E}[\mathbf{Z}] = \sum_{k=0}^{\infty} \zeta_{k+2}\mathbb{E}[\mathbf{Y}^{(k)}] = \sum_{k=0}^{\infty} \zeta_{k+2}\mathbf{A}^{k+2} = \mathbf{U}. \tag{15}$$

□

3.2 Practical Implementation of the Probabilistic Method

To transform Lemma 2 into a practical algorithm, we must first select a finite sample size N_s and then compute the expected value $\mathbb{E}[\mathbf{Z}]$ in (13) as the corresponding arithmetic mean. Additionally, each random walk must terminate after a finite number m of steps. Mathematically, this is equivalent to considering only the first m terms of the power series expansion. Since some random walks may retain important information of the matrix for longer steps than others and it is very difficult to determine a priori the number of steps required for achieving a specific precision, we adopt the following termination criteria: the computation of the random walk will end when the associated weight is less than a relative threshold W_c [18]. In other words, a random walk terminates at step m when

$$W^{(m)} \leq W_c W^{(0)}, \tag{16}$$

where $W^{(m)}$ is the weight after m steps and $W^{(0)}$ is the weight at the initial step of the random walk. Formally, the infinite series in (15) is truncated as

$$f(\mathbf{A}) \approx \mathbf{H} + \hat{\mathbf{U}} \tag{17}$$

with

$$\hat{\mathbf{U}} = \frac{1}{N_s} \sum_{s=1}^{N_s} \sum_{k=0}^m \zeta_{k+2} C_{X_0(s)} \Omega_{\ell_0\ell_k}^{(k)} R_{X_k(s)} \tag{18}$$

where $\Omega_{ij}^{(k)}$ corresponds to the weight $W^{(k)}$ of the random walk that began at state i and arrived at state j after k steps. Here s indicates a realization of a random walk.

Algorithm 2 A probabilistic algorithm for computing the matrix \mathbf{F} as an approximation of $f(\mathbf{A})$. N_s represents the total number of random walks and W_c the weight cutoff.

```

1: function RANDFUNM( $\mathbf{A}$ ,  $N_s$ ,  $W_c$ )
2:    $\mathbf{Q} = \mathbf{0}$ 
3:    $\mathbf{T} = \left\{ t_{ij} = \frac{|a_{ij}|}{\sum_k |a_{ik}|} \right\}$ 
4:   for  $i = 1, 2, \dots, n$  do ▷ for each column in  $\mathbf{A}$ 
5:      $N_i = N_s \mathbb{P}(\ell_0 = i)$  ▷ see Eq. (9)
6:     for  $s = 1, 2, \dots, N_i$  do ▷ for each random walk
7:        $\ell_0 = i$ ;  $W^{(0)} = \frac{1}{N_i}$ ;  $k = 0$ 
8:       while  $W^{(k)} > W_c W^{(0)}$  do ▷ compute the  $k$ -th step
9:          $q_i \ell_k = q_i \ell_k + \zeta_{k+2} W^{(k)}$ 
10:         $\ell_{k+1} = \text{SELECTNEXTSTATE}(\mathbf{T}, \ell_k)$ 
11:         $W^{(k+1)} = W^{(k)} \frac{a_{\ell_k \ell_{k+1}}}{t_{\ell_k \ell_{k+1}}}$ 
12:         $k = k + 1$ 
13:      end while
14:    end for
15:  end for
16:   $\mathbf{F} = \zeta_0 \mathbf{I} + \zeta_1 \mathbf{A} + \mathbf{AQA}$ 
17:  return  $\mathbf{F}$ 
18: end function

```

Computing $\hat{\mathbf{U}}$ directly from (18) is ill-advised due to the large number of outer products, while also being very difficult to be parallelised efficiently. Instead, let us rearrange (17) to a more suitable form. The random walks with the same starting column can be grouped as

$$\hat{\mathbf{U}} = \sum_{i=1}^n C_i \left(\frac{1}{N_i} \sum_{s=1}^{N_i} \sum_{k=0}^m \zeta_{k+2} \Omega_i^{(k)} R_{X_k(s)} \right), \tag{19}$$

where N_i is the number of random walks that began at column i . Assuming that $N_s \gg 1$, the value of N_i can be estimated a priori as $N_i \approx p_i N_s$ with $p_i = \mathbb{P}(X_0 = i)$ as defined in (9).

Let v_{ij} denote a visit to the state j at the step k of a random walk that started at state i . For each visit v_{ij} , the weight of the walk is added to the entry q_{ij} of matrix \mathbf{Q} , defined as

$$q_{ij} = \frac{1}{N_i} \sum_{v_{ij}} \zeta_{k+2} \Omega_{ij}^{(k)}. \tag{20}$$

Then, we can rewrite (18) as

$$\hat{\mathbf{U}} = \sum_{i=1}^n \sum_{j=1}^n C_i q_{ij} R_j = \mathbf{AQA}. \tag{21}$$

Algorithm 2 describes the procedure for approximating $f(\mathbf{A})$ based on Eqs. (17) and (21). Assuming that matrix \mathbf{A} is sparse with N_{nz} nonzero entries, Algorithm 2 requires $O(N_s m)$ operations to construct the matrix \mathbf{Q} and $O(n N_{nz})$ to calculate the final product \mathbf{AQA} , for a total computational cost of order of $O(N_s m + n N_{nz})$. It also uses an additional n^2 space in memory to store the matrix \mathbf{Q} . It is possible to reduce memory consumption if the program divides rows of \mathbf{Q} into blocks in such a way that only one block is computed at a time. At the

end of each block, the program updates the matrix $f(\mathbf{A})$ and reuses the memory allocation for the next block.

3.3 Diagonal of the Matrix Function

Algorithm 2 can be conveniently modified to compute only the diagonal of the matrix function. Let Q_k denote the k -th row of matrix \mathbf{Q} defined in (20). Then, the diagonal of $f(\mathbf{A})$ is approximated by vector $\mathbf{d} = (d_i)$ as follows

$$d_i = \mathbf{e}_i^\top f(\mathbf{A}) \mathbf{e}_i = \zeta_0 + \zeta_1 a_{ii} + \sum_{k=1}^n a_{ik} \langle Q_k, C_i \rangle, \tag{22}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product and e_i a vector from the canonical basis. Essentially, the program updates the value of d_i immediately after the computation of row Q_k . In this way, only a single row of \mathbf{Q} needs to reside in memory at a time. Naturally, if $a_{ik} \sim 0$, the program can skip the calculation of Q_k in order to save computational resources. Note that multiple entries can be computed at the same time by reusing Q_k and then selecting the appropriate entry a_{ik} and column C_i .

In terms of computational cost, the computation of all diagonal entries requires $O(N_s m + N_{nz} \bar{n}_c)$ floating-point operations, where \bar{n}_c denotes the average number of nonzero entries per column and consumes an additional n space in memory. The algorithm described in this section will be referred to as `RandFunmDiag`.

3.4 Action of the Matrix Function Over a Vector

Let \mathbf{v} be a real vector in \mathbb{R}^n , our goal is to develop another algorithm based on Lemma 2 for computing $f(\mathbf{A}) \mathbf{v}$. First, let us multiply the truncated series (17) by the vector \mathbf{v} :

$$f(\mathbf{A})\mathbf{v} \approx \mathbf{h} + \hat{\mathbf{u}} \tag{23}$$

where

$$\hat{\mathbf{u}} = \frac{1}{N_s} \sum_{s=1}^{N_s} \sum_{k=0}^m \zeta_{k+2} C_{X_0(s)} \Omega_{\ell_0 \ell_k}^{(k)} r_{X_k(s)} \tag{24}$$

with $r_i = \langle R_i, \mathbf{v} \rangle$ and $\mathbf{h} = \mathbf{H}\mathbf{v}$. Rearranging the series such that the random walks with the same starting column are grouped:

$$\hat{\mathbf{u}} = \sum_{i=1}^n C_i \left(\frac{1}{N_i} \sum_{s=1}^{N_i} \sum_{k=0}^m \zeta_{k+2} \Omega_{i \ell_k}^{(k)} r_{X_k(s)} \right).$$

Then, the action of the matrix function $f(\mathbf{A})$ over the vector \mathbf{v} can be approximated as

$$f(\mathbf{A})\mathbf{v} \approx \mathbf{h} + \mathbf{A}\mathbf{q} \tag{25}$$

with

$$\mathbf{q} = (q_i); \quad \mathbf{q}_i = \frac{1}{N_i} \sum_{s=1}^{N_i} \sum_{k=0}^m \zeta_{k+2} \Omega_{i \ell_k}^{(k)} r_{X_k(s)}.$$

Algorithm 3 A probabilistic algorithm for computing \mathbf{y} as an approximation of $f(\mathbf{A})\mathbf{v}$. N_s represents the total number of random walks and W_c the weight cutoff threshold.

```

1: function RANDFUNMACTION( $\mathbf{A}$ ,  $\mathbf{v}$ ,  $N_s$ ,  $W_c$ )
2:    $\mathbf{q} = \mathbf{0}$ 
3:    $\mathbf{T} = \left\{ t_{ij} = \frac{|a_{ij}|}{\sum_k |a_{ik}|} \right\}$ 
4:    $\mathbf{r} = \mathbf{A}\mathbf{v}$ 
5:   for  $i = 1, 2, \dots, n$  do ▷ for each column in  $\mathbf{A}$ 
6:      $N_i = N_s \mathbb{P}(\ell_0 = i)$  ▷ see Eq. (9)
7:     for  $s = 1, 2, \dots, N_i$  do ▷ for each random walk
8:        $\ell_0 = i$ ;  $W^{(0)} = \frac{1}{N_i}$ ;  $k = 0$ 
9:       while  $W^{(k)} > W_c W^{(0)}$  do ▷ compute the  $k$ -th step
10:         $q_i = q_i + \zeta_{k+2} W^{(k)} r_{\ell_k}$ 
11:         $\ell_{k+1} = \text{SELECTNEXTSTATE}(\mathbf{T}, \ell_k)$ 
12:         $W^{(k+1)} = W^{(k)} \frac{a_{\ell_k \ell_{k+1}}}{t_{\ell_k \ell_{k+1}}}$ 
13:         $k = k + 1$ 
14:       end while
15:     end for
16:   end for
17:    $\mathbf{y} = \zeta_0 \mathbf{v} + \zeta_1 \mathbf{r} + \mathbf{A}\mathbf{q}$ 
18:   return  $\mathbf{y}$ 
19: end function

```

Algorithm 3 describes the procedure for approximating $f(\mathbf{A})\mathbf{v}$ based on Eq. (25) and the definition of the vector \mathbf{q} . It has a time complexity of $O(N_s m + N_{nz})$ and requires an additional n space in memory to store the vector \mathbf{q} .

3.5 Convergence of the Method and Numerical Errors

In the following, we prove the convergence of the Monte Carlo method described by Eqs. (17) and (21) through the following theorem:

Theorem 1 *Let m be any positive integer. For each $k \in \{0, \dots, m\}$, let $(\xi^{(k)}(s))_{s \geq 1}$ be a collection of i.i.d. vector-valued random variable in \mathbb{R}^{n^2} defined as $\xi^{(k)}(s) = \zeta_{k+2} \Omega_{\ell_0 \ell_k}^{(k)} \text{vec}(C_{X_0(s)} R_{X_k(s)})$, and $V(s) = \sum_{k=0}^m \xi^{(k)}(s)$. Let $\psi = \sum_{s=0}^{N_s} V(s)$, $\mu = \mathbb{E}[V(s)]$, $\alpha = \max_i \{(\sum_{j=1}^n |a_{ij}|)^2\} < 1$, and $|\zeta_{k+1}| < |\zeta_k|$ when $k \rightarrow \infty$. Then*

$$\lim_{N_s \rightarrow \infty} \frac{\psi - N_s \mu}{\sqrt{N_s}} \tag{26}$$

converges in distribution to a random vector distributed according to a multivariate normal distribution $N[0, \Theta]$. Here $\Theta = (\theta_{ij})$ is the covariance matrix, with $\theta_{ij} = \text{Cov}(v_i(s), v_j(s))$, and $v_i(s)$ the i -th component of the random vector $V(s)$.

Proof This proof is based on the proof described in Theorem 3.4 [58] conveniently modified for the current numerical method. Assuming that all random walks are independently generated, then

$$\text{Var}(v_i(s)) = \sum_{k=0}^m \text{Var}(\xi_i^{(k)}(s)) \leq \sum_{k=0}^m \mathbb{E}[(\xi_i^{(k)}(s))^2] \leq \sum_{k=0}^{\infty} \mathbb{E}[(\xi_i^{(k)}(s))^2]. \tag{27}$$

Here, $\xi_i^{(k)}(s)$ denotes the i -th component of the random vector $\xi^{(k)}(s)$. To compute the expected value $\mathbb{E}[(\xi_i^{(k)}(s))^2]$, we have to enumerate all the different transitions that occurred between a given initial state ℓ_0 and an arbitrary final state ℓ_k of the Markov chain in k steps, along with the corresponding probabilities. This yields,

$$\mathbb{E}[(\xi_i^{(k)}(s))^2] = \zeta_{k+2}^2 \sum_{i_1=1}^n \cdots \sum_{i_k=1}^n t_{\ell_0 i_1} t_{i_1 i_2} \cdots t_{i_{k-1} i_k} \frac{1}{p_{\ell_0}^2} (\Omega_{\ell_0 i_k}^{(k)})^2 (g_i^{(i_k)})^2 \tag{28}$$

where $\mathbf{g}^{(i_k)} = \text{vec}(C_{\ell_0} R_{i_k})$ is the vector obtained after vectorizing the matrix $C_{\ell_0} R_{i_k}$ with $g_i^{(i_k)}$ as the i -th component. From Eq. (12), it follows that

$$\begin{aligned} \mathbb{E}[(\xi_i^{(k)}(s))^2] &= \zeta_{k+2}^2 \sum_{i_1=1}^n \sum_{i_2=1}^n \cdots \sum_{i_k=1}^n \frac{1}{p_{\ell_0}^2} \frac{(a_{\ell_0 i_1} a_{i_1 i_2} \cdots a_{i_{k-1} i_k})^2}{t_{\ell_0 i_1} t_{i_1 i_2} \cdots t_{i_{k-1} i_k}} (g_i^{(i_k)})^2 \\ &= \frac{\zeta_{k+2}^2}{p_{\ell_0}^2} \sum_{i_1=1}^n \frac{a_{\ell_0 i_1}^2}{t_{\ell_0 i_1}} \sum_{i_2=1}^n \frac{a_{i_1 i_2}^2}{t_{i_1 i_2}} \cdots \sum_{i_k=1}^n \frac{a_{i_{k-1} i_k}^2}{t_{i_{k-1} i_k}} (g_i^{(i_k)})^2. \end{aligned} \tag{29}$$

Note that $\sum_{j=1}^n \frac{a_{ij}^2}{i_j} = (\sum_{j=1}^n |a_{ij}|)^2$ from Eq. (10), then it holds

$$\mathbb{E}[(\xi_i^{(k)}(s))^2] \leq \frac{\zeta_{k+2}^2}{p_{\ell_0}^2} \alpha^k \beta. \tag{30}$$

Here $\beta = \max_j \{(g_i^{(j)})^2\}$. Since $\zeta_{k+2} \leq 1, \forall k$ for any matrix function of interest, from Eq. (27), we have

$$\text{Var}(v_i(s)) \leq \sum_{k=0}^{\infty} \mathbb{E}[(\xi_i^{(k)}(s))^2] \leq \frac{\beta}{p_{\ell_0}^2} \sum_{k=0}^{\infty} |\zeta_{k+2}| \alpha^k \leq \frac{\beta}{\alpha^2 p_{\ell_0}^2} \sum_{j=0}^{\infty} |\zeta_j| \alpha^j. \tag{31}$$

Since $\lim_{k \rightarrow \infty} \alpha \frac{|\zeta_{k+1}|}{|\zeta_k|} < 1$, it holds that the series $\sum_{j=0}^{\infty} |\zeta_j| \alpha^j$ converges and therefore the variance $\text{Var}(v_i(s))$ is bounded.

Note, however, that for the specific case of the exponential function, $|\zeta_{k+1}|/|\zeta_k| = 1/(k+1)$, and therefore any value of α is allowed to ensure convergence of the series. However, for the inverse function $\alpha < 1$ is strictly mandatory.

Since the variance is finite, the Central Limit Theorem for vector-valued random variables (see [56] e.g.) guarantees that

$$\frac{\psi - N_s \mu}{\sqrt{N_s}} \rightarrow^d N[0, \Theta]. \tag{32}$$

□

Therefore, for a finite sample size N_s , replacing the expected value in (13) by the arithmetic mean introduces an error which is statistical in nature and known to be distributed according to a normal distribution. From Theorem 1 the standard error of this mean ε can be readily estimated as $\sigma^2/\sqrt{N_s}$, being σ the corresponding standard deviation.

Table 1 Numerical examples for evaluating the algorithms. Here, $k = 10^3$, $M = 10^6$ and $B = 10^9$

Name	Digraph?	Nodes	Edges	Description
smallworld-<n>	No	2^n	10×2^n	Random graph based on the Watts–Strogatz model [75]. Each edge has a 10% rewiring chance
kronecker-<n>	No	$\sim 2^n$	$\sim 16 \times 2^n$	Kronecker graph used by the Graph500 benchmark [47, 63]
yeast	No	2114	4480	Protein interaction network for yeast [24, 57, 73]
power-us	No	4941	13k	Topological representation of the power grid of the western states in the US [68, 75]
internet	No	23k	96k	Symmetrized snapshot of the structure of the internet at the level of autonomous systems circa 2006 [68]
cond-mat	No	40k	351k	Collaboration network of scientists in the field of condensed matter from 1995 to 2005 [68, 69]
twitch	No	168k	6.8M	Social network of Twitch users in Spring 2018 [61, 72]
stanford	Yes	281k	2.3M	Web graph of the Stanford University domain in 2002 [61, 62]
orkut	No	3.1M	117M	Social network of Orkut users in 2007 [61, 65]
uk-2005	Yes	39.5M	936M	Web graph of the .uk domain in 2005 [19–21]
twitter	Yes	42.6M	1.47B	Social network of Twitter users in 2009 [61, 76]

4 Numerical Examples

To illustrate the applicability of our method, we compute the subgraph centrality and total communicability of several complex networks using the matrix exponential $e^{\gamma \mathbf{A}}$ with $\gamma \in [0, 1]$. Due to the random nature of the Monte Carlo algorithms, all results reported in this section correspond to the mean value of 10 runs of the algorithm using different random seeds.

All numerical simulations were executed on a commodity server with an AMD EPYC 9554P 64C 3.75 GHz and 256 GB of RAM, running Fedora 38. All randomized algorithms were implemented in C++ using OpenMP. The code was compiled with AMD AOCC v4.0 with the `-O3` and `-march=znver4` flags. Our implementation uses the `PCG64DXSM` [71] random number generator.

The algorithms were tested using two synthetic graphs—`smallworld` and `kronecker`—as well as a set of networks extracted from real-world data, which are described in Table 1. Note that, before calculating the subgraph centrality and total communicability of directed graphs, their adjacency matrix \mathbf{A} must be symmetrized as

$$\mathbf{B} = \begin{bmatrix} 0 & \mathbf{A} \\ \mathbf{A}^\top & 0 \end{bmatrix} \quad (33)$$

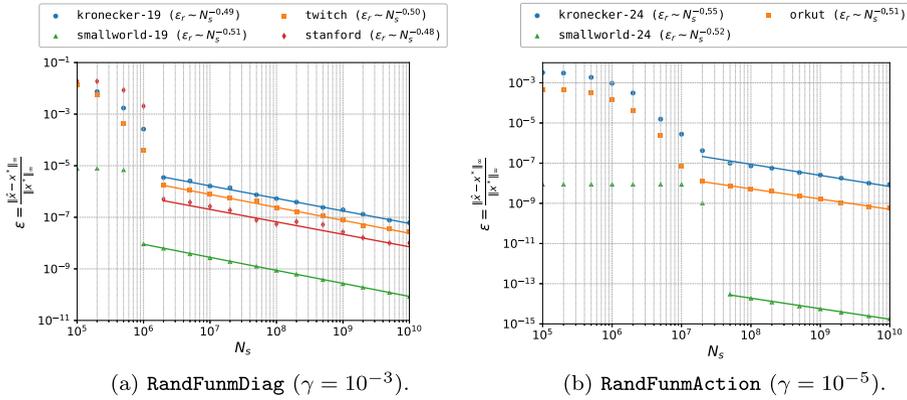


Fig. 1 Relative ℓ_∞ error as function of the number of random walks N_s for $W_c = 10^{-6}$. The degree γ of the polynomial of the fitting curve is indicated as $\varepsilon_r \sim N_s^\gamma$

in order to split apart the outgoing and incoming edges of the graph [17]. We also remove all duplicated edges, loops and disconnected nodes from the `kronecker` graph generated by the `Graph500` code [47].

4.1 Numerical Errors and Accuracy

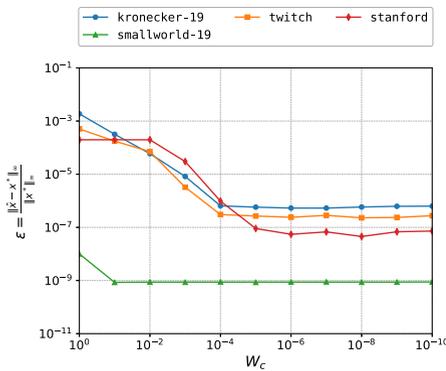
Figure 1 show the relative ℓ_∞ error of our method as function of N_s . Recall from Sect. 3.2 that the algorithm assigns a priori $N_j \sim \|C_j\|_2$ random walks to each node j of the graph. Therefore, if N_s is too low, very few random walks will be assigned to a node i with a low norm, such that its centrality score is basically approximated by just the first two terms in the series, i.e., $SC(i) \approx 1 + a_{ii}$ and $TC(i) \approx 1 + (A\mathbf{1})_i$. For this reason, the relative errors are highly dependent on the structure of the graph. In particular, the nodes in the `smallworld` network have very similar probabilities, and therefore it can happen that for $N_s < n$ some nodes will have no chance to be randomly chosen. Only when the sample size is sufficiently large, the algorithm can properly estimate the centrality of every node. In this scenario, the numerical error of the algorithm scales with $O(N_s^{-0.5})$, similar to other probabilistic methods. This relation is confirmed numerically by the trend lines in Fig. 1 which has a slope of approximately -0.5 in the logarithmic scale. Table 2 shows the relative ℓ_∞ error for a fixed number of samples. The table also shows the standard measurement error obtained after several independent runs.

In Fig. 2 we show the relative ℓ_∞ error as function of W_c . The value of W_c controls the length of the random walks in terms of the number of steps, which is related to the number of terms of the power series expansion. When the value of W_c is large, the algorithm stops the random walk generation too early, leading to large errors. On the other hand, when the value of W_c is small, the algorithm continues generating the random walk for longer steps than necessary. Note that this increases the computational cost without necessarily improving the accuracy of the method. In fact, we have to consider also the statistical error, which depends on the number of generated random walks. According to Fig. 2 the optimal value for W_c is around 10^{-6} for these networks.

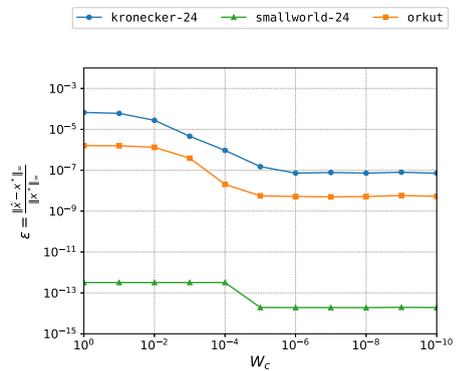
Figure 3 shows how the relative ℓ_∞ error grows with the size of the `smallworld` and `kronecker` networks. The `smallworld` network starts as a ring lattice with n nodes,

Table 2 Relative ℓ_∞ error for $N_s = 10^8$ and $W_c = 10^{-6}$ as well as the standard measurement error obtained after several independent runs

(a) RandFunmDiag ($\gamma = 10^{-3}$)		ε_r
kronecker-19		$(1.94 \pm 0.14) \times 10^{-7}$
twitch		$(8.09 \pm 0.67) \times 10^{-8}$
smallworld-19		$(2.70 \pm 0.04) \times 10^{-10}$
stanford		$(2.70 \pm 0.45) \times 10^{-8}$
(b) RandFunmAction ($\gamma = 10^{-5}$)		ε_r
kronecker-24		$(2.57 \pm 0.26) \times 10^{-8}$
orkut		$(1.67 \pm 0.09) \times 10^{-9}$
smallworld-24		$(5.59 \pm 0.16) \times 10^{-15}$

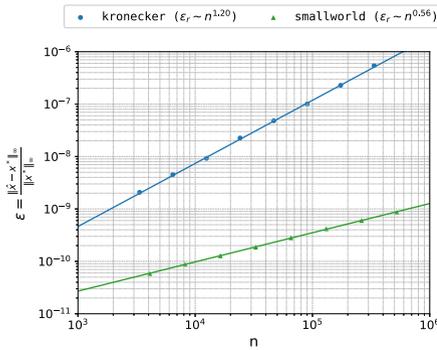


(a) RandFunmDiag ($\gamma = 10^{-3}$).

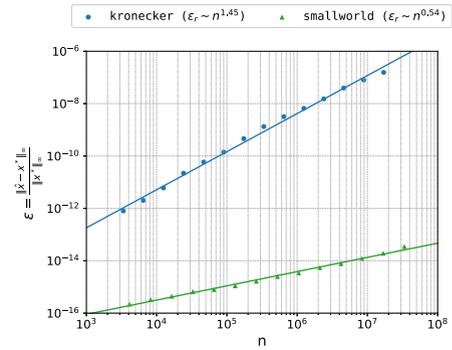


(b) RandFunmAction ($\gamma = 10^{-5}$).

Fig. 2 Relative ℓ_∞ error as function of the weight cutoff W_c for $N_s = 10^8$



(a) RandFunmDiag ($\gamma = 10^{-3}$).



(b) RandFunmAction ($\gamma = 10^{-5}$).

Fig. 3 Relative ℓ_∞ error as function of the number of nodes n of the graph considering $W_c = 10^{-6}$ and $N_s = 10^8$. The degree γ of the polynomial of the fitting curve is indicated as $\varepsilon_r \sim n^\gamma$

each connected to 10 neighbours. The algorithm then rewires each edge in the lattice with a 10% chance, i.e., the edge (i, j) is replaced by (i, k) where k is chosen at random from all the possible nodes in the network such that there are no loops or duplicated edges. As a result, the random walks have very similar weights $W^{(k)}$ independently of the sequence of nodes visited. In other words, the norm of the covariance matrix $\sigma^2 = \|\Theta\|_\infty$ is very low. Considering that N_s is fixed, there are fewer random walks to estimate the centrality score of each node as the graph size increases, which reduces the precision of the algorithm by a factor of \sqrt{n} , as shown in Fig. 3b, c. This is in line with the theoretical results from Sect. 3.5, where $\varepsilon_r \sim \sigma^2 N_s^{-0.5}$.

In contrast, the nodes in the kronecker graph are organized hierarchically. At the top level, there is a single node acting as the central hub for the entire graph. As we move down the hierarchy, there are more nodes per level, but they have fewer connections. The number of levels in the hierarchy as well as the number of nodes and connections at each level are determined by the size of the graph. Therefore, the weight of the random walks can vary greatly depending on which nodes are visited and their position in the hierarchy. Larger graphs have a higher covariance norm σ^2 due to a wider degree difference between nodes.

4.2 Comparison with Other Methods

There are a few algorithms available in the literature for computing the matrix exponential. Perhaps the most well-known scheme is the `expm` routine from MATLAB [5, 53, 54]. The method first scales the matrix \mathbf{A} by a power of 2 to reduce the norm to order 1, calculates the Padé approximant of the matrix exponential and then repeatedly squares the result to recover the original exponent. For a generic $n \times n$ matrix, `expm` requires $O(n^3)$ arithmetic operations and an additional $O(n^2)$ space in memory. `expm` calculate the entire matrix $e^{\gamma \mathbf{A}}$.

To rank the nodes using the subgraph centrality, we only need to calculate the diagonal entries of $e^{\gamma \mathbf{A}}$, not the complete matrix. Methods for estimating the individual entries of the matrix function have been proposed by Golub, Meurant and others [14, 42, 46] and are based on Gaussian quadrature rules and the Lanczos algorithm. They require $O(n)$ operations to determine each diagonal entry, resulting in a total cost of $O(n^2)$ to calculate the subgraph centrality for all nodes. In practice, this method may suffer from numerical breakdowns when \mathbf{A} is large and sparse [12, 42]. For this reason, it is often restricted to estimate only the k most important nodes in the graph.

Likewise, the total communicability only requires the action of $f(\mathbf{A})$ over a vector setting $\mathbf{v} = \mathbf{1}$, which can be computed efficiently using either a polynomial or rational Krylov method [3, 33, 51, 52]. These methods consist in generating a Krylov basis using the input matrix and then evaluating the function over the projected matrix through some direct method, such as `expm`. Assuming a sparse matrix with N_{nz} nonzeros and a Krylov basis with m vectors, the computational cost is $O(mN_{nz})$. In particular, we compared our method against the restarted polynomial Krylov [3, 33] from the `funm_kry1` toolbox [50].

While writing this article, Güttel and Schweitzer published a paper [49] proposing two new randomized algorithms—`sFOM` and `sGMRES`—for estimating $f(\mathbf{A})\mathbf{v}$. Here, we focus on `sFOM` since `sGMRES` works best with Stieltjes functions and requires a numerical quadrature rule for all the other functions. `sFOM` first creates a random *sketch* of the \mathbf{A} and then uses an incomplete Arnoldi decomposition to generate a non-orthogonal Krylov basis from this *sketch*. However, the basis may be ill-conditioned, and thus, for stabilizing the method, it is required to compute a thin QR decomposition (also called *whitening* [67]) of the basis before

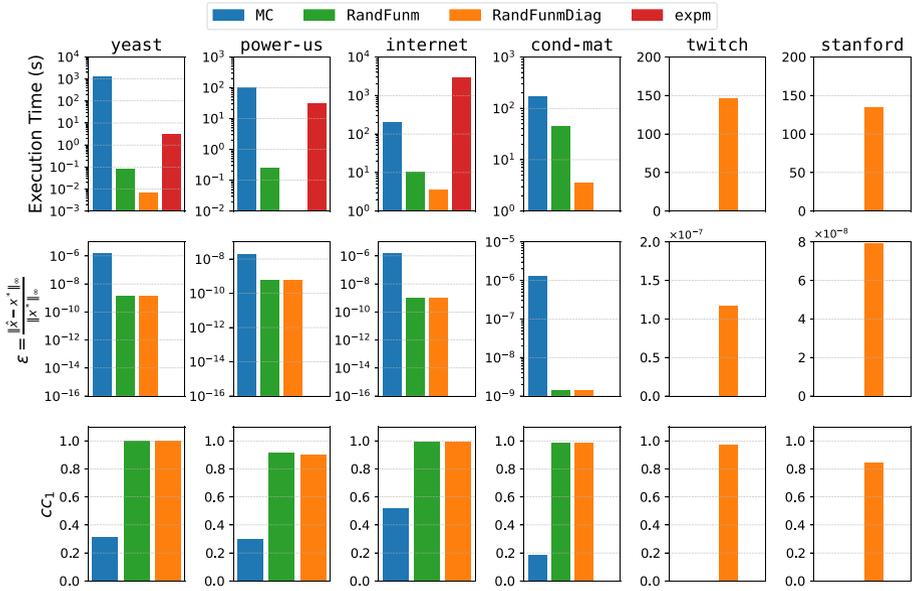


Fig. 4 Comparison between different algorithms when calculating the subgraph centrality for real networks and $\gamma = 10^{-3}$. Here, we consider the centrality scores generated by our algorithm with $N_s = 10^{11}$ as reference

evaluating the matrix function over the projected matrix. The computational cost of sFOM is $O(N_{nz}m \log m + m^3)$.

Another preprint by Cortinovis, Kressner and Nakatsukasa [25] was also published recently proposing a different randomization strategy for the Krylov method. They propose an Arnoldi-like decomposition to iteratively build the non-orthogonal Krylov basis \mathbf{V}_m using only random sketches of the basis vectors. Again, the method may apply a whitening [67] to improve the condition number of the basis. Afterwards, the program solves a least-square problem to obtain the projected matrix. We will denote this algorithm as `rand_kry1` and it has an overall computational cost of $O(N_{nz}m^2 + m^3)$.

The MC represents the Monte Carlo method adapted from [18] as described in Algorithm 1. Similar to our randomized algorithm, the length of the random walks depends on the weight cutoff W_c . It has a computational cost of $O(mN_s)$, where m denotes the average number of steps in the random walk, and does not require additional space in memory. This method can be modified to calculate $f(\mathbf{A})\mathbf{v}$ or $f(\mathbf{A})_{ii}$ instead of the full matrix function. Note that for the latter, the method still computes the full $f(\mathbf{A})$, but discards all the off-diagonal entries.

Figure 4 compares the serial execution time and accuracy among the different methods when computing the subgraph centrality. The graphs are sorted according to the number of nodes. The similarities between the two node rankings are measured using the Pearson correlation coefficient [13]. Here, cc_1 denotes the correlation coefficient between the top 1% nodes between a reference list and the ranking obtained by the algorithm. Note that, if two or more nodes have similar centrality scores, numerical deviations can alter their ranking order, lowering the correlation coefficient. Nevertheless, these nodes have similar importance within the network, and thus, their order in the ranking may not be relevant to understanding the dynamics of the graph.

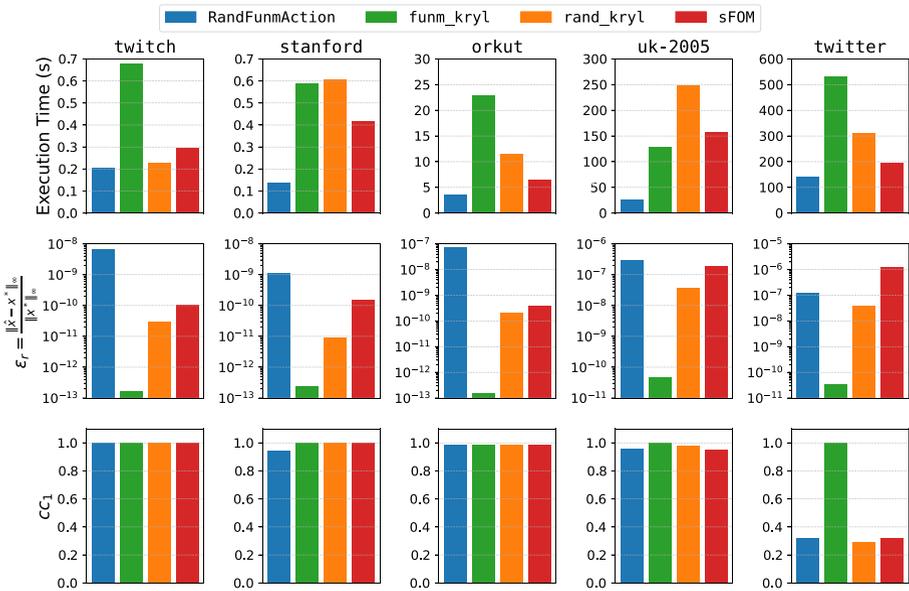


Fig. 5 Comparison between different algorithms when computing the total communicability for real networks and $\gamma = 10^{-5}$. Here, we consider the centrality scores generated by `expmv` [6] as reference

Although `expm` reaches machine precision, it cannot be used for large graphs due to its hefty computational cost and cubic scaling. In fact, the `cond-mat` graph with 40k nodes is already too large for `expm` and cannot be executed in a reasonable amount of time. The MC requires a very large number of random walks to estimate the subgraph centrality as it only updates a single entry of the matrix exponential at a time, and it is more likely for this entry to be outside the diagonal if the matrix is very large. For this reason, the accuracy of MC is quite poor even with a large number of random walks. Both `RandFunm` and `RandFunmDiag` have the same accuracy and correlation since the core algorithm is the same. However, `RandFunmDiag` does not require the computation of the full matrix product at the end of the algorithm, resulting in a speedup between 3 to 24 over `RandFunm`. Moreover, the full matrix exponential of the `twitch` and `stanford` graphs are too large to be fully represented in memory, and thus, their subgraph centrality can only be calculated by `RandFunmDiag`. The randomized algorithms also show a very high correlation for top-1% nodes in the ranking compared with the reference list.

Figure 5 compares `RandFunmAction` against all Krylov-based methods in terms of the serial execution time and accuracy. The tolerance of `funm_kryl` was set to 10^{-8} and the size of the Krylov basis of `sFOM` and `rand_kryl` was set to 4, such that all algorithms have comparable precision. We choose a small value of γ to avoid overflow as we are working with large positive matrices, and consequently, all algorithms converge very fast to target precision.

The stopping criterion of `funm_kryl` is well-known to be pessimistic [52], resulting in much higher precision than the target at the cost of higher execution times. In some networks (`twitch`, `orkut` and `twitter`), `sFOM` and `rand_kryl` outperformed `funm_kryl` mainly due to the smaller basis, while in others (`uk-2005`), the additional cost associated with the *sketching*, *whitening*, least square QR and other operations lead to significant slow-

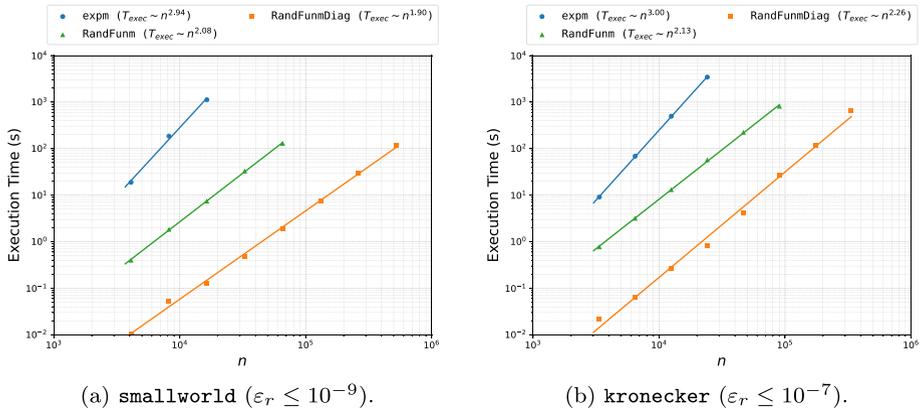


Fig. 6 Elapsed time for computing the subgraph centrality as a function of the number of nodes n for a fixed accuracy ϵ_r and $\gamma = 10^{-3}$

downs. Considering the accuracy difference, the randomization in the Krylov method does not seem to be very effective when the basis is relatively small. RandFunmAction shows the best performance among all algorithms, in particular, for the twitter network, being $3.8\times$ faster than funm_kryl, while sFOM and rand_kryl are $2.7\times$ and $1.7\times$ faster, respectively.

For complex networks, it suffices for the algorithm to be sufficiently accurate to differentiate the centrality score between all nodes in the graph, there is no benefit in having higher accuracy. Indeed, the ranking produced by RandFunmAction has a correlation greater than 0.95 for the top 1% nodes despite having a lower accuracy than the others. The only exception is the twitter network. As a massive social network, there is no clear structure or hierarchy in the graph, such that many of them have similar centrality scores and small numerical variations can drastically change the rank order. In comparison, uk-2005 is an equally large web graph that follows a more clear structure with a well-defined hub and authorities, and thus, it is less susceptible to noise. For this reason, the correlation for the twitter network is much lower than other graphs and requires greater precision to differentiate the nodes. Note that the top-1% in the ranking contains almost 1 million nodes in both the twitter and uk-2005 networks with a wide range of centrality scores. If we consider only the top-0.1%, the correlation of RandFunmAction for the twitter network increases to 0.79.

Figures 6 and 7 show the elapsed serial time as a function of the number of nodes for the kronecker and smallworld networks. The computational cost of the expm algorithm is of the order of $O(n^3)$. This is regardless of the sparsity or the distribution of the nonzeros of the matrix since it was originally proposed for dense matrices. The MC algorithm was too slow for the prescribed accuracy, and thus, it was not included in the graph. When $N_{nz} \sim n$ the computational cost of the RandFunm and RandFunmDiag algorithms become of order $O(n^2)$. This is because the computational cost of computing the matrix product is higher than generating random walks. Note that this cost is similar to other algorithms [42] for estimating diagonal entries of the matrix. However, the main advantage here is the smaller proportionality constant and the capability to compute the subgraph centrality for sparse and large matrices without worrying about a numerical breakdown [42]. Again, the RandFunmDiag algorithm is significantly faster than the RandFunm algorithm as it only requires the partial evaluation

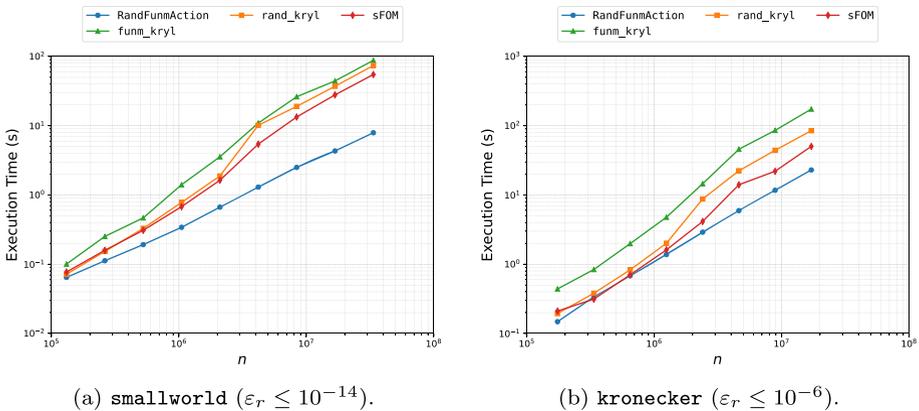


Fig. 7 Elapsed time for computing the total communicability as a function of the number of nodes n for a fixed accuracy ϵ_r and $\gamma = 10^{-5}$

Table 3 Relative error for calculating the total communicability of a single node $i = \arg \max_j \sum_k |a_{jk}|$ considering $N_s = 10^8$, $W_c = 10^{-6}$ and $\gamma = 10^{-5}$

	stanford	orkut	kronecker-24
MC	1.48×10^{-9}	6.59×10^{-8}	1.33×10^{-6}
RandFunmAction	4.64×10^{-11}	3.87×10^{-10}	2.01×10^{-8}

of the matrix product at the end of the algorithm. All Krylov-based methods scale linearly with n as they rely on matrix–vector multiplications. Similar to the other Monte Carlo methods, RandFunmAction spends more time computing the matrix–vector product than generating the random walks, and thus, also scales linearly with n .

4.3 Single Entry

One of the main advantages of Monte Carlo algorithms is the ability to calculate a single entry of the solution without requiring the computation of the full solution. Table 3 shows the relative ℓ_∞ error for calculating the total communicability of the node with the highest degree. Both RandFunmAction and MC were modified to calculate a single entry of the solution as efficiently as possible. Due to the ability to sample entire rows and columns, RandFunmAction produces a much better approximation for $(f(\mathbf{A})v)_i$ than MC for the same number of random walks, independently of the network.

4.4 Parallel Performance

Parallelizing our randomized algorithm is fairly straightforward. Multiple rows of \mathbf{Q} can be computed at the same time in the RandFunmDiag algorithm, yet the diagonal entries must be updated atomically to avoid data races. Likewise, the RandFunmAction algorithm can compute the vector \mathbf{q} as well as the final product $\mathbf{A}\mathbf{q}$ completely in parallel.

Fig. 8 Strong scaling for the RandFunmDiag algorithm considering $W_c = 10^{-8}$ and $\gamma = 10^{-3}$

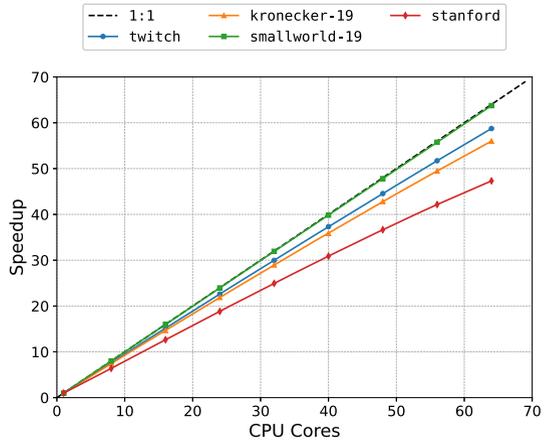
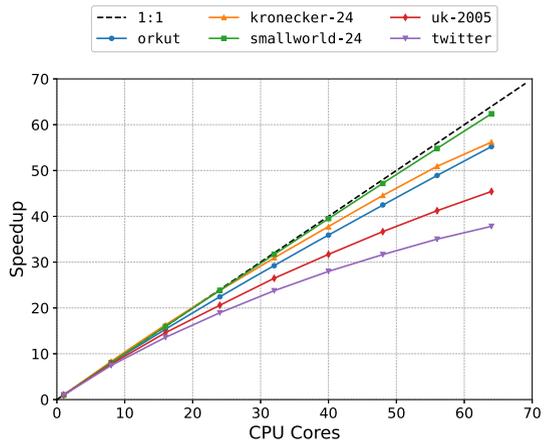


Fig. 9 Strong scaling for the RandFunmAction algorithm considering $W_c = 10^{-8}$ and $\gamma = 10^{-5}$



Figures 8 and 9 show the strong scaling for the parallel implementation of the Rand FunmDiag and RandFunmAction algorithms, respectively. The scalability of both algorithms is excellent, attaining more than 85% in efficiency for most networks when using 64 cores. In particular, the parallel code was able to achieve near-perfect scaling for the smallworld network due to its low degree per node and an almost uniform structure. This leads to a more efficient usage of the cache as well as an even distribution of load across the processors.

In most networks, the random walks are not distributed equally across the nodes, such that some rows of \mathbf{Q} and entries of \mathbf{q} take longer to compute than others. To solve this load imbalance, the program dynamically distributes the vector \mathbf{q} and matrix \mathbf{Q} over the CPU cores. This solution was very effective for most networks, improving significantly the performance of the program. Yet, the CPU may still be underutilized at the end of the code if the graph is very unbalanced. This is the case of directed graphs due to the symmetrization of the adjacent matrix as shown in (33).

Another limiting factor is the latency and bandwidth of the main memory. Most operations with large and sparse matrices are well-known to be memory-bound as they cannot utilize the cache hierarchy effectively while requiring additional logic and memory accesses for

handling the sparse storage format. In fact, the `kry1_funm` algorithm shows no benefits when running in a multithreaded environment since it relies on sparse matrix–vector products and the majority of the code is written in MATLAB. In contrast, our randomized algorithm only needs to compute two sparse matrix products: one at the beginning and another at the end of the algorithm. This still affects the scalability of the method when working with massive networks, such as `twitter` and `uk-2005`. Even under these conditions, the program was able to obtain significant speedups when using 64 cores, achieving 60% efficiency for `twitter` and 70% for `uk-2005`.

4.5 Katz Centrality

One of the most well-known centrality measures based on matrix functions is Katz's Centrality (KC) [55, 60]. It is defined as $(\mathbf{I} - \gamma\mathbf{A})\mathbf{x} = \mathbf{1}$ with $KC(i) = x_i$ as the centrality score of the node i . Here, the γ is called attenuation factor and should be between 0 and $\rho(\mathbf{A})^{-1}$, where $\rho(\mathbf{A})$ is the spectral radius of \mathbf{A} [60]. Different information can be extracted from the network by changing the value of γ [16]. For instance, if γ tends to $\rho(\mathbf{A})^{-1}$, Katz's Centrality approximates the eigenvalue centrality [22, 23]. If γ tends to 0, then it converges to the degree centrality.

There are several ways to solve the linear system $(\mathbf{I} - \gamma\mathbf{A})\mathbf{x} = \mathbf{1}$. Direct solvers, such as MUMPS [7, 8] or Intel Pardiso [74], first compute the LU factorization or similar and then solve the linear system using backward/forward substitution. However, factorization is a very costly procedure, scaling with $O(n^3)$, while also requiring additional space to store matrices \mathbf{L} and \mathbf{U} . On the other hand, sparse iterative solvers, such as Conjugate Gradient, GMRES, and BiCGSTAB, can converge very quickly to the solution especially provided a good preconditioning is available. Last, but not least, Monte Carlo methods solve the linear system as the truncated series

$$(\mathbf{I} - \gamma\mathbf{A})^{-1}\mathbf{v} = \sum_{k=0}^m (\gamma\mathbf{A})^k \mathbf{v}$$

with $\mathbf{v} = \mathbf{1}$. However, this series only converges when $\rho(\gamma\mathbf{A}) < 1$. Moreover, if $\rho(\gamma\mathbf{A})$ is near 1, the convergence rate will be very slow, and thus, requires computing many terms of the expansion to reach a reasonable accuracy.

Figure 10 compares `RandFunmAction` against the original Monte Carlo method (MC) and a simple Conjugate Gradient algorithm (CG). Here, the error ε of the CG is equal to residual norm $\|\mathbf{1} - \mathbf{A}\hat{\mathbf{x}}\|_2$, while for `RandFunmAction` and MC, it corresponds to ℓ_∞ error using the results from the CG as reference. Note that we avoid the costly computation of the eigenvalue by leveraging the Gershgorin's Theorem [45], i.e., $\rho(\gamma\mathbf{A}) \approx \max_i \sum_k |\gamma a_{ik}|$. Again, `RandFunmAction` is faster and more accurate than MC for the same number of random walks N_s , yet it still is not as good as the sparse iterative solver. Even without preconditioning, CG can converge extremely quickly to the solution due to the small value of γ . It is possible to enhance the performance of our methods by combining it with a Richardson iteration as shown in [18]. This is left for future work.

We want to emphasize that Monte Carlo methods are better suited to evaluate other matrix functions than the matrix inverse, whereas this is either too expensive or is not even possible with classical methods.

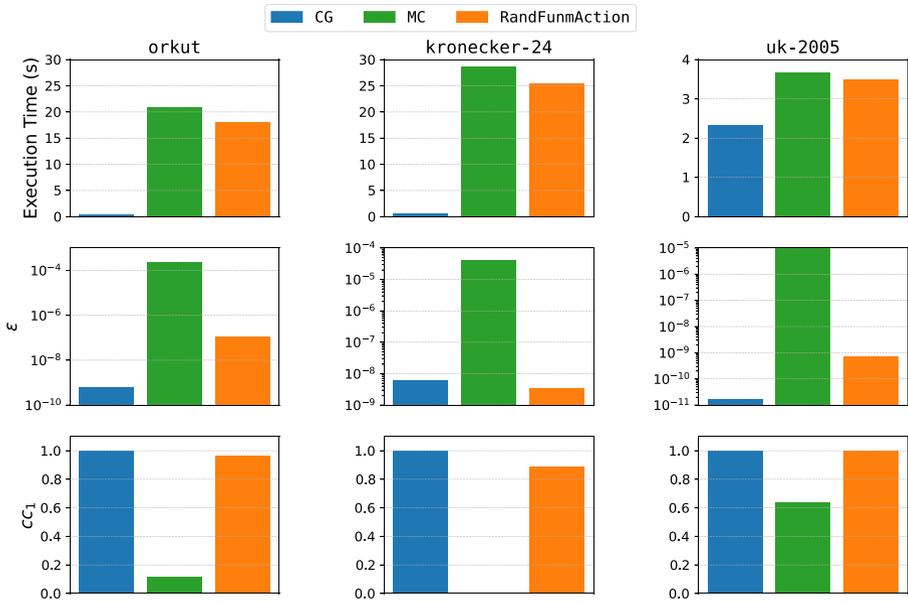


Fig. 10 Comparison among the different algorithms when computing the katz-centrality for $\gamma = 0.85 \max_j \sum_k |a_{jk}|$ and $N_s = 10^9$. The execution time is measured using all 64 cores

5 Conclusion

This paper proposes a novel stochastic algorithm that randomly samples rows and columns of the matrix for approximating different powers of the power series expansion. It can evaluate any matrix function by using the corresponding coefficients of the series. The algorithm can be conveniently modified to compute either $f(\mathbf{A})\mathbf{v}$ or the diagonal of $f(\mathbf{A})$ without the need to compute the entire matrix function. As a way to test the applicability of our method, we compute the subgraph centrality and total communicability of several large networks using the matrix exponential. Within this context, the stochastic algorithm has proven to be particularly effective, outperforming the competition. Our method also is highly scalable in a multithreaded environment, showing remarkable efficiency when using up to 64 cores.

In this paper, we primarily focus on the analysis of complex networks as it provided a very close relation with the method itself, but the algorithm can be applied to any scientific problem that can be expressed in terms of matrix functions, providing a quick way to estimate the solution of the problem with reasonable accuracy.

Funding Open access funding provided by FCT/IFCCN (b-on). This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects URA-HPC PTDC/08838/2022 and UIDB/50021/2020 (DOI:10.54499/UIDB/50021/2020) and grant 2022.11506.BD. JA was funded by Ministerio de Universidades and specifically the requalification program of the Spanish University System 2021–2023 at the Carlos III University.

Data availability The datasets generated during and/or analysed during the current study are available in the randfunn-networks repository, available at <https://gitlab.com/moccalib/applications/randfunn-networks>.

Declaration

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Acebrón, J.: A Monte Carlo method for computing the action of a matrix exponential on a vector. *Appl. Math. Comput.* **362**, 124545 (2019). <https://doi.org/10.1016/j.amc.2019.06.059>
2. Acebrón, J.A., Herrero, J.R., Monteiro, J.: A highly parallel algorithm for computing the action of a matrix exponential on a vector based on a multilevel Monte Carlo method. *Comput. Math. Appl.* **79**(12), 3495–3515 (2020). <https://doi.org/10.1016/j.camwa.2020.02.013>
3. Afanasjew, M., Eiermann, M., Ernst, O.G., Güttel, S.: Implementation of a restarted Krylov subspace method for the evaluation of matrix functions. *Linear Algebra Appl.* **429**(10), 2293–2314 (2008). <https://doi.org/10.1016/j.laa.2008.06.029>
4. Albert, R., Jeong, H., Barabási, A.L.: Error and attack tolerance of complex networks. *Nature* **406**(6794), 378–382 (2000). <https://doi.org/10.1038/35019019>
5. Al-Mohy, A.H., Higham, N.J.: A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.* **31**(3), 970–989 (2010). <https://doi.org/10.1137/09074721X>
6. Al-Mohy, A.H., Higham, N.J.: Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.* **33**(2), 488–511 (2011). <https://doi.org/10.1137/100788860>
7. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.* **23**(1), 15–41 (2001). <https://doi.org/10.1137/S0895479899358194>
8. Amestoy, P.R., Buttari, A., L'Excellent, J.Y., Mary, T.: Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Trans. Math. Softw.* **45**(1), 1–26 (2019). <https://doi.org/10.1145/3242094>
9. Aparicio, J.T., Arsenio, E., Santos, F.C., Henriques, R.: LINES: multimodal transportation resilience analysis. *Sustainability* **14**(13), 7891 (2022). <https://doi.org/10.3390/su14137891>
10. Arrigo, F., Benzi, M.: Edge modification criteria for enhancing the communicability of digraphs. *SIAM J. Matrix Anal. Appl.* **37**(1), 443–468 (2016). <https://doi.org/10.1137/15M1034131>
11. Arrigo, F., Durastante, F.: Mittag-Leffler functions and their applications in network science. *SIAM J. Matrix Anal. Appl.* **42**(4), 1581–1601 (2021). <https://doi.org/10.1137/21M1407276>
12. Bai, Z., Day, D., Ye, Q.: ABLE: an adaptive block Lanczos method for non-Hermitian eigenvalue problems. *SIAM J. Matrix Anal. Appl.* **20**(4), 1060–1082 (1999). <https://doi.org/10.1137/S0895479897317806>
13. Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson correlation coefficient. In: Cohen, I., Huang, Y., Chen, J., Benesty, J. (eds.) *Noise Reduction in Speech Processing*. Springer Topics in Signal Processing, pp. 1–4. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-00296-0_5
14. Benzi, M., Boito, P.: Quadrature rule-based bounds for functions of adjacency matrices. *Linear Algebra Appl.* **433**(3), 637–652 (2010). <https://doi.org/10.1016/j.laa.2010.03.035>
15. Benzi, M., Klymko, C.: Total communicability as a centrality measure. *J. Complex Netw.* **1**(2), 124–149 (2013). <https://doi.org/10.1093/comnet/cnt007>
16. Benzi, M., Klymko, C.: On the limiting behavior of parameter-dependent network centrality measures. *SIAM J. Matrix Anal. Appl.* **36**(2), 686–706 (2015). <https://doi.org/10.1137/130950550>
17. Benzi, M., Estrada, E., Klymko, C.: Ranking hubs and authorities using matrix functions. *Linear Algebra Appl.* **438**(5), 2447–2474 (2013). <https://doi.org/10.1016/j.laa.2012.10.022>

18. Benzi, M., Evans, T.M., Hamilton, S.P., Lupo Pasini, M., Slattery, S.R.: Analysis of Monte Carlo accelerated iterative methods for sparse linear systems. *Numer. Linear Algebra Appl.* (2017). <https://doi.org/10.1002/nla.2088>
19. Boldi, P., Vigna, S.: The WebGraph framework I: compression techniques. In: *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, pp. 595–601. ACM Press, Manhattan (2004)
20. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: UbiCrawler: a scalable fully distributed web crawler. *Softw. Pract. Exp.* **34**(8), 711–726 (2004)
21. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: *Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proceedings of the 20th International Conference on World Wide Web*, pp. 587–596. ACM Press, Hyderabad (2011)
22. Bonacich, P.: Factoring and weighting approaches to status scores and clique identification. *J Math Sociol* **2**(1), 113–120 (1972). <https://doi.org/10.1080/0022250X.1972.9989806>
23. Bonacich, P.: Power and centrality: a family of measures. *Am. J. Sociol.* **92**(5), 1170–1182 (1987). <https://doi.org/10.1086/228631>
24. Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., Chen, R.: Topological structure analysis of the protein–protein interaction network in budding yeast. *Nucleic Acids Res.* **31**(9), 2443–2450 (2003)
25. Cortinovis, A., Kressner, D., Nakatsukasa, Y.: Speeding up Krylov subspace methods for computing $f(A)b$ via randomization (2023)
26. Davies, P.L., Higham, N.J.: A Schur–Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.* **25**(2), 464–485 (2003). <https://doi.org/10.1137/S0895479802410815>
27. de la Peña, J.A., Gutman, I., Rada, J.: Estimating the Estrada index. *Linear Algebra Appl.* **427**(1), 70–76 (2007). <https://doi.org/10.1016/j.laa.2007.06.020>
28. Dimov, I.: *Monte Carlo Methods for Applied Scientists*. World Scientific, Singapore (2008)
29. Dimov, I., Alexandrov, V., Karaivanova, A.: Parallel resolvent Monte Carlo algorithms for linear algebra problems. *Math. Comput. Simul.* **55**(1–3), 25–35 (2001). [https://doi.org/10.1016/S0378-4754\(00\)00243-3](https://doi.org/10.1016/S0378-4754(00)00243-3)
30. Dimov, I., Maire, S., Sellier, J.M.: A new Walk on Equations Monte Carlo method for solving systems of linear algebraic equations. *Appl. Math. Model.* **39**(15), 4494–4510 (2015). <https://doi.org/10.1016/j.apm.2014.12.018>
31. Drineas, P., Kannan, R.: Fast Monte-Carlo algorithms for approximate matrix multiplication. In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 452–459 (2001). <https://doi.org/10.1109/SFCS.2001.959921>
32. Drineas, P., Kannan, R., Mahoney, M.W.: Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication. *SIAM J. Comput.* **36**(1), 132–157 (2006). <https://doi.org/10.1137/S0097539704442684>
33. Eiermann, M., Ernst, O.G.: A restarted Krylov subspace method for the evaluation of matrix functions. *SIAM J. Numer. Anal.* **44**(6), 2481–2504 (2006). <https://doi.org/10.1137/050633846>
34. Estrada, E.: Characterization of 3D molecular structure. *Chem. Phys. Lett.* **319**(5–6), 713–718 (2000). [https://doi.org/10.1016/S0009-2614\(00\)00158-5](https://doi.org/10.1016/S0009-2614(00)00158-5)
35. Estrada, E.: Virtual identification of essential proteins within the protein interaction network of yeast. *PROTEOMICS* **6**(1), 35–40 (2006). <https://doi.org/10.1002/pmic.200500209>
36. Estrada, E.: *The Structure of Complex Networks: Theory and Applications*. Oxford University Press, Oxford (2012)
37. Estrada, E., Hatano, N.: Statistical-mechanical approach to subgraph centrality in complex networks. *Chem. Phys. Lett.* **439**(1), 247–251 (2007). <https://doi.org/10.1016/j.cplett.2007.03.098>
38. Estrada, E., Higham, D.J.: Network properties revealed through matrix functions. *SIAM Rev.* **52**(4), 696–714 (2010). <https://doi.org/10.1137/090761070>
39. Estrada, E., Rodríguez-Velázquez, J.A.: Subgraph centrality in complex networks. *Phys. Rev. E* **71**(5), 056103 (2005). <https://doi.org/10.1103/PhysRevE.71.056103>
40. Estrada, E., Rodríguez-Velázquez, J.A.: Subgraph centrality and clustering in complex hyper-networks. *Phys. A Stat. Mech. Appl.* **364**, 581–594 (2006). <https://doi.org/10.1016/j.physa.2005.12.002>
41. Estrada, E., Hatano, N., Benzi, M.: The physics of communicability in complex networks. *Phys. Rep.* **514**(3), 89–119 (2012). <https://doi.org/10.1016/j.physrep.2012.01.006>
42. Fenu, C., Martin, D., Reichel, L., Rodríguez, G.: Block Gauss and anti-Gauss quadrature with application to networks. *SIAM J. Matrix Anal. Appl.* **34**(4), 1655–1684 (2013). <https://doi.org/10.1137/120886261>
43. Forsythe, G.E., Leibler, R.A.: Matrix inversion by a Monte Carlo method. *Math. Tables Other Aids Comput* **4**(31), 127–129 (1950). <https://doi.org/10.2307/2002508>

44. Freeman, L.C.: Centrality in social networks conceptual clarification. *Soc. Netw.* **1**(3), 215–239 (1978). [https://doi.org/10.1016/0378-8733\(78\)90021-7](https://doi.org/10.1016/0378-8733(78)90021-7)
45. Gershgorin, S.: Über die abgrenzung der eigenwerte einer matrix. *Izvest. Akad. Nauk SSSR Serija Mat.* **7**(3), 749–754 (1931)
46. Golub, G.H., Meurant, G.: *Matrices. Moments and Quadrature with Applications*. Princeton University Press, Princeton (2009)
47. Graph500. <https://graph500.org/>
48. Guidotti, N.L., Acebrón, J., Monteiro, J.: A stochastic method for solving time-fractional differential equations (2023). <https://doi.org/10.48550/arXiv.2303.15458>
49. Güttel, S., Schweitzer, M.: Randomized sketching for Krylov approximations of large-scale matrix functions. *SIAM J. Matrix Anal. Appl.* **44**(3), 1073–1095 (2023). <https://doi.org/10.1137/22M1518062>
50. Güttel, S.: Funm_kryl toolbox for MATLAB. http://www.guettel.com/funm_kryl/
51. Güttel, S.: Rational Krylov approximation of matrix functions: numerical methods and optimal pole selection. *GAMM-Mitteilungen* **36**(1), 8–31 (2013). <https://doi.org/10.1002/gamm.201310002>
52. Güttel, S., Kressner, D., Lund, K.: Limited-memory polynomial methods for large-scale matrix functions. *GAMM-Mitteilungen* **43**(3), e202000019 (2020). <https://doi.org/10.1002/gamm.202000019>
53. Higham, N.J.: The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.* **26**(4), 1179–1193 (2005). <https://doi.org/10.1137/04061101X>
54. Higham, N.J.: *Functions of Matrices. Other Titles in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia (2008). <https://doi.org/10.1137/1.9780898717778>
55. Hubbell, C.H.: An input–output approach to clique identification. *Sociometry* **28**(4), 377–399 (1965). <https://doi.org/10.2307/2785990>
56. Jacod, J., Protter, P.: *Probability Essentials*. Universitext. Springer, Berlin (2004). <https://doi.org/10.1007/978-3-642-55682-1>
57. Jeong, H., Mason, S.P., Barabási, A.L., Oltvai, Z.N.: Lethality and centrality in protein networks. *Nature* **411**(6833), 41–42 (2001). <https://doi.org/10.1038/35075138>
58. Ji, H., Mascagni, M., Li, Y.: Convergence analysis of Markov Chain Monte Carlo Linear solvers using Ulam–Von Neumann algorithm. *SIAM J. Numer. Anal.* **51**(4), 2107–2122 (2013)
59. Jordán, F., Benedek, Z., Podani, J.: Quantifying positional importance in food webs: A comparison of centrality indices. *Ecol. Model.* **205**(1), 270–275 (2007). <https://doi.org/10.1016/j.ecolmodel.2007.02.032>
60. Katz, L.: A new status index derived from sociometric analysis. *Psychometrika* **18**(1), 39–43 (1953). <https://doi.org/10.1007/BF02289026>
61. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection (2014). <http://snap.stanford.edu/data>
62. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters (2008). <https://doi.org/10.48550/arXiv.0810.1355>
63. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: an approach to modeling networks. *J. Mach. Learn. Res.* **11**, 985–1042 (2010)
64. Martinsson, P.G., Tropp, J.A.: Randomized numerical linear algebra: foundations and algorithms. *Acta Numer.* **29**, 403–572 (2020). <https://doi.org/10.1017/S0962492920000021>
65. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pp. 29–42. ACM, San Diego California USA (2007). <https://doi.org/10.1145/1298306.1298311>
66. Murray, R., Demmel, J., Mahoney, M.W., Erichson, N.B., Melnichenko, M., Malik, O.A., Grigori, L., Luszczek, P., Dereziński, M., Lopes, M.E., Liang, T., Luo, H., Dongarra, J.: *Randomized Numerical Linear Algebra: A Perspective on the Field With an Eye to Software* (2023). <https://doi.org/10.48550/arXiv.2302.11474>
67. Nakatsukasa, Y., Tropp, J.A.: Fast & Accurate Randomized Algorithms for Linear Systems and Eigenvalue Problems (2022). <https://doi.org/10.48550/arXiv.2111.00113>
68. Newman, M.: Network data. <http://www-personal.umich.edu/~mejn/netdata/> (2013)
69. Newman, M.E.J.: The structure and function of complex networks. *SIAM Rev.* **45**(2), 167–256 (2003). <https://doi.org/10.1137/S003614450342480>
70. Newman, M.: *Networks: An Introduction*. Oxford University Press, Oxford (2018)
71. O’Neill, M.E.: PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. *Tech. Rep. HMC-CS-2014-0905*, Harvey Mudd College, Claremont, CA (2014)
72. Rozemberczki, B., Sarkar, R.: Twitch Gamers: A Dataset for Evaluating Proximity Preserving and Structural Role-based Node Embeddings (2021). <https://doi.org/10.48550/arXiv.2101.03091>
73. Vladimir, B., Mrvar, A.: Pajek datasets. <http://vlado.fmf.uni-lj.si/pub/networks/data/default.htm> (2006)

74. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: Intel Math Kernel Library. In: Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y. (eds.) *High-Performance Computing on the Intel® Xeon Phi™: How to Fully Exploit MIC Architectures*, pp. 167–188. Springer, Cham (2014)
75. Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’ networks. *Nature* **393**(6684), 440–442 (1998). <https://doi.org/10.1038/30918>
76. Yang, J., Leskovec, J.: Patterns of temporal variation in online media. In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, WSDM '11*, pp. 177–186. Association for Computing Machinery, New York, NY, USA (2011). <https://doi.org/10.1145/1935826.1935863>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.