Sparse Approximation of Complex Networks

Omar De la Cruz Cabrera* Jiafeng Jin* Lothar Reichel*

Abstract

This paper considers the problem of recovering a sparse approximation $\mathbf{A} \in \mathbb{R}^{n\times n}$ of an unknown (exact) adjacency matrix $\mathbf{A}_{\rm true}$ for a network from a corrupted version of a communicability matrix $\mathbf{C} = \exp(\mathbf{A}_{\rm true}) + \mathbf{N} \in \mathbb{R}^{n\times n}$, where $\mathbf{N}$ denotes an unknown "noise matrix". We consider two methods for determining an approximation $\mathbf{A}$ of $\mathbf{A}_{\rm true}$: (i) a Newton method with soft-thresholding and line search, and (ii) a proximal gradient method with line search. These methods are applied to compute the solution of the minimization problem

$$\arg\min_{\mathbf{A}\in\mathbb{R}^{n\times n}} \left\{ \|\exp(\mathbf{A}) - \mathbf{C}\|_F^2 + \mu \|\mathrm{vec}(\mathbf{A})\|_1 \right\},$$

where $\mu > 0$ is a regularization parameter that controls the amount of shrinkage. We discuss the effect of μ on the computed solution, conditions for convergence, and the rate of convergence of the methods. Computed examples illustrate their performance when applied to directed and undirected networks.

1 Introduction

Many complex phenomena can be usefully modeled as networks. A network can be represented by a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, which consists of a set of *vertices* or *nodes* $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and a set of *edges* $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ that connect the vertices. Edges may be directed (and model one-way streets) or undirected (and model two-way streets). Detailed expositions of the study of complex networks and their use as models are provided by, e.g., Estrada [9] and Newman [16]. Here we only mention that network models often are able to capture essential properties of a given complex phenomenon while being simple enough to be amenable to quantitative analysis.

We say that node v_i is *directly* connected to node v_j if there is a single edge from node v_i pointing to node v_j. In this situation, node v_j is said to be *adjacent* to node v_i. In case the edge between these nodes is undirected, node v_j also is directly connected to node v_i, and v_i also is adjacent to node v_j. An implicit assumption in network models is that of *transitivity* or indirect connection: if node v_i is directly connected to node v_j by a single edge, and v_j is directly connected to node v_k by another single edge, then we can expect that v_i will "influence" v_k somewhat, even if there is no edge between these nodes. This way, even if only fairly few of all pairs of nodes are connected by a single edge, the resulting relationships among all nodes may be quite complex.

Node v_i is said to be *indirectly* connected to node v_j if the latter node can be reached from the former by following at least two edges, which may be directed, from v_i. We observe that complicated pairwise node relationships can be established by using a fairly small amount of information. To some extent, the success of a network as a model may result from the ability to capture many pairwise relationships by means of quite few explicit direct connections.

*Department of Mathematical Sciences, Kent State University, Kent, OH 44242, USA.
Email: odelacru@kent.edu (O. De la Cruz Cabrera), jjin3@kent.edu (J. Jin), reichel@math.kent.edu (L. Reichel)

In many applications, the direct connections between the nodes of a network are explicitly known, i.e., one knows all the edges starting or ending at every node. A question of interest then is how "well connected" each node is. A typical example is a road network: the intersections are regarded as nodes and the edges model road segments between intersections. The problem is to quantify how easy, or difficult, it is to reach one node in the network from other nodes. Several methods for quantifying all the pairwise "communicability" relationships between nodes are based on using the matrix exponential or other matrix functions.

Let $e(v_i \to v_j)$ denote an edge from node v_i to v_j; where the edge may be directed or undirected. A sequence of edges (not necessarily distinct)

$$\{e(v_1 \to v_2), e(v_2 \to v_3), \ldots, e(v_k \to v_{k+1})\} \tag{1}$$

forms a *walk* of length k. If the edges in a walk are distinct, then the walk is referred to as a *path*.

Introduce the adjacency matrix $\mathbf{A}_{\rm true} = [a_{ij}^{(\rm true)}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$ associated with a graph $\mathcal{G}$ and assume that the graph is unweighted. The matrix $\mathbf{A}_{\rm true}$ then has the property that $a_{ij}^{(\rm true)} = 1$ if node v_j is adjacent to node v_i.

The adjacency matrix $\mathbf{A}_{\rm true}$ is *sparse* in most applications, i.e., the matrix has many more zero entries than entries one. The matrix is symmetric if for each edge there is also an edge in the opposite direction. The graph determined by such an adjacency matrix is said to be undirected. If at least one edge of a graph is directed, then the graph is referred to as directed. We will assume that the network associated with $\mathbf{A}_{\rm true}$ is connected and that there are no loops and multiple edges. Then all diagonal entries of $\mathbf{A}_{\rm true}$ vanish.

Estrada and Rodriguez-Velazquez [10] define for graphs with a symmetric adjacency matrix the communicability matrix

$$\mathbf{C} = [c_{ij}]_{i,j=1}^n = \exp(\mathbf{A}_{\rm true}).$$

The entry c_{ij}, $i \neq j$, denotes the communicability between the nodes v_i and v_j. A relatively large value generally implies that it is easy for the nodes v_i and v_j to communicate. Estrada and Rodriguez-Velazquez [10] measure the importance of the node v_i by the subgraph centrality c_{ii}. Related measures of communicability and node importance can be defined when the adjacency matrix $\mathbf{A}_{\rm true}$ is nonsymmetric; see [6] for a discussion.

Consider the power series expansion

$$\exp(\mathbf{A}_{\rm true}) = \mathbf{I} + \mathbf{A}_{\rm true} + \frac{\mathbf{A}_{\rm true}^2}{2!} + \frac{\mathbf{A}_{\rm true}^3}{3!} + \ldots \ . \tag{2}$$

Let $\mathbf{A}_{\rm true}^k = [a_{ij}^{(k,\rm true)}]_{i,j=1}^n$. Then a nonvanishing entry $a_{ij}^{(k,\rm true)}$ indicates that there is at least one walk of k edges from node v_i to node v_j. The denominators in the terms of the expansion (2) ensure that the expansion converges and that terms $\frac{\mathbf{A}_{\rm true}^k}{k!}$ with k large contribute only little to $\exp(\mathbf{A}_{\rm true})$, i.e., they contribute very little to the communicability and node importance. It follows that short walks typically are more important than long ones. This is in agreement with the fact that messages propagate better along short walks than along long ones.

We consider the process of determining node importance using the communicability matrix as the "forward" problem. In many applications, one is interested in the "inverse" problem: data about communicability for all pairs of nodes are available (possibly contaminated by noise), and the task is to infer which nodes are directly connected, rather than indirectly connected. A typical application in which this inverse problem arises is in *gene regulatory networks*: the nodes represent genes in the genome of a particular organism and edges represent regulatory relationships, in which the activity of one gene increases or decreases the activity of another gene by a specific molecular mechanism. Pairwise measures of correlation are assumed to be known. A high (absolute) correlation between two genes might be due to direct or indirect regulatory effects, or due to noise. Elucidating which pairwise relationships are direct can be important to understand the behavior

of a system. We therefore are interested in inferring direct connections between pairs of nodes in a network from the associated (possibly noise-contaminated) communicability matrix.

Thus, suppose that there is an underlying network with known nodes $v_1, \ldots, v_n$, but that the edges are not known. Assume, moreover, that for all $i, j \in \{1, \ldots, n\}$, we are given a measure of communicability c_{ij} from node v_i to v_j; this measure may be affected by noise. Our goal is to infer which nodes are directly connected. Our modeling assumption is that

$$\mathbf{C} = [c_{ij}]_{i,j=1}^{n} = \exp(\mathbf{A}_{\text{true}}) + \mathbf{N}, \tag{3}$$

where $\mathbf{A}_{\text{true}} = [a_{ij}^{(\text{true})}]_{i,j=1}^{n}$, with $a_{ij}^{(\text{true})} \in \{0, 1\}$, is an unknown adjacency matrix to be determined and the matrix $\mathbf{N} \in \mathbb{R}^{n\times n}$ models the error (also referred to as noise) in the matrix $\mathbf{C}$.

In the noise-free case, one can in theory compute the matrix logarithm $\mathbf{A}_{\text{true}}$ of $\mathbf{C}$ exactly. One would expect the computed matrix to be sparse since $\mathbf{A}_{\text{true}}$ is sparse. However, even a small amount of perturbation in the data $\mathbf{C}$ typically makes the matrix $\log(\mathbf{C})$ dense. The perturbation may be present in the available data in the form of noise, or be introduced in the form of round-off errors during the computations. Moreover, if $\mathbf{A}_{\text{true}}$ is nonsymmetric, the computed matrix logarithm may have complex entries. It follows that some kind of regularization is necessary to determine an accurate approximation of $\mathbf{A}_{\text{true}}$. Since our interest lies in determining a sparse adjacency matrix, we will use a penalized optimization approach to regularize the problem.

We propose the use of a *sparse approximate matrix logarithm* as a way to determine edges between adjacent nodes in a network with relatively few edges. Consider the minimization problem

$$\arg\min_{\mathbf{A}\in\mathbb{R}^{n\times n}} \left\{ \frac{1}{2} \| \exp(\mathbf{A}) - \mathbf{C} \|_F^2 + \mu \|\text{vec}(\mathbf{A})\|_1 \right\}, \tag{4}$$

where the matrix $\mathbf{C} \in \mathbb{R}^{n\times n}$ is known and denotes the available imperfect data that has been tainted by an unknown error $\mathbf{N}$; (cf. (3)). Let $\mathbf{C}_{\text{true}}$ be the unknown noise-free communicability matrix associated with $\mathbf{C}$; thus, $\mathbf{C}_{\text{true}} = \exp(\mathbf{A}_{\text{true}})$, where $\mathbf{A}_{\text{true}}$ is the desired adjacency matrix. The trailing term in (4) is a regularization term, where the function vec stacks the columns of the matrix $\mathbf{A}$ in a vector. The solution $\mathbf{A}$ of (4) furnishes an approximation of $\mathbf{A}_{\text{true}}$. The parameter $\mu > 0$ is a regularization parameter which balances the relative influence of the two terms in (4). Throughout this paper $\|\cdot\|_F$ denotes the matrix Frobenius norm and $\|\cdot\|_1$ stands for the vector 1-norm.

We will use Newton's method described by Higham et al. [11, p. 285] or the proximal gradient (PG) method [1] to determine a solution of (4). The regularization term in (4) is known to promote sparsity of the computed solution $\mathbf{A}$. This term can be implemented with soft-thresholding. We therefore refer to Newton's method applied to the solution of (4) as Newton's method with soft-thresholding (NST).

This paper is organized as follows. Section 2 collects some useful properties of matrix functions. We describe the NST method with line search in Section 3, where we also consider convergence properties. Section 4 reviews the PG method with line search and discusses its convergence properties. Computed illustrations for some small and moderately large networks are presented in Section 5. Concluding remarks are provided in Section 6. We explain in the appendix why the MATLAB function `logm` is not useful for determining a sparse approximation of $\mathbf{A}_{\text{true}}$ from the communicability matrix $\mathbf{C}$.

2 Some properties of matrix functions

This section reviews some properties of matrix functions and their derivatives. These properties will be used in the sequel.

2.1 The Fréchet derivative of a matrix function

Consider a matrix function $\mathbf{f} : \mathbf{A} \in \mathbb{R}^{n\times n} \to \mathbf{f}(\mathbf{A}) \in \mathbb{R}^{n\times n}$, which we assume to be differentiable. Let $\mathbf{E} \in \mathbb{R}^{n\times n}$. We have

$$\mathbf{f}(\mathbf{A}+\mathbf{E}) = \mathbf{f}(\mathbf{A}) + \mathbf{L}_{\mathbf{f}}(\mathbf{A},\mathbf{E}) + o(\|\mathbf{E}\|_p), \tag{5}$$

where the norm $\|\cdot\|_p$ is any matrix norm, and $\mathbf{L}_{\mathbf{f}}(\mathbf{A},\mathbf{E})$ denotes the Fréchet derivative of $\mathbf{f}$ at $\mathbf{A}$ in the direction $\mathbf{E}$; see, e.g., [11] for details. For notational convenience, we will sometimes denote $\mathbf{L}_{\mathbf{f}}(\mathbf{A},\mathbf{E})$ by $\partial\mathbf{f}(\mathbf{A})/\partial\mathbf{E}$.

2.2 Fréchet derivatives of the matrix trace

Let $\mathbf{A} = [a_{ij}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$. Then the matrix $\mathbf{A}$ and its transpose, $\mathbf{A}^T$, have the same trace

$$\mathrm{Tr}\{\mathbf{A}\} = \sum_{i=1}^n a_{ii} = \mathrm{Tr}\{\mathbf{A}^T\}.$$

Consider the matrices $\mathbf{X} = [x_{ij}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$, $\mathbf{Y} = [y_{ij}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$, and $\mathbf{Z} = [z_{ij}]_{i,j=1}^n \in \mathbb{R}^{n\times n}$. Then

$$\mathrm{Tr}\{\mathbf{X}+\mathbf{Y}\} = \mathrm{Tr}\{\mathbf{X}\} + \mathrm{Tr}\{\mathbf{Y}\}$$

and

$$\mathrm{Tr}\{\mathbf{XYZ}\} = \mathrm{Tr}\{\mathbf{YZX}\} = \mathrm{Tr}\{\mathbf{ZXY}\} = \sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n x_{ij}y_{jk}z_{ki};$$

see, e.g., [15]. The Fréchet derivative in the direction $\mathbf{A}$ of the matrix trace satisfies the following properties

(i) $$\frac{\partial\,\mathrm{Tr}\{\mathbf{X}+\mathbf{Y}\}}{\partial\mathbf{A}} = \frac{\partial\,\mathrm{Tr}\{\mathbf{X}\}}{\partial\mathbf{A}} + \frac{\partial\,\mathrm{Tr}\{\mathbf{Y}\}}{\partial\mathbf{A}},$$

(ii) $$\frac{\partial\,\mathrm{Tr}\{\mathbf{XYZ}\}}{\partial\mathbf{A}} = \frac{\partial\,\mathrm{Tr}\{\mathbf{YZX}\}}{\partial\mathbf{A}} = \frac{\partial\,\mathrm{Tr}\{\mathbf{ZXY}\}}{\partial\mathbf{A}},$$

(iii) $$\frac{\partial\,\mathrm{Tr}\{\mathbf{XY}\}}{\partial\mathbf{A}} = \frac{\partial\,\mathrm{Tr}\{\mathbf{X}_c\mathbf{Y}\}}{\partial\mathbf{A}} + \frac{\partial\,\mathrm{Tr}\{\mathbf{XY}_c\}}{\partial\mathbf{A}},$$

where the matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are functions of $\mathbf{A}$, and $\mathbf{X}_c$ and $\mathbf{Y}_c$ are treated as constant matrices; see [3] for further information.

3 The Newton method

This section discusses the use of Newton's method with soft-thresholding (NST) and line search to compute a solution of (4). We derive the NST method from the classical Newton method. The latter is considered, e.g., by Dieci [8] and Higham [11]. Subsection 3.1 describes the method, while its convergence is considered in Subsection 3.2.

3.1 Newton's method with soft-thresholding and line search

Introduce the matrix function $\mathbf{f}(\mathbf{A}) = \exp(\mathbf{A}) - \mathbf{C} \in \mathbb{R}^{n\times n}$. Then eq. (4) can be expressed as

$$\min_{\mathbf{A}\in\mathbb{R}^{n\times n}} \left\{ \frac{1}{2}\|\mathbf{f}(\mathbf{A})\|_F^2 + \mu\|\mathrm{vec}(\mathbf{A})\|_1 \right\}. \tag{6}$$

Let $\mathbf{A}_0$ be an initial approximate solution of (6). Then the iterations with the NST method (without line search) can be expressed as

$$\begin{cases} \mathbf{L}_\mathbf{f}(\mathbf{A}_k, \mathbf{E}_k) = -\mathbf{f}(\mathbf{A}_k), \\ \tilde{\mathbf{A}}_{k+1} = \mathbf{A}_k + \mathbf{E}_k, \qquad k = 0, 1, \ldots, \\ \mathbf{A}_{k+1} = \mathbf{S}_\mu(\tilde{\mathbf{A}}_{k+1}). \end{cases} \tag{7}$$

The step $\mathbf{E}_k$ is determined in the first line; see below for details. Moreover, $\mathbf{S}_\mu$ denotes the soft-thresholding operator. It is applied entry-wise to $\tilde{\mathbf{A}}_{k+1}$ and is for a real argument a defined as

$$\mathbf{S}_\mu(a) = \begin{cases} \mathrm{sign}(a)(|a| - \mu), & \text{if } |a| > \mu, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Here μ is the regularization parameter in (4). It is well known that the ℓ_1-penalty term in (4) is equivalent to application of the soft-threshold operator (8) element-wise.

To determine the step $\mathbf{E}_k$, we use eq. (5) without the remainder term. We obtain for $\mathbf{f}(\mathbf{A}_k) = \exp(\mathbf{A}_k) - \mathbf{C}$,

$$\begin{aligned} \mathbf{L}_\mathbf{f}(\mathbf{A}_k, \mathbf{E}_k) &\approx \exp(\mathbf{A}_k + \mathbf{E}_k) - \exp(\mathbf{A}_k) \\ &\approx \mathbf{E}_k + \frac{\mathbf{A}_k\mathbf{E}_k + \mathbf{E}_k\mathbf{A}_k}{2!} + \frac{\mathbf{A}_k^2\mathbf{E}_k + \mathbf{A}_k\mathbf{E}_k\mathbf{A}_k + \mathbf{E}_k\mathbf{A}_k^2}{3!} + \ldots \\ &= \sum_{\ell=1}^{\infty} \frac{1}{\ell!} \sum_{j=0}^{\ell-1} \mathbf{A}_k^j \mathbf{E}_k \mathbf{A}_k^{\ell-1-j}. \end{aligned} \tag{9}$$

The application of Newton's method requires the determination of the Newton step $\mathbf{E}_k$ in (7) in each iteration. We proceed as follows when the matrices $\mathbf{A}_k$ and $\mathbf{E}_k$ do not commute: Using (9), we approximate the first equation of (7) by

$$\sum_{\ell=1}^{\infty} \frac{1}{\ell!} \sum_{j=0}^{\ell-1} \mathbf{A}_k^j \mathbf{E}_k \mathbf{A}_k^{\ell-1-j} = \mathbf{C} - \exp(\mathbf{A}_k).$$

This expression can be written in the form

$$\sum_{\ell=1}^{\infty} \frac{1}{\ell!} \sum_{j=0}^{\ell-1} (\mathbf{A}_k^{\ell-1-j})^T \otimes \mathbf{A}_k^j \mathrm{vec}(\mathbf{E}_k) = \mathrm{vec}(\mathbf{C} - \exp(\mathbf{A}_k)), \tag{10}$$

where $\otimes$ denotes the tensor product. The infinite sum is truncated when the terms are small. Generally, the system matrix in the above expression is nonsingular; otherwise the Moore-Penrose pseudoinverse can be used.

In the special case when $\mathbf{A}_k$ and $\mathbf{E}_k$ commute, eq. (9) simplifies to

$$\mathbf{L}_\mathbf{f}(\mathbf{A}_k, \mathbf{E}_k) = \exp(\mathbf{A}_k)\mathbf{E}_k = \mathbf{E}_k \exp(\mathbf{A}_k) = \exp\left(\frac{\mathbf{A}_k}{2}\right) \cdot \mathbf{E}_k \cdot \exp\left(\frac{\mathbf{A}_k}{2}\right).$$

PROPOSITION 1. *([11, Chapter 4] Assuming that the matrices $\mathbf{A}_k$ and $\mathbf{E}_k$ commute. Then $\mathbf{L}_\mathbf{f}(\mathbf{A}_k, \mathbf{E}_k) = \mathbf{L}_\mathbf{f}(\mathbf{A}_k)\mathbf{E}_k = \mathbf{E}_k\mathbf{L}_\mathbf{f}(\mathbf{A}_k)$, where $\mathbf{L}_\mathbf{f}(\mathbf{A}_k)$ is given by $\mathbf{f}'(\mathbf{A}_k)$, and $\mathbf{f}'(\mathbf{A}_k)$ denotes the derivative of the scalar function f evaluated at $\mathbf{A}_k$.*

Assume that the matrices $\mathbf{A}_k$ and $\mathbf{E}_k$ commute, we will use the formula

$$\mathbf{E}_k = \exp\left(\frac{-\mathbf{A}_k}{2}\right) \cdot \mathbf{C} \cdot \exp\left(\frac{-\mathbf{A}_k}{2}\right) - \mathbf{I}$$

to determine the Newton step $\mathbf{E}_k$ in eq. (7) in computed examples of Section 5. Then $\mathbf{E}_k$ is symmetric when $\mathbf{C}$ is symmetric, which implies that $\mathbf{A}_{k+1}$ is symmetric.

Newton's method based on the computations (7) might not converge, because the Newton step matrix $\mathbf{E}_k$ may be of too large norm. This difficulty can be remedied by implementing a line search in direction $\mathbf{E}_k$. Newton's method with a line search yields a globally convergent iterative method at least for sufficiently smooth functions $\mathbf{f}$; see [7, Chapter 6]. Algorithm 3.1.1 describes Newton's method with a line search based on backtracking. The method is referred to as Newton's method with soft-thresholding and line search (NST-LS). Properties of the Newton method with backtracking and for smooth functions $\mathbf{f}$, without soft-thresholding, are discussed in [7, Chapter 6].

Algorithm 3.1.1 The NST-LS method for the situation when the matrices $\mathbf{C}$ and $\mathbf{A}_0$ commute

Input: $\mathbf{C} \in \mathbb{R}^{n\times n}$, regularization parameter $\mu > 0$, initial approximate solution $\mathbf{A}_0 \in \mathbb{R}^{n\times n}$, backtracking parameter $\beta > 0$

Output: $\mathbf{A}$ (the computed approximation of $\mathbf{A}_{\text{true}}$)

for $k = 1, 2, \ldots$ until convergence **do**

$\mathbf{E}_{k-1} = \exp(-\mathbf{A}_{k-1}/2) \cdot \mathbf{C} \cdot \exp(-\mathbf{A}_{k-1}/2) - \mathbf{I}$;

$t_{k-1} = 1$;

while $\mathbf{f}(\mathbf{A}_{k-1} + t_{k-1}\mathbf{E}_{k-1}) > \mathbf{f}(\mathbf{A}_{k-1}) + 0.5 \cdot t_{k-1}\|\mathbf{E}_{k-1}\|^2$ **do**

$t_{k-1} = \beta t_{k-1}$;

end while

$\tilde{\mathbf{A}}_k = \mathbf{A}_{k-1} + t_{k-1}\mathbf{E}_{k-1}$;

$\mathbf{A}_k = \mathbf{S}_\mu(\tilde{\mathbf{A}}_k)$;

end for

Throughout this paper $\|\cdot\|$ denotes the spectral norm.

We set $\beta = 0.8$. Since the matrix $\mathbf{C}$ is contaminated by the error matrix $\mathbf{N}$, (cf.(3)), the matrices $\mathbf{C}$ and $\mathbf{A}_0$ commute only rarely. However, if $\mathbf{A}_0$ and $\mathbf{A}_{\text{true}}$ commute, and the noise matrix $\mathbf{N}$ is of norm much smaller than $\mathbf{C}$, then Algorithm 3.1.1 yields good results; see Section 5. The NST-LS method for the situation when the matrices $\mathbf{C}$ and $\mathbf{A}_0$ are not close to commuting matrices is more difficult to implement. It requires the evaluation of an approximation of the solution of the linear system of equations (10). We omit the details.

3.2 Convergence of the NST method

In some circumstances, it is possible to prove the existence of a useful fixed point for the NST iteration built for $\mathbf{C} = \exp(\mathbf{A}) + \mathbf{N}$. In general, this fixed point is not the matrix $\mathbf{A}$ itself, but rather a "shrunken" version of $\mathbf{A}$, in which the entries with value zero are correct but the entries with value 1 in $\mathbf{A}$ are reduced by a factor of $1 - \mu$, and are also individually perturbed by a small amount.

It should be noted that, if $\mathbf{N}$ is small enough, the Newton iterations (without thresholding) may converge to the (real) matrix logarithm of $\mathbf{C}$. However, the result will in general not be sparse. In this section we show that the soft-thresholded iteration of the NST method will under suitable conditions recover the exact sparsity pattern of $\mathbf{A}$.

The matrix $\mathbf{A}$ models an unweighted network, which may be directed or undirected. Thus, $\mathbf{A}$ has entries 0 or 1, and is not necessarily symmetric. The noise matrix $\mathbf{N}$ will be regarded as fixed, except in Corollary 1, where it is assumed to be random. Also, define $\mathbf{C} = \exp(\mathbf{A}) + \mathbf{N}$.

A simple one-sided version of the NST method will be used, and we will not use line search but, instead, use a fixed step size of 1; thus, the iteration is given by

$$\mathbf{A}_{k+1} = \mathbf{S}_\mu \left(\mathbf{A}_k - \mathbf{I} + \exp(-\mathbf{A}_k)\mathbf{C} \right). \tag{11}$$

We seek to prove a statement of the form: If the noise $\mathbf{N}$ is small enough, then there exists $\mu > 0$ such that the iterations (11) have a fixed point of the form $(1-\mu)\mathbf{A} + \mathbf{B}$ with $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n\times n}$ and $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{n\times n}$, where $b_{ij} = 0$ whenever $a_{ij} = 0$ and $|b_{ij}| < \mu$ for all i, j. First, we state a condition that suffices for the existence of the postulated fixed point.

PROPOSITION 2. *Assume that $0 < \mu \leq 1/2$, and assume that there exists a matrix $\mathbf{G} = [g_{ij}] \in \mathbb{R}^{n\times n}$ that satisfies:*

1. *$|g_{ij}| < \mu$ for all i, j,*
2. *$\mathbf{G} = \mathbf{B} + \mathbf{D}$, with $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{n\times n}$ and $\mathbf{D} = [d_{ij}] \in \mathbb{R}^{n\times n}$ such that:*
   (a) *$b_{ij} = g_{ij}$ and $d_{ij} = 0$, whenever $a_{ij} = 1$, and*
   (b) *$d_{ij} = g_{ij}$ and $b_{ij} = 0$, whenever $a_{ij} = 0$,*
3. *we have that $\mathbf{F}(\mathbf{G}) = \mathbf{0}$, where*

$$\mathbf{F}(\mathbf{G}) := \exp(-((1-\mu)\mathbf{A} + \mathbf{B}))\mathbf{C} - \mathbf{I} - \mu\mathbf{A} - \mathbf{D}.$$

Then $(1-\mu)\mathbf{A} + \mathbf{B}$ is a fixed point for the iteration (11).

Proof. Suppose such $\mathbf{G} = \mathbf{B} + \mathbf{D}$ exists, satisfying $|g_{ij}| < \mu$ for all i, j. The Newton iteration step applied to $(1-\mu)\mathbf{A} + \mathbf{B}$ results in

$$(1-\mu)\mathbf{A} + \mathbf{B} - \mathbf{I} + \exp(-((1-\mu)\mathbf{A} + \mathbf{B}))\mathbf{C}.$$

If we subtract $\mathbf{F}(\mathbf{G}) = \mathbf{0}$, then we change nothing, yet we obtain that the result equals

$$\mathbf{A} + \mathbf{B} + \mathbf{D} = \mathbf{A} + \mathbf{G}.$$

Now, applying the soft-thresholding step, we obtain

$$\mathbf{S}_\mu \left(\mathbf{A} + \mathbf{G} \right) = (1-\mu)\mathbf{A} + \mathbf{B}.$$

Indeed, for each i, j: if $a_{ij} = 0$, then $|a_{ij} + g_{ij}| = |g_{ij}| < \mu$ and the result of soft-thresholding is 0, which equals $(1-\mu)a_{ij} + b_{ij}$; if $a_{ij} = 1$, then $a_{ij} + g_{ij} > 1 - \mu \geq \mu$ (because we assumed $\mu \leq 1/2$), so the result of soft-thresholding is $a_{ij} + g_{ij} - \mu = 1 - \mu + g_{ij} = (1-\mu)a_{ij} + b_{ij}$. ∎

This proposition reduces our problem to finding a root for the function $\mathbf{F}$, that is, solving a nonlinear system of n^2 equations in n^2 variables, and establishes that the root $\mathbf{G}$ is small (in the sense that $|g_{ij}| < \mu$ for all i, j).

A remark on notation: When necessary, we will make explicit the dependence of $\mathbf{F}$ on μ and $\mathbf{N}$ by writing $\mathbf{F}_{\mu,\mathbf{N}}$. Also, we will abuse notation and use $\mathbf{F}(\mathbf{G})$ to refer to the map from $\mathbb{R}^{n^2}$ to $\mathbb{R}^{n^2}$ obtained by flattening the matrices.

Consider the matrix $\mathbf{K} = [k_{rs}] \in \mathbb{R}^{m\times m}$ defined as follows. Recall that the edges of the network are given by $e_1, \ldots, e_m$; each edge corresponds to one entry with value 1 in $\mathbf{A}$. Let k_{rs} be the partial derivative of $(\mathbf{F}(\mathbf{G}))_{i'j'}$ with respect to $\mathbf{G}_{ij}$ evaluated at $\mathbf{G} = \mathbf{0}$, and then evaluated at $\mathbf{N} = \mathbf{0}$ and $\mu = 0$, where e_r is the edge connecting node v_i to v_j, and e_s is the edge connecting $v_{i'}$ to $v_{j'}$. In other words, $\mathbf{K}$ is the submatrix of the Jacobian matrix of (a flattened version of) $\mathbf{F}$, evaluated at $\mathbf{G} = \mathbf{0}$, corresponding to the entries that equal 1 in $\mathbf{A}$, and then evaluated at $\mathbf{N} = \mathbf{0}$ and $\mu = 0$.

THEOREM 1. *If the matrix* $\mathbf{K}$ *defined above is nonsingular, then there exists* $\eta > 0$ *such that, whenever* $\|\mathbf{N}\|_F < \eta$*, there exists* $\mu > 0$ *and a matrix* $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{n\times n}$ *satisfying* $b_{ij} = 0$ *when* $a_{ij} = 0$ *and* $|b_{ij}| < \mu$ *otherwise, so that* $(1-\mu)\mathbf{A} + \mathbf{B}$ *is a fixed point for the NST iteration in equation* (11).

Proof. Consider the map $\mathbf{H} : \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R} \to \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}$ given by

$$\mathbf{H}(\mathbf{G}, \mathbf{N}, \mu) = (\mathbf{F}_{\mu,\mathbf{N}}(\mathbf{G}), \mathbf{N}, \mu)\,.$$

The Jacobian matrix $\mathbf{J}_\mathbf{H} \in \mathbb{R}^{(2n^2+1)\times(2n^2+1)}$ has the block structure

$$\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0) = \begin{bmatrix} \mathbf{K} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_1 & -\mathbf{I}_{n^2-m} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{I}_{n^2} & \mathbf{0} \\ \mathbf{z}_1 & \mathbf{z}_2 & \mathbf{0} & 1 \end{bmatrix}.$$

The first block in the second column is zero because changing nonzero elements of $\mathbf{D}$ does not change the values of the entries of $\mathbf{F}(\mathbf{G})$ corresponding to nonzero entries of $\mathbf{B}$. The other zero blocks result from the fact that changing values in $\mathbf{G}$ does not affect the values of $\mathbf{N}$ or μ, and $\mathbf{N}$ and μ do not affect each other.

Since $\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0)$ is block triangular, its determinant equals $\det(\mathbf{K})(-1)^{n^2-m}$. By our hypothesis about $\mathbf{K}$, the matrix $\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0)$ is nonsingular, and since $\mathbf{H}(\mathbf{0},\mathbf{0},0) = (\mathbf{0},\mathbf{0},0)$, by the Inverse Function Theorem, there are neighborhoods $\mathcal{U}, \mathcal{V}$ of $(\mathbf{0},\mathbf{0},0)$ in $\mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R}$ such that $\mathbf{H} : \mathcal{U} \to \mathcal{V}$ is invertible, $\mathbf{H}^{-1} : \mathcal{V} \to \mathcal{U}$ is differentiable, and the Jacobian matrix of $\mathbf{H}^{-1}$ evaluated at $(\mathbf{0},\mathbf{0},0)$ equals $(\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0))^{-1}$.

We claim now that the bottom row of $(\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0))^{-1}$ equals $[\mathbf{0},\mathbf{0},\mathbf{0},1]$. To prove this, we first need to establish that the blocks $\mathbf{z}_1$ and $\mathbf{z}_2$ of $\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0)$ are both zero. We do this by computing $\frac{\partial}{\partial\mu}\mathbf{F}_{\mu,\mathbf{N}}(\mathbf{G})$ and evaluating the result at $\mathbf{0}$. This yields

$$\frac{\partial}{\partial\mu}\mathbf{F}_{\mu,\mathbf{N}}(\mathbf{G}) = \left(\frac{\partial}{\partial\mu}\Big(\exp(-((1-\mu)\mathbf{A}+\mathbf{B}))\Big)\right)\mathbf{C} - \mathbf{A}.$$

We now expand $\exp(-((1-\mu)\mathbf{A}+\mathbf{B}))$ as a power series, and differentiate term by term. Since $\mathbf{A}$ and $\mathbf{B}$ do not necessarily commute, the resulting power series does not simplify; however, when evaluating at $(\mathbf{0},\mathbf{0},0)$, all terms that contain a factor of μ or $\mathbf{B}$ vanish, and the remaining terms add up to $\mathbf{A}\exp(-\mathbf{A})$. Since $\mathbf{N}$ also is evaluated as zero, $\mathbf{C}$ reduces to $\exp(\mathbf{A})$, and we obtain

$$\left(\frac{\partial}{\partial\mu}\mathbf{F}_{\mu,\mathbf{N}}(\mathbf{G})\right)(\mathbf{0},\mathbf{0},0) = \mathbf{A}\exp(-\mathbf{A})\exp(\mathbf{A}) - \mathbf{A} = \mathbf{0}.$$

Having that blocks $\mathbf{z}_1$ and $\mathbf{z}_2$ of $\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0)$ are both zero, we can now show that the bottom row of $(\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0))^{-1}$ equals $[\mathbf{0},\mathbf{0},\mathbf{0},1]$. $\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0)$ is block triangular, with one block of size $2n^2 \times 2n^2$ at the top left, and a block of size 1×1 at the bottom right; therefore, $(\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0))^{-1}$ has the same block structure, and its bottom row must equal $[\mathbf{0},\mathbf{0},\mathbf{0},1]$.

Having that the bottom row of $(\mathbf{J}_\mathbf{H}(\mathbf{0},\mathbf{0},0))^{-1}$ equals $[\mathbf{0},\mathbf{0},\mathbf{0},1]$ means that, as μ increases from 0, the entries in $\mathbf{F}^{-1}(\mathbf{0})$ grow slower than μ. Thus, we can pick $\mu_0 > 0$ such that $|g_{ij}| < \mu_0$, for all i, j, with $\mathbf{G} = \mathbf{F}^{-1}(\mathbf{0})$, and such that $(\mathbf{F}^{-1}(\mathbf{0}),\mathbf{0},\mu_0) \in \mathcal{U}$. That is, the intersection of the open sets $\mathcal{U}$ and

$$\{(\mathbf{G},\mathbf{N},\mu) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \times \mathbb{R} : |g_{ij}| < \mu_0, \text{ for all } i,j\}$$

is nonempty and open. Let $\mathcal{V}'$ denote the image of this open set under $\mathbf{H}$; $\mathcal{V}'$ is open and contains the point $(\mathbf{0}, \mathbf{0}, \mu_0)$. There exists an open ball $\mathcal{W} \subset \mathbb{R}^{n \times n}$ centered at $\mathbf{0}$, such that if $\mathbf{N} \in \mathcal{W}$ then $(\mathbf{0}, \mathbf{N}, \mu_0) \in \mathcal{V}'$.

Finally, letting η be the radius of the ball $\mathcal{W}$, we get that $\mathbf{G} = \mathbf{F}^{-1}(\mathbf{0})$ and μ_0 satisfy the hypotheses of Proposition 2, so a fixed point of the specified form exists, where the matrix $\mathbf{B}$ is extracted from $\mathbf{G}$. ■

If we would like the noise term to be random, then we can obtain the following result.

COROLLARY 1. *Suppose that the matrix* $\mathbf{K}$ *defined above is nonsingular, and assume that the entries of* $\mathbf{N}$ *are independent random variables with mean 0 and variance* σ^2*. With high probability, provided that* σ^2 *is small enough, there exists* $\mu > 0$ *and a matrix* $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{n \times n}$ *such that* $b_{ij} = 0$ *when* $a_{ij} = 0$ *and* $|b_{ij}| < \mu$ *otherwise, so that* $(1 - \mu)\mathbf{A} + \mathbf{B}$ *is a fixed point for the NST iteration in eq.* (11)*.*

That is, for all $\varepsilon > 0$ *there exists* $\delta > 0$ *such that if* $\sigma^2 < \delta$*, then the probability of the conclusion above holding is greater than* $1 - \epsilon$*.*

Proof. Let $\mathbf{A}$ be such that $\mathbf{K}$ is nonsingular, and let $\varepsilon > 0$. Since $E(n_{ij}^2) = \sigma^2$ for all i, j, we have $E\|\mathbf{N}\|_F^2 = n^2\sigma^2$. By Markov's inequality,

$$P\left(\|\mathbf{N}\|_F^2 \geq \eta^2\right) \leq \frac{n^2\sigma^2}{\eta^2},$$

where $\eta > 0$ is the number whose existence is guaranteed in the statement of Theorem 1. Taking $\delta = \frac{\varepsilon\eta^2}{n^2}$, we conclude that $\sigma^2 < \delta$ implies that $P(\|\mathbf{N}\|_F < \eta) > 1 - \varepsilon$. Thus, the probability that the conclusion of Theorem 1 holds also is at least $1 - \varepsilon$. ■

The following result applies Theorem 1.

THEOREM 2. *Let* $\mathbf{A}$ *be the adjacency matrix of a directed graph with no directed cycles (that is, a* directed acyclic graph, DAG*). Then the conclusions of Theorem 1 and of Corollary 1 hold.*

Proof. It suffices to show that the matrix $\mathbf{K}$ defined above is nonsingular. Let the edge e_r in the graph point from v_i to v_j. We say that an edge e_s *detours* e_r if e_s is part of a (directed) walk that starts at v_i and ends at v_j. Clearly, every edge detours itself. We claim that, since the graph is acyclic, given two distinct edges e_r and e_s, they cannot both detour each other. Indeed, assume that e_r points from v_i to v_j and e_s points from v_k to v_l, and that both detour each other. Assume, without loss of generality, that $v_i \neq v_k$. Let w_1 be a walk that starts at v_i, ends at v_j, and contains e_s, and let w_2 be a walk that starts at v_k, ends in v_l, and contains e_r. Then we can take the initial segment of w_1 that starts at v_i and ends v_k, and join it to the initial segment of w_2 that starts at v_k and ends at v_i, obtaining a directed cycle. (If $v_i = v_k$, then we must have $v_j \neq v_l$, and a similar argument can be made to obtain a cycle, using final segments of the walks.)

The entries of $\mathbf{K}$ are obtained by computing partial derivatives, evaluated at $\mathbf{0}$, of the entries of $\mathbf{F}(\mathbf{G})$, but only for those entries that correspond to edges (i.e., values of 1 in $\mathbf{A}$). From the definition of $\mathbf{F}$ (see eq. (11)), we see that this corresponds to perturbing one of the entries of $\mathbf{B}$ away from zero by a small amount, and measuring the change in the entries of the output. Let the matrix $\delta\mathbf{B}_r \in \mathbb{R}^{n \times n}$ have all entries zero except for the entry (i, j), which has a small nonzero value δx, where e_r points from v_i to v_j.

We claim that the only entries that change from $\mathbf{F}(\mathbf{0})$ to $\mathbf{F}(\delta\mathbf{B}_r)$ correspond to edges e_s that are detoured by e_r. Indeed, notice first that I, $\mu\mathbf{A}$, and $\mathbf{D}$ are all constants, so the only change can happen in the term $\exp(-((1 - \mu)\mathbf{A} + \delta\mathbf{B}_r))\exp(\mathbf{A})$. Expanding the two power series and multiplying term by term, we end up with three types of term: Those that do not contain a factor of $\delta\mathbf{B}_r$, those that contain exactly one factor of $\delta\mathbf{B}_r$, and those that contain two or more factors of $\delta\mathbf{B}_r$. The terms in the first group cancel out when we

compute $\mathbf{F}(\delta\mathbf{B}_r) - \mathbf{F}(\mathbf{0})$, and the terms in the last group vanish when we divide $\mathbf{F}(\delta\mathbf{B}_r) - \mathbf{F}(\mathbf{0})$ by δx and let $\delta x \to 0$. This only leaves the terms in the second group, which are of the form

$$\frac{1}{d_1!d_2!}\mathbf{A}^{d_1}\delta\mathbf{B}_r\mathbf{A}^{d_2}. \tag{12}$$

There also could be factors of $1-\mu$, but this expression is evaluated at $\mu = 0$. Therefore, these factors disappear.

Consider now an edge e_s in the graph, pointing from v_k to v_l. The (k,l)th entry of the matrix $\mathbf{A}^{d_1}\delta\mathbf{B}_r\mathbf{A}^{d_2}$ accumulates the weights of walks of length d_1+d_2+1 that start at v_k and end at v_l, having e_r as its (d_1+1)st edge; for each such walk, its weight is given by $1\times 1\times\cdots\times 1\times(1+\delta x)\times 1\times\cdots\times 1 = 1+\delta x$. Therefore, the perturbation of e_r affects the entry of $\mathbf{F}(\delta\mathbf{B}_r)$ only if e_r detours e_s. In other words, the entry k_{rs} of $\mathbf{K}$ is nonzero only if e_r detours e_s. Since we can re-order the edges $e_1,\ldots,e_m$ in such a way that e_r detours e_s only if $r \le s$, we can make the matrix $\mathbf{K}$ upper triangular.

It only remains to verify that the elements on the diagonal are all nonzero. Indeed e_r detours itself; but also the term in equation (12) with $d_1 = d_2 = 0$, which is $\delta\mathbf{B}_r$, has coefficient 1, while all the other terms have the coefficient $\frac{1}{d_1!d_2!}$. Even though they may have different signs, the smaller terms are not enough to cancel the leading terms.

In conclusion, $\mathbf{K}$ is upper triangular and its diagonal elements are nonzero. Hence, $\mathbf{K}$ is nonsingular, and Theorem 1 (and Corollary 1) can be applied. ■

REMARK 1. The proof of Theorem 2 only used the fact that in a DAG two distinct edges cannot both detour each other. It is easy to prove that a directed graph with this property must be a DAG.

REMARK 2. Experiments suggest that the conclusion of these theorems holds for more general classes of graphs. A proof might require a more delicate analysis.

4 The proximal gradient method

The proximal gradient (PG) method furnishes an alternative to Newton's method for solving eq. (4). We first briefly review the PG method and then discuss the incorporation of a line search. The section is concluded with some comments on convergence properties of this method. Beck [2, Chapter 10] describes the PG method, and we refer to this reference for further details.

Consider the minimization problem (4) and define the function

$$\mathbf{f}(\mathbf{A}) = \frac{1}{2}\|\exp(\mathbf{A}) - \mathbf{C}\|_F^2. \tag{13}$$

Let $\mathbf{A}_0 \in \mathbb{R}^{n\times n}$ be an arbitrary approximate solution of (4). Then the PG method produces a sequence of improved approximate solutions $\mathbf{A}_k$, $k = 1, 2, \ldots$, defined by

$$\mathbf{A}_{k+1} = \mathrm{prox}(\mathbf{A}_k - t_k\nabla_{\mathbf{A}}\mathbf{f}(\mathbf{A}_k), \mu), \quad k = 0, 1, \ldots , \tag{14}$$

where t_k and $\nabla_{\mathbf{A}}\mathbf{f}(\cdot)$ stand for the step size and search direction, respectively, and $\mathrm{prox}(\cdot)$ denotes the proximal operator, i.e.,

$$\mathrm{prox}(\mathbf{A},\mu) = \underset{\mathbf{X}\in\mathbb{R}^{n\times n}}{\arg\min}\left\{\frac{1}{2}\|\mathbf{A}-\mathbf{X}\|_F^2 + \mu\|\mathrm{vec}(\mathbf{X})\|_1\right\}; \tag{15}$$

see [2, Chapter 10]. The prox operator can be implemented with soft-thresholding; see (8).

In order to use eq. (14) to determine an approximation of the solution of (4), we have to compute the gradient of the function (13). Note that

$$\begin{aligned}\mathbf{f}(\mathbf{A}) &= \frac{1}{2}\,\mathrm{Tr}\left\{(\exp(\mathbf{A})-\mathbf{C})^T(\exp(\mathbf{A})-\mathbf{C})\right\}\\ &= \frac{1}{2}\,\mathrm{Tr}\left\{\exp(\mathbf{A})^T\exp(\mathbf{A})\right\}-\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}+\frac{1}{2}\,\mathrm{Tr}\left\{\mathbf{C}^T\mathbf{C}\right\}.\end{aligned}$$

Thus, we have to minimize

$$\frac{1}{2}\,\mathrm{Tr}\left\{\exp(\mathbf{A})^T\exp(\mathbf{A})\right\}-\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}$$

over $\mathbf{A}\in\mathbb{R}^{n\times n}$. To this end, we determine the gradients of $\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}$ and $\mathrm{Tr}\left\{\exp(\mathbf{A})^T\exp(\mathbf{A})\right\}$ with respect to $\mathbf{A}$.

THEOREM 3. *Let $\mathbf{A}=[a_{ij}]\in\mathbb{R}^{n\times n}$ and $\mathbf{C}=[c_{ij}]\in\mathbb{R}^{n\times n}$. Then*

$$\nabla_{\mathbf{A}}\left\{\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}\right\}=\sum_{\ell=1}^{\infty}\frac{1}{\ell!}\sum_{r=0}^{\ell-1}\left(\mathbf{A}^r\mathbf{C}^T\mathbf{A}^{\ell-r-1}\right)^T.$$

In the special case when $\mathbf{A}^T$ and $\mathbf{C}$ commute, we have

$$\nabla_{\mathbf{A}}\left\{\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}\right\}=\exp\left(\frac{\mathbf{A}}{2}\right)^T\cdot\mathbf{C}\cdot\exp\left(\frac{\mathbf{A}}{2}\right)^T.$$

Proof. The power series expansion

$$\exp(\mathbf{A}^T)\mathbf{C}=\sum_{\ell=0}^{\infty}\frac{(\mathbf{A}^T)^\ell\mathbf{C}}{\ell!}$$

yields

$$\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}=\sum_{\ell=0}^{\infty}\frac{1}{\ell!}\,\mathrm{Tr}\left\{(\mathbf{A}^T)^\ell\mathbf{C}\right\}.$$

For $\ell=0$, we have $\nabla_{\mathbf{A}}\,\mathrm{Tr}\{\mathbf{C}\}=\mathbf{0}$. Direct computations for $\ell=1$ give

$$\mathrm{Tr}\{\mathbf{A}^T\mathbf{C}\}=\sum_{j=1}^{n}\sum_{i=1}^{n}a_{ij}c_{ij}$$

and, therefore,

$$\nabla_{\mathbf{A}}\,\mathrm{Tr}\{\mathbf{A}^T\mathbf{C}\}=\mathbf{C}.$$

For $\ell=2$, we first compute the gradient of $\mathrm{Tr}\{(\mathbf{A}^T)^2\mathbf{C}\}$ using the cyclic permutation and product rules described in Subsection 2.2. This way we obtain

$$\nabla_{\mathbf{A}}\,\mathrm{Tr}\{(\mathbf{A}^T)^2\mathbf{C}\}=\mathbf{A}^T\mathbf{C}+\mathbf{C}\mathbf{A}^T.$$

Carrying out the computations for $\ell>2$ yields

$$\nabla_{\mathbf{A}}\,\mathrm{Tr}\left\{(\mathbf{A}^T)^\ell\mathbf{C}\right\}=\sum_{r=0}^{\ell-1}\left(\mathbf{A}^r\mathbf{C}^T\mathbf{A}^{\ell-r-1}\right)^T$$

and, therefore,

$$\nabla_{\mathbf{A}}\left\{\mathrm{Tr}\left\{\exp(\mathbf{A})^T\mathbf{C}\right\}\right\}=\sum_{\ell=1}^{\infty}\frac{1}{\ell!}\sum_{r=0}^{\ell-1}\left(\mathbf{A}^r\mathbf{C}^T\mathbf{A}^{\ell-r-1}\right)^T.$$

When the matrices $\mathbf{A}^T$ and $\mathbf{C}$ commute, the above expression simplifies to

$$\nabla_{\mathbf{A}}\left\{ \operatorname{Tr}\left\{ \exp(\mathbf{A})^T\mathbf{C}\right\}\right\} = \exp\left(\frac{\mathbf{A}}{2}\right)^T \cdot \mathbf{C} \cdot \exp\left(\frac{\mathbf{A}}{2}\right)^T.$$

■

THEOREM 4. *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$. *Then*

$$\nabla_{\mathbf{A}}\left\{ \operatorname{Tr}\left\{ \exp(\mathbf{A})^T \exp(\mathbf{A})\right\}\right\} = 2\sum_{\ell=1}^{\infty}\frac{1}{\ell!}\sum_{r=0}^{\ell-1}\left(\mathbf{A}^r \exp(\mathbf{A})^T\mathbf{A}^{\ell-r-1}\right)^T. \tag{16}$$

In the special case when $\mathbf{A}$ *is symmetric, this expression simplifies to*

$$\nabla_{\mathbf{A}}\left\{ \operatorname{Tr}\left\{ \exp(\mathbf{A})^T \exp(\mathbf{A})\right\}\right\} = 2\exp(2\mathbf{A}). \tag{17}$$

Proof. Application of the product rule introduced in Subsection 2.2 and some straightforward but tedious computations yield (16). When $\mathbf{A}$ is symmetric, expression (16) simplifies to (17). ■

In view of the above results, proximal gradient iteration can be expressed as

$$\begin{cases} \mathbf{G}_k = \displaystyle\sum_{\ell=1}^{\infty}\frac{1}{\ell!}\sum_{r=0}^{\ell-1}\left(\mathbf{A}_k^r\Big(\exp(\mathbf{A}_k)^T - \mathbf{C}^T\Big)\mathbf{A}_k^{\ell-r-1}\right)^T, \\ \tilde{\mathbf{A}}_{k+1} = \mathbf{A}_k - t_k\mathbf{G}_k, \\ \mathbf{A}_{k+1} = \mathbf{S}_\mu(\tilde{\mathbf{A}}_{k+1}). \end{cases} \tag{18}$$

When $\mathbf{A}_{\text{true}}$ is symmetric, the matrix (3) is close to symmetric if $\|\boldsymbol{N}\|_F \ll \|\exp(\mathbf{A})\|_F$. Moreover, if we choose the initial approximate solution $\mathbf{A}_0$ to be symmetric and to commute with $\mathbf{C}$, then eq. (18) can be written as

$$\begin{cases} \mathbf{G}_k \approx \exp\left(\frac{\mathbf{A}_k}{2}\right)\Big(\exp(\mathbf{A}_k) - \mathbf{C}\Big)\exp\left(\frac{\mathbf{A}_k}{2}\right), \\ \tilde{\mathbf{A}}_{k+1} = \mathbf{A}_k - t_k\mathbf{G}_k, \\ \mathbf{A}_{k+1} = \mathbf{S}_\mu(\tilde{\mathbf{A}}_{k+1}). \end{cases} \tag{19}$$

We use Algorithm 4.0.1 below to approximate the gradient of the trace $\operatorname{Tr}\{\exp(\mathbf{A})^T\mathbf{Y}\}$. Here $\mathbf{Y} = \mathbf{C}$ or $\mathbf{Y} = \exp(\mathbf{A})$. Define the relative difference

$$r := \|\mathbf{G}_{\text{new}} - \mathbf{G}_{\text{old}}\|_F / \|\mathbf{G}_{\text{old}}\|_F, \tag{20}$$

where $\mathbf{G}_{\text{old}}$ and $\mathbf{G}_{\text{new}}$ denote approximations determined by carrying out m and $m+1$ steps, respectively, of an iterative method.

Algorithm 4.0.1 Compute an approximation of $\nabla_{\mathbf{A}}\{\mathrm{Tr}\{\exp(\mathbf{A})^T\mathbf{Y}\}\}$

Input: $\mathbf{A}, \mathbf{Y} \in \mathbb{R}^{n\times n}$

Output: $\mathbf{G}$: approximation of the $\nabla_{\mathbf{A}}\{\mathrm{Tr}\{\exp(\mathbf{A})^T\mathbf{Y}\}\}$

$\mathbf{G}_{\text{old}} = \mathbf{0}$;
$\mathbf{G}_{\text{new}} = \mathbf{Y}$;
$k = 1$;
while $r > 10^{-4}$ **do** (r is defined by (20))
  $k = k + 1$;
  $\mathbf{G}_{\text{old}} = \mathbf{G}_{\text{new}}$;
  for $i = 1, \ldots, k$ **do**
    $\mathbf{G}_{\text{new}} = \mathbf{G}_{\text{new}} + \frac{1}{k!}\left(\mathbf{A}^{i-1}\mathbf{Y}^T\mathbf{A}^{k-i}\right)^T$;
  end for
end while

Similarly as the NST method, the PG method might diverge if the step sizes t are too large. We therefore equip Algorithm 4.0.1 with a backtracking line search method. The backtracking is based on two parameters α and β, where $0 < \alpha \leq 0.5$ and $0 < \beta < 1$. Boyd and Vandenberghe [4, Section 9.2] suggest to choose $\beta \in [0.1, 0.8]$. We let $\alpha = 0.5$. The general procedure is to start with $t_k = 1$, and then update t_k by βt_k while

$$\mathbf{f}(\mathbf{A}_k - t_k\nabla_{\mathbf{A}}\mathbf{f}(\mathbf{A}_k)) > \mathbf{f}(\mathbf{A}_k) - 0.5 \cdot t_k\|\nabla_{\mathbf{A}}\mathbf{f}(\mathbf{A}_k)\|^2.$$

Algorithm 4.0.2 describes the PG method with backtracking line search (PG-LS).

Algorithm 4.0.2 PG-LS

Input: $\mathbf{C}$, μ, and $\beta \in [0.1, 0.8]$

Output: $\mathbf{A}_{k+1}$: the approximation of $\mathbf{A}_{\text{true}}$

Starting with $\mathbf{A}_0$
for $k = 1, 2, \ldots$ until convergence **do**
  Use Algorithm 4.0.1 to estimate the gradient $\mathbf{G}_k$ (If $\mathbf{A}_0 = \mathbf{A}_0^T$ and $\mathbf{A}_0\mathbf{A} = \mathbf{A}\mathbf{A}_0$, then use the first row of eq. (19) to estimate $\mathbf{G}_k$.)
  $t_k = 1$
  while $\mathbf{f}(\mathbf{A}_k - t_k\mathbf{G}_k) > \mathbf{f}(\mathbf{A}_k) - 0.5 \cdot t_k\|\mathbf{G}_k\|^2$ **do**
    $t_k = \beta t_k$
  end while
  $\tilde{\mathbf{A}}_{k+1} = \mathbf{A}_k - t_k\mathbf{G}_k$
  $\mathbf{A}_{k+1} = \mathbf{S}_\mu(\tilde{\mathbf{A}}_{k+1})$
end for

We comment on the convergence of the PG method. Beck [2, Chapter 10] provides a convergence analysis of this method when applied to minimize composite functions of the form

$$\underset{\mathbf{A}\in\mathbb{R}^{n\times n}}{\arg\min}\{\mathbf{f}(\mathbf{A}) + \mu\mathbf{h}(\mathbf{A})\}, \tag{21}$$

where $\mathbf{f}$ is a smooth function that has a Lipschitz continuous gradient, $\mathbf{h}$ is a proper closed convex function that is not required to be smooth, and $\mu > 0$ is a regularization parameter. When the function $\mathbf{f}$ is not

convex, Beck shows convergence of the norm of the gradient mapping to zero and that all limit points of the sequence $\{\mathbf{A}_k\}$ are stationary points of problem (4). When $\mathbf{f}$ is convex, the rate of convergence of the iterates generated with the PG-LS method is $O(1/k)$, where k denotes the number of iterations.

We conclude this section by noting that it may be attractive to combine the NST-LS and PG-LS methods into a hybrid method that first applies the PG-LS method to determine a fairly accurate approximate solution of (4), and then improves this approximate solution by applying the NST-LS method. We refer to this hybrid method as PGNST-LS. Its performance is illustrated in the following section.

5 Numerical experiments

This section presents some computed examples with small and large networks to illustrate the performance of the algorithms described in Sections 3 and 4, as well of some other related methods. The computations are carried out using MATLAB R2021a on a laptop computer with the Mac OS Big Sur 11.6 operating system, an M1 chip, and 16 GB of memory. The Celegansneural data set used in Example 5.3 and Western US power data set used in Example 5.4, are available for download from the web site [5]; the matrices in Example 5.2 and the Appendix are generated with the CONTEST toolbox [17].

We terminate the iterations with the Algorithms 3.1.1 and 4.0.2 as soon as two consecutive iterates are sufficiently close, i.e., as soon as

$$\frac{\|\mathbf{A}_{k+1} - \mathbf{A}_k\|_F}{\|\mathbf{A}_k\|_F} < 10^{-4}, \tag{22}$$

where $\mathbf{A}_k$ and $\mathbf{A}_{k+1}$ denote approximations determined by carrying out k and $k+1$ steps, respectively, of the algorithms. When using the hybrid PGNST-LS method mentioned at the end of the previous section, we first determine an approximate solution with the PG-LS method by terminating Algorithm 4.0.2 as soon as

$$\frac{\|\mathbf{G}_{k+1} - \mathbf{G}_k\|_F}{\|\mathbf{G}_k\|_F} < 0.1, \tag{23}$$

where $\mathbf{G}_k$ and $\mathbf{G}_{k+1}$ denote gradient approximations obtained by carrying out k and $k+1$ steps, respectively. Then we use the corresponding computed approximate solution as the initial initial approximate solution for the NST-LS method. We terminate the computations with the latter method as soon as the stopping criterion (22) is satisfied.

The norm of the communicability matrix $\mathbf{C}$ generally is much larger than the norm of the associated adjacency matrix $\mathbf{A}$. Therefore, the first term in the minimization problem (4) dominates and the second term has little influence on the computed solution. To remedy this fact, we replace the matrix $\mathbf{C}$ by the scaled matrix

$$\tilde{\mathbf{C}} = \exp\left(\frac{\log(\mathbf{C})}{\log(\rho(\mathbf{C}))}\right), \tag{24}$$

where $\rho(\mathbf{C})$ denotes the Perron root of $\mathbf{C}$. This scaling has proved to perform well for both small and large matrices $\mathbf{C}$. The computed solution is scaled correspondingly. We note that the Perron root can easily be computed by the MATLAB functions eig or eigs for small and large matrices, respectively.

The computation of $\log(\mathbf{C})$ in (24) requires some care for certain matrices $\mathbf{C}$. If $\mathbf{C} = \exp(\mathbf{A}_{\text{true}})$ is computed in exact arithmetic, then, of course, $\log(\mathbf{C}) = \mathbf{A}_{\text{true}}$. However, due to the error term $\mathbf{N}$ in (3) and round-off errors introduced during the evaluation of $\log(\mathbf{C})$, the computed matrix $\log(\mathbf{C})$ may be quite different from $\mathbf{A}_{\text{true}}$ and might contain complex-valued entries. We replace any non-real entry of $\log(\mathbf{C})$ by the closest real number (i.e., by its real part), and use this approximation of $\mathbf{C}$ in the right-hand side of (24). Numerous computed examples indicate that this replacement often is not required.

We will refer to the exact adjacency matrix of the network as the "true solution", and denote the approximate solution computed by the NST-LS method, as the "NST-LS solution". The computed solution depends on the choice of the regularization parameter $\mu > 0$. A large value of μ results in a network with few edges, in fact the network might not be connected when μ is too large; a small value of $\mu > 0$ yields a network with many edges. The desired number of edges in the computed graph depends on the application and has to be decided by the user. In the computed examples reported in this section, we calculate solutions with μ-values in the interval $[0.015, 0.55]$ with a granularity of 1000 points. The iterations are terminated when the error defined by (25) is smaller than a user-specified tolerance, which in all the examples of this paper is set to 10^{-3} or when 1000 iterations have been carried out.

We seek to determine an adjacency matrix $\mathbf{A}$ with entries zero or one. However, the computed solution of (4) has nonvanishing entries that are different from one. We set entries of the solution $\mathbf{A}$ of (4) that are greater than μ to one, and other nonvanishing entries to zero. This defines the matrix $\mathbf{A}_{\text{projected}}$. We define the error

$$\text{Error} = \frac{\|\text{vec}(\mathbf{A}_{\text{projected}} - \mathbf{A}_{\text{true}})\|_1}{n^2}, \tag{25}$$

where the denominator equals the number of entries of $\mathbf{A}_{\text{true}}$.

Among the μ-values used in our examples, we denote the value that yields a solution $\mathbf{A}$ that is closest to $\mathbf{A}_{\text{true}}$ by μ_{optimal}. We will compare the networks determined with $\mu = \mu_{\text{optimal}}$ with networks obtained for some other choices of μ.

5.1 A small network

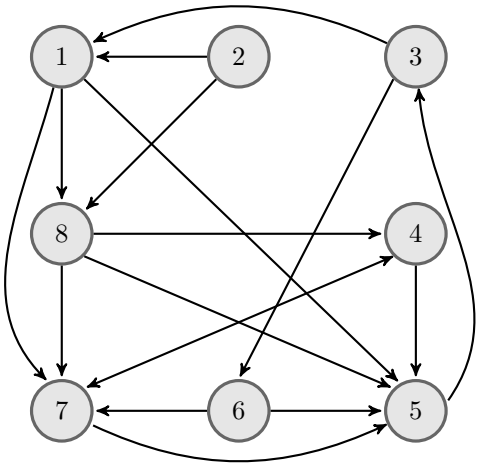

Figure 1: Graph for Example 5.1.

EXAMPLE 5.1. Consider the directed graph with 8 nodes shown in Figure 1. The associated communicability matrix, which is corrupted by 1% white Gaussian noise is given by

$$\mathbf{C} = \mathbf{C}_{\text{true}} + \mathbf{N} = \begin{bmatrix} 1.3050 & -0.0081 & 1.0200 & 1.2673 & 2.8164 & 0.2746 & 2.0381 & 1.0685 \\ 1.1383 & 1.0025 & 0.5311 & 1.1118 & 1.9821 & 0.1167 & 1.5287 & 1.4951 \\ 1.1176 & -0.0141 & 1.5113 & 0.5693 & 1.7592 & 1.1143 & 1.3505 & 0.5239 \\ 0.2311 & 0.0025 & 0.7273 & 1.5807 & 1.8514 & 0.2119 & 1.3090 & 0.0603 \\ 0.5158 & -0.0019 & 1.1283 & 0.1528 & 1.5321 & 0.5248 & 0.4174 & 0.1917 \\ 0.2324 & 0.0140 & 0.7648 & 0.5645 & 1.8720 & 1.2355 & 1.3042 & 0.0327 \\ 0.2240 & -0.0056 & 0.7518 & 1.1913 & 1.8575 & 0.2399 & 1.6510 & 0.0525 \\ 0.2815 & 0.0074 & 0.9921 & 1.7642 & 2.6028 & 0.2890 & 1.8480 & 1.0654 \end{bmatrix},$$

where $\mathbf{N}$ denotes the "noise matrix." We rescale the matrix $\mathbf{C}$ according to (24) to obtain the rescaled matrix $\tilde{\mathbf{C}}$.

	μ	Number of iterations	CPU time	Error
NST-LS soln.	$\mu_{\text{optimal}} = 0.015$	6	0.003	0
NST-LS soln.	$\mu = 0.0025$	6	0.002	3.4375×10^{-1}
NST-LS soln.	$\mu = 0.25$	8	0.004	0
PG-LS soln.	$\mu_{\text{optimal}} = 0.015$	59	0.117	0
PG-LS soln.	$\mu = 0.0025$	62	0.052	4.6875×10^{-2}
PG-LS soln.	$\mu = 0.25$	38	0.022	1.8750×10^{-1}
PGNST-LS soln.	$\mu_{\text{optimal}} = 0.015$	13 (PG)+3 (NST)	0.016	0
PGNST-LS soln.	$\mu = 0.0025$	26 (PG)+3 (NST)	0.026	3.4375×10^{-1}
PGNST-LS soln.	$\mu = 0.25$	5 (PG)+7 (NST)	0.007	0

Table 1: Example 5.1: Comparison of the approximate solutions determined by the NST-LS, PG-LS, and PGNST-LS algorithms in terms of the values of μ, the number of iterations, CPU time required (in seconds), and the error (25).

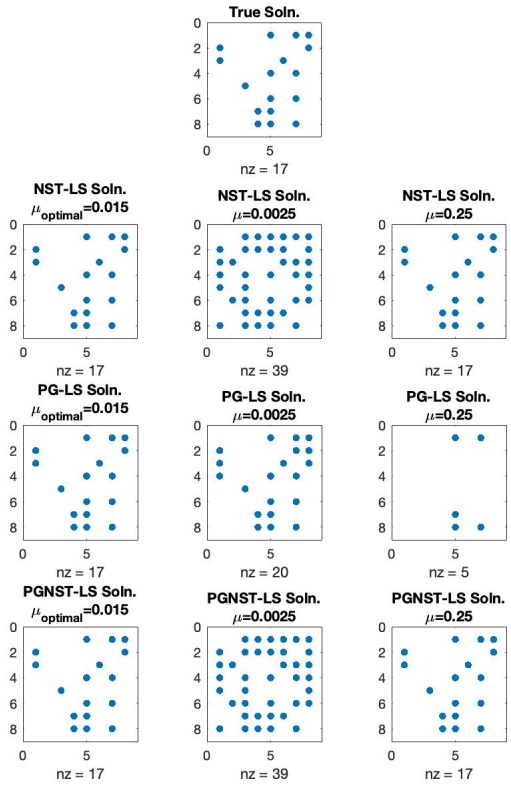

Figure 2: Example 5.1: Comparison of the true and the approximate solutions determined by the NST-LS, PG-LS, and PGNST-LS algorithms for several μ-values. The heading of each subplot shows the μ-value used and the bottom indicates the number of nonzero entries (which have been projected to 1).

Table 1 and Figure 2 show all the methods to determine the true solution when $\mu_{\text{optimal}} = 0.015$. A much smaller value of μ gives a computed solution with more edges than in $\mathbf{A}_{\text{true}}$, while a much larger value of μ gives a computed solution with fewer edges than in $\mathbf{A}_{\text{true}}$. Clearly, the choice of the regularization parameter $\mu > 0$ is important.

5.2 Three additional methods that may be applied to recover a network

5.2.1 The proximal Newton method

Lee et al. [13] introduced the *proximal Newton method* for the minimization of expressions of the form (4). Let $\mathbf{A}_0 \in \mathbb{R}^{n\times n}$ denote the initial approximate solution. Then the proximal Newton method determines a sequence of adjacency matrices $\{\mathbf{A}_k\}_{k=1,2,\ldots}$ as follows:

$$\begin{cases} \mathbf{Y}_k = \text{prox}\Big(\mathbf{A}_{k-1} - \boldsymbol{H}_{k-1}^{-1}\nabla\mathbf{f}(\mathbf{A}_{k-1}), \mu\Big), & \text{where} \quad \boldsymbol{H}_{k-1} = \nabla^2\mathbf{f}(\mathbf{A}_{k-1}), \\ \mathbf{A}_k = \mathbf{A}_{k-1} + t_k(\mathbf{Y}_k - \mathbf{A}_{k-1}). \end{cases}$$

Here prox$(\cdot)$ is the proximal operator (15), and the step size t_k is determined by backtracking. To carry out the computations with the proximal Newton method, we have to reshape the matrices $\mathbf{A}_{k-1}$ and $\nabla\mathbf{f}(\mathbf{A}_{k-1}) \in \mathbb{R}^{n\times n}$ into the vectors $\text{vec}(\mathbf{A}_{k-1}) \in \mathbb{R}^{n^2}$ and $\text{vec}(\nabla\mathbf{f}(\mathbf{A}_{k-1})) \in \mathbb{R}^{n^2}$, respectively. Then we have $\boldsymbol{H}_{k-1} = \nabla(\text{vec}(\nabla\mathbf{f}(\mathbf{A}))) \in \mathbb{R}^{n^2\times n^2}$. To determine an approximation of $\boldsymbol{H}_{k-1}$, we evaluate the Fréchet derivative of $\nabla\text{vec}(\mathbf{f})$ with respect to all the elements a_{ij}, for $i, j = 1, , 2 \ldots, n$. This may be expensive. Denote the computed approximation by $\text{vec}(\mathbf{A}_k)$. The vector obtained is then reshaped into a matrix. We refer to the proximal Newton method with backtracking as the PN-LS method. We remark that for large-scale networks, in the situation when $\mathbf{A}$ is symmetric, a Krylov method introduced by Kandolf et al. [12, Algorithm 2] can be applied to approximate the Fréchet derivative.

Recall that

$$\mathbf{G} = \nabla\mathbf{f}(\mathbf{A}_k) \approx \underbrace{\exp\left(\frac{\mathbf{A}_k}{2}\right)}_{\mathbf{f}_1}\underbrace{\left(\exp(\mathbf{A}_k) - \mathbf{C}\right)}_{\mathbf{f}_2}\underbrace{\exp\left(\frac{\mathbf{A}_k}{2}\right)}_{\mathbf{f}_1}.$$

To approximate the Fréchet derivative of $\mathbf{G}$ in the direction $\mathbf{E}$, we use the product rule

$$\begin{aligned}
\mathbf{G}(\mathbf{A}+\mathbf{E}) &= \mathbf{f}_1(\mathbf{A}+\mathbf{E})\mathbf{f}_2(\mathbf{A}+\mathbf{E})\mathbf{f}_1(\mathbf{A}+\mathbf{E}) \\
&= \Big(\mathbf{f}_1(\mathbf{A}) + \mathbf{L}_{\mathbf{f}_1}(\mathbf{A},\mathbf{E}) + o(\|\mathbf{E}\|_p)\Big)\Big(\mathbf{f}_2(\mathbf{A}) + \mathbf{L}_{\mathbf{f}_2}(\mathbf{A},\mathbf{E}) + o(\|\mathbf{E}\|_p)\Big)\Big(\mathbf{f}_1(\mathbf{A}) + \mathbf{L}_{\mathbf{f}_1}(\mathbf{A},\mathbf{E}) + o(\|\mathbf{E}\|_p)\Big) \\
&= \mathbf{f}_1(\mathbf{A})\mathbf{f}_2(\mathbf{A})\mathbf{f}_1(\mathbf{A}) + \mathbf{L}_{\mathbf{f}_1}(\mathbf{A},\mathbf{E})\mathbf{f}_2(\mathbf{A})\mathbf{f}_1(\mathbf{A}) + \mathbf{f}_1(\mathbf{A})\mathbf{L}_{\mathbf{f}_2}(\mathbf{A},\mathbf{E})\mathbf{f}_1(\mathbf{A}) + \mathbf{f}_1(\mathbf{A})\mathbf{f}_2(\mathbf{A})\mathbf{L}_{\mathbf{f}_1}(\mathbf{A},\mathbf{E}) + o(\|\mathbf{E}\|_p).
\end{aligned}$$

This expression implies that

$$\mathbf{L}_{\mathbf{G}}(\mathbf{A},\mathbf{E}) = \mathbf{L}_{\mathbf{f}_1}\mathbf{f}_2\mathbf{f}_1 + \mathbf{f}_1\mathbf{L}_{\mathbf{f}_2}\mathbf{f}_1 + \mathbf{f}_1\mathbf{f}_2\mathbf{L}_{\mathbf{f}_1}.$$

We refer to this method as the PNKKRS-LS method.

5.2.2 Proximal gradient methods with alternative thresholding operators

Liu and Barber [14] describe how the proximal gradient method can be applied with ℓ_q-thresholding. They also introduce the concept of a proximal gradient method with reciprocal thresholding. Both these thresholding methods are aimed at minimizing composite functions of the form (4). Liu and Barber [14] consider minimization problems

$$\arg\min_{\mathbf{A}\in\mathbb{R}^{n\times n}}\{\mathbf{f}(\mathbf{A}) + \mu\mathbf{h}(\mathbf{A})\},$$

where $\mathbf{f}$ is given by (13), $\mathbf{h}$ is a regularization operator, and $\mu > 0$ is a regularization parameter; cf. (21). Liu and Barker [14] choose

$$\mathbf{h}(\mathbf{A}) = \|\text{vec}(\mathbf{A})\|_{2/3}.$$

This choice can be implemented as

$$\mathbf{A}_{k+1} = \boldsymbol{\Psi}^{\ell_{2/3}}\Big(\mathbf{A}_k - t_k\nabla\mathbf{f}(\mathbf{A}_k)\Big),$$

where $t_k > 0$ is the step size and $\boldsymbol{\Psi}(\cdot)$ represents the generalized thresholding operator. It can be written in closed form

$$\boldsymbol{\Psi}^{\ell_{2/3}}(a_{ij}) = \begin{cases} \text{sign}(a_{ij})\cdot\left(\dfrac{|z| + \sqrt{\frac{2|a_{ij}|}{|z|} - |z|^2}}{2}\right)^3, & |a_{ij}| > \dfrac{\sqrt[4]{48}}{3}\mu^{3/4} \\ 0, & \text{otherwise,} \end{cases}$$

for $i, j = 1, 2, \ldots, n$, where

$$z = \frac{2}{\sqrt{3}}\mu^{1/4}\left(\cosh\left(\frac{1}{3}\,\text{arccosh}\left(\frac{27}{16}\mu^{-3/2}(a_{ij})^2\right)\right)\right)^{1/2};$$

see [18]. We refer to this solution approach as the PGLQT-LS method.

The reciprocal thresholding operator, defined by Liu and Barber [14], is given by

$$\mathbf{\Psi}_S^{RT}(a_{ij}) = \begin{cases} \text{sign}(a_{ij}) \cdot \left(\frac{1}{2}|a_{ij}| + \frac{1}{2}\sqrt{|a_{ij}|^2 - \tau^2} \right), & \text{if } a_{ij} \in S \\ 0, & \text{otherwise,} \end{cases}$$

where S represents the set consisting of the s largest entries of the matrix $\mathbf{A}$, and τ denotes the thresholding level such that $\tau = \max_{a_{ij} \notin S} |a_{ij}|$. This approach is independent of the choice of μ. However, we will use the value μ_{optimal} to determine the projection of the computed solution onto the matrix with entries 0 and 1. We refer to this scheme as the PGRT-LS method. The following example compares the methods outlined above to the ones described in the previous sections of this paper.

EXAMPLE 5.2. Consider the *preferential attachment* model, whose adjacency matrix $\mathbf{A}_{\text{true}}$ can be generated by code $\mathsf{pref}(n, d)$ in the CONTEST toolbox, where n denotes the dimension of the matrix, and d denotes the minimal node degree. We choose $n = 80$ and $d = 4$, and obtain a symmetric adjacency matrix $\mathbf{A}_{\text{true}} \in \mathbb{R}^{80\times 80}$ with 554 nonzero elements. The communicability matrix $\mathbf{C}$ is obtained by adding 1% white Gaussian noise to the true communicability matrix $\mathbf{C}_{\text{true}} = \exp(\mathbf{A}_{\text{true}})$. We rescale $\mathbf{C}$ according to (24) to obtain the scaled matrix $\tilde{\mathbf{C}}$. The initial approximate solution for both the PN-LS and PNKKRS-LS methods is $\mathbf{A}_0 = \tilde{\mathbf{C}}$. Furthermore, we adjust μ_{optimal} by increments and decrements of 10% to assess the significance of determining μ close to μ_{optimal}.

Table 2 displays the μ-values, the number of iterations, the required CPU time (in seconds), and the error in the computed approximate solution for the methods in our comparison. The table shows the PGLQT-LS method to determine more accurate approximate solutions than the approximate solutions determined with soft- and reciprocal thresholding, but the former method requires many more iterations to satisfy the stopping criterion (22) than the NST-LS method. Moreover, Table 2 shows the PG-LS method to determine approximate solutions with smaller error than the NST-LS method for several values of μ. However, it is important to note that this method demands many more iterations to satisfy the stopping criterion (22) than the NST-LS method. Among the four Newton-type methods, the PN-LS method attains the smallest error for the best value of μ used in the computations. However, this method may be impractical for large values of n due to the high computational cost associated with the evaluation of the Hessian matrix. We also note that the CPU time for the PNKKRS-LS method is not very different from that for the PN-LS method because the Arnoldi algorithm needs to be applied four times for each loop in the PNKKRS-LS method. Finally, we remark that the NST-LS method yields slightly less accurate results, but at a much lower computational cost than the other three Newton-type methods. Figure 3 illustrates that the approximations are close to the true solution when μ is well chosen, and the approximations become less sparse than the true solution when μ is small. Table 2 shows the NST-LS algorithm to require the least computing time of the six algorithms in our comparison.

	μ	Number of iterations	Elapsed time (in seconds)	Error
PGRT-LS soln.	$\mu_{\text{optimal1}} = 0.015$	1000	23.604	1.5625×10^{-4}
NST-LS soln.	$\mu_{\text{optimal2}} = 0.02464$	8	0.582	9.3750×10^{-4}
NST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal2}}$	7	0.033	1.0938×10^{-3}
NST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal2}}$	8	0.041	1.0938×10^{-3}
PG-LS soln.	$\mu_{\text{optimal3}} = 0.015$	52	3.080	6.2500×10^{-4}
PG-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal3}}$	56	3.355	6.2500×10^{-4}
PG-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal3}}$	51	2.821	4.6875×10^{-4}
PGNST-LS soln.	$\mu_{\text{optimal4}} = 0.02464$	29 (PG)+6 (NST)	37.941	9.3750×10^{-4}
PGNST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal4}}$	31 (PG)+6 (NST)	1.740	1.0938×10^{-3}
PGNST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal4}}$	27 (PG)+6 (NST)	1.326	1.0938×10^{-3}
PN-LS soln.	$\mu_{\text{optimal5}} = 0.015536$	12	493.368	6.2500×10^{-4}
PN-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal5}}$	8	192.766	4.3750×10^{-3}
PN-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal5}}$	7	165.476	1.5625×10^{-3}
PNKKRS-LS soln.	$\mu_{\text{optimal6}} = 0.015$	7	215.836	9.3750×10^{-4}
PNKKRS-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal6}}$	7	214.855	6.0938×10^{-3}
PNKKRS-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal6}}$	6	178.844	1.5625×10^{-3}
PGLQT-LS soln.	$\mu_{\text{optimal7}} = 0.015$	77	6.427	0
PGLQT-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal7}}$	81	7.465	0
PGLQT-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal7}}$	74	5.902	4.6875×10^{-4}

Table 2: Example 5.2: Comparison of several algorithms for a few values of μ in terms of the number of iterations, required CPU time (in seconds), and the error (25).

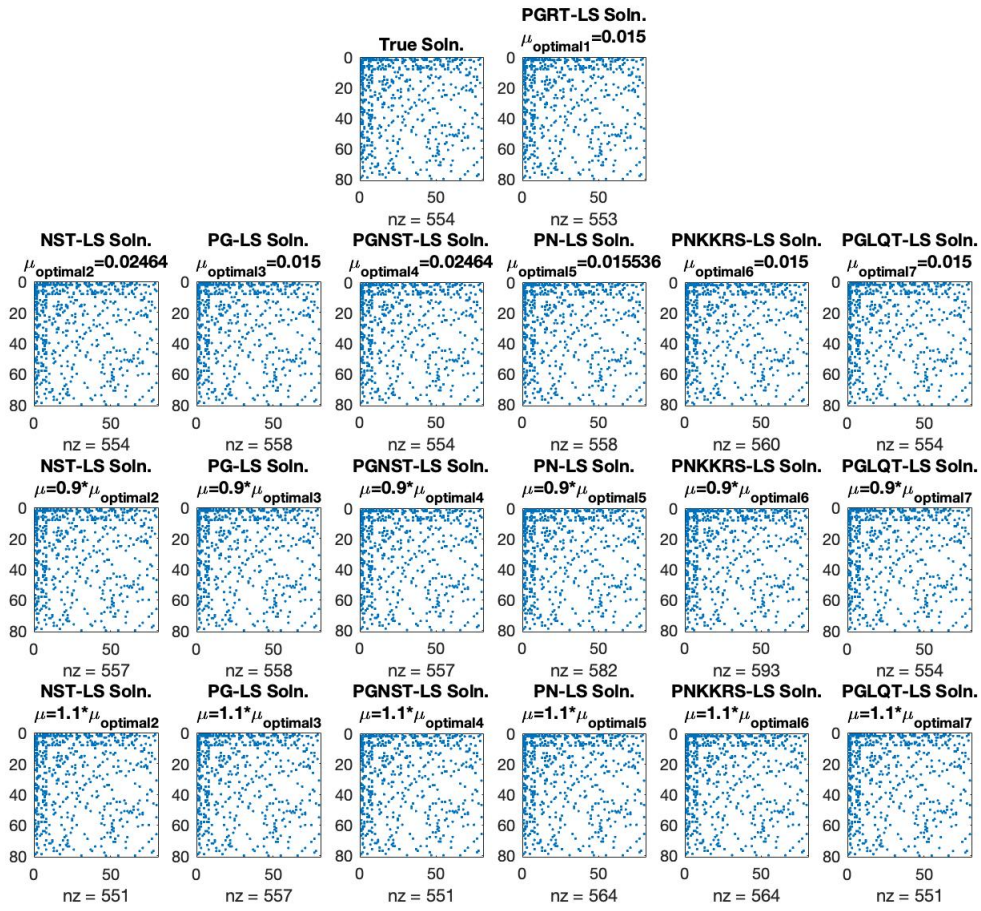

Figure 3: Example 5.2: Comparison of the true and approximate solutions for several methods and μ-values. The heading of each subplot shows the μ-value and the bottom indicates the number of ones in each solution.

5.3 Large-scale networks

EXAMPLE 5.3. The network Celegansneural is represented by a weighted directed graph with 297 nodes that corresponds to the number of neurons in the C.elegans worm. Directed edges represent synaptic connections between neurons. We are interested in inferring which nodes are connected directly rather than indirectly. We convert the weighted graph to an unweighted one by replacing the nonzero weights of the edges by one. Additionally, we add 1% white Gaussian noise to $\mathbf{C}_{\text{true}}$ to obtain $\mathbf{C}$, and then rescale $\mathbf{C}$ by eq. (24) to get $\tilde{\mathbf{C}}$. We adjust μ_{optimal} by increments and decrements of 10% to assess the sensitivity of the solution to the value of μ.

The μ-values, the number of iterations, the CPU times (in seconds) required for computing the results with the different methods, and error in the computed approximate solutions are reported in Table 3. The PG-LS method produces results with a fairly small error for several choices of μ, but this method requires many more iterations to satisfy the stopping criterion (22) than the NST-LS method. Figure 4 shows that when μ is (too) small, the computed solutions are denser than the true solution. When employing the NST-

LS method, a 10% increase in the μ_{optimal} also gives a solution that is close to $\mathbf{A}_{\text{true}}$. We remark that the sensitivity of the solution of (4) to the value of μ depends on the network at hand.

	μ	Num. of iterations	Elapsed Time (in seconds)	Error
NST-LS soln.	$\mu_{\text{optimal1}} = 0.018749$	7	3.854	8.7293×10^{-4}
NST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal1}}$	7	0.423	1.3944×10^{-3}
NST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal1}}$	7	0.483	4.9882×10^{-4}
PG-LS soln.	$\mu_{\text{optimal2}} = 0.015$	106	377	2.4941×10^{-4}
PG-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal2}}$	121	442	2.6074×10^{-4}
PG-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal2}}$	95	312	2.6074×10^{-4}
PGNST-LS soln.	$\mu_{\text{optimal3}} = 0.018749$	25 (PG)+8 (NST)	1739	8.7293×10^{-4}
PGNST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal3}}$	24 (PG)+8 (NST)	213	1.3944×10^{-3}
PGNST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal3}}$	27 (PG)+9 (NST)	301	4.9882×10^{-4}

Table 3: Example 5.3: Comparison of several algorithms in terms of different values of μ, the number of iterations, required CPU time (in seconds), and the error (25).

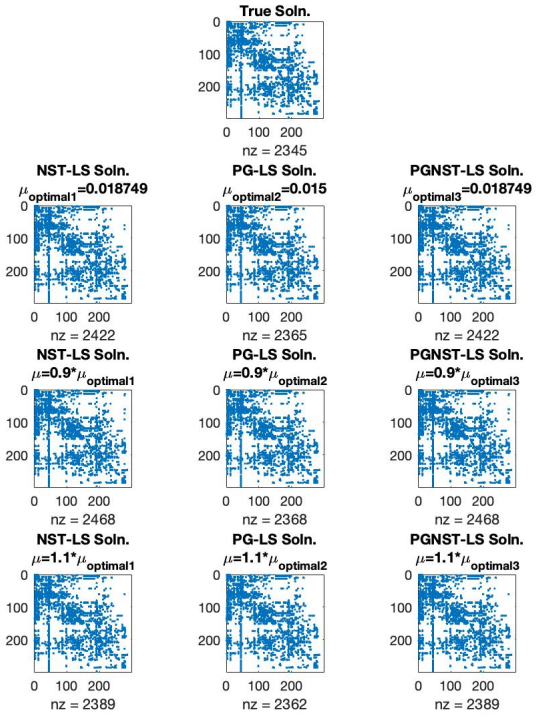

Figure 4: Example 5.3: Comparison of the true and computed approximate solutions by several algorithms for different μ-values. The heading of each subplot shows the corresponding μ-value and the bottom indicates the number of ones in each solution.

EXAMPLE 5.4. We consider the Western US power network, which is represented by an unweighted undirected graph comprising 1454 nodes and 5300 edges. Every node represents a power plant and the edges represent transmission lines. We introduce 0.5% white Gaussian noise to the communication matrix $\mathbf{C}_{\text{true}}$. This gives the noise-contaminated communication matrix $\mathbf{C}$. The latter matrix is rescaled to obtain the matrix $\tilde{\mathbf{C}}$; see (24). We determine the regularization parameter μ by modifying μ_{optimal} in increments or decrements of 10% to assess sensitivity of the computed solution to changes in the value of μ_{optimal}.

For this example, the gradient approximations in the stopping criterion (23) are large. As a result, the hybrid PGNST-LS method requires many steps to satisfy this criterion. This makes the application of this method expensive. We therefore increase the right-hand side of (23) to 0.15. Moreover, the norm of the approximate solution $\mathbf{A}_{\text{temp}}$ obtained with the PG-LS method is large. Therefore, the NST-LS method fails to converge. To circumvent this difficulty, we rescale $\mathbf{A}_{\text{temp}}$ as

$$\tilde{\mathbf{A}} = \frac{\mathbf{A}_{\text{temp}}}{\|\mathbf{A}_{\text{temp}}\|}$$

and use $\tilde{\mathbf{A}}$ as the initial approximate solution for the method of NST-LS. Table 4 shows the μ-values, the number of iterations, the CPU times (in seconds) needed by the different methods, as well as the error in the computed approximate solutions. The Newton-type methods can be seen to determine the true solution for several values of μ, while the approximate solutions computed with the PG-LS methods yield relatively large errors, because the computed solutions have fewer edges than $\mathbf{A}_{\text{true}}$. Figure 5 shows a graphical comparison.

	μ	Num. of iterations	Elapsed Time (in seconds)	Error
NST-LS soln.	$\mu_{\text{optimal1}} = 0.015$	5	13.45	0
NST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal1}}$	5	13.94	0
NST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal1}}$	6	17.10	0
PG-LS soln.	$\mu_{\text{optimal2}} = 0.015$	1000	3.10 hours	6.5276×10^{-5}
PG-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal2}}$	1000	3.13 hours	6.8114×10^{-5}
PG-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal2}}$	1000	3.01 hours	4.8247×10^{-5}
PGNST-LS soln.	$\mu_{\text{optimal3}} = 0.015$	3 (PG)+5 (NST)	42.87	0
PGNST-LS soln.	$\mu = 0.9 \cdot \mu_{\text{optimal3}}$	3 (PG)+5 (NST)	42.84	0
PGNST-LS soln.	$\mu = 1.1 \cdot \mu_{\text{optimal3}}$	3 (PG)+5 (NST)	42.93	0

Table 4: Example 5.4: Comparison of several algorithms in terms of different values of μ, the number of iterations, required CPU time (in seconds), and the error (25).

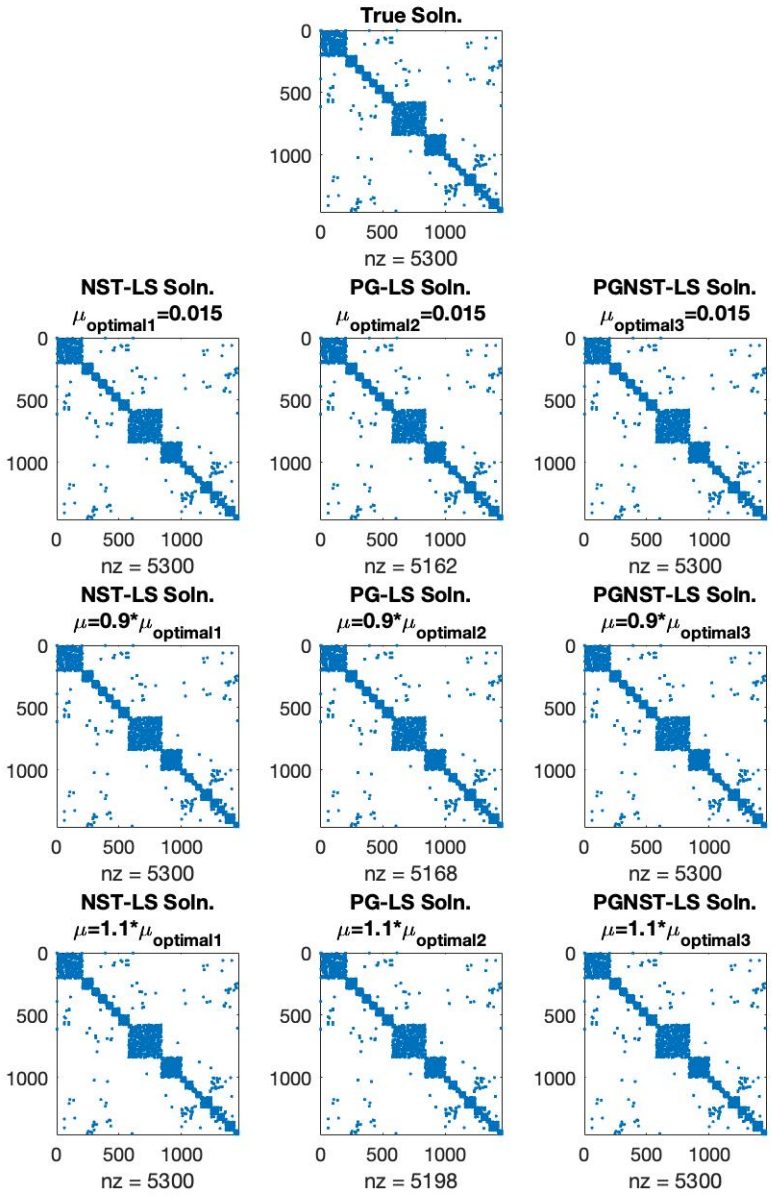

Figure 5: Example 5.4: Comparison of the true and computed approximate solutions by several algorithms for different μ-values. The heading of each subplot shows the corresponding μ-value and the bottom indicates the number of ones in each solution.

The examples of this section, as well as numerous other numerical experiments, indicate that both the PG-type methods typically give more accurate approximate solutions than the NST-type methods, but the former methods may require many iterations. The computational cost of the PN-LS and PNKKRS-LS is too

high to make these methods attractive when the number of nodes, n, is large.

6 Conclusion

This paper introduces several methods for recovering an unknown sparse adjacency matrix of a network based on a corrupted version of the communicability matrix. It discusses convergence properties of the NST-LS and PG-LS methods, and illustrates the effect the choice of the regularization parameter μ has on the computed results.

7 Appendix

We illustrate with an example that the use of the MATLAB function logm to compute the matrix logarithm of an available noise-contaminated communicability matrix $\mathbf{C}$ may yield inferior results when compared to results determined by the NST-LS and PG-LS methods. Consider the Erdős-Rényi model, whose adjacency matrix $\mathbf{A}_{\text{true}}$ can be generated by using the code erdrey(n, m) in the CONTEST toolbox. Here n is the number of nodes and $2m$ is the number of 1's in the matrix. We let $n = 50$ and $m = 113$, and obtain a symmetric adjacency matrix $\mathbf{A}_{\text{true}}$ with 226 entries equal to one. Adding 3% white Gaussian noise $\mathbf{N} \in \mathbb{R}^{50\times 50}$ to the true communicability matrix $\mathbf{C}_{\text{true}} = \exp(\mathbf{A}_{\text{true}})$ gives us the corrupted version $\mathbf{C} = \mathbf{C}_{\text{true}} + \mathbf{N}$. We apply the MATLAB function logm to $\mathbf{C}$, resulting a nonprincipal matrix logarithm along with a warning message. Subsequently, we replace all non-real entries of $\log(\mathbf{C})$ with their real parts. This led us to a densely approximated solution, denoted as $\mathbf{A}_d$. We then select the regularization parameter $\mu > 0$ in a manner that ensured the projected solution has 226 ones.

We also use $\mathbf{A}_d$ in the right-hand side of (24) and apply some of the algorithms NST-LS, PG-LS, and PGNST-LS to solve (4) and choose the regularization parameter $\mu > 0$ so that the computed solutions after projection have 226 ones. Figure 6 shows that the computed solution determined by using logm is missing some edges when compared with $\mathbf{A}_{\text{true}}$. In fact, it is less accurate than the approximations obtained by the Newton-type and PG-LS methods.

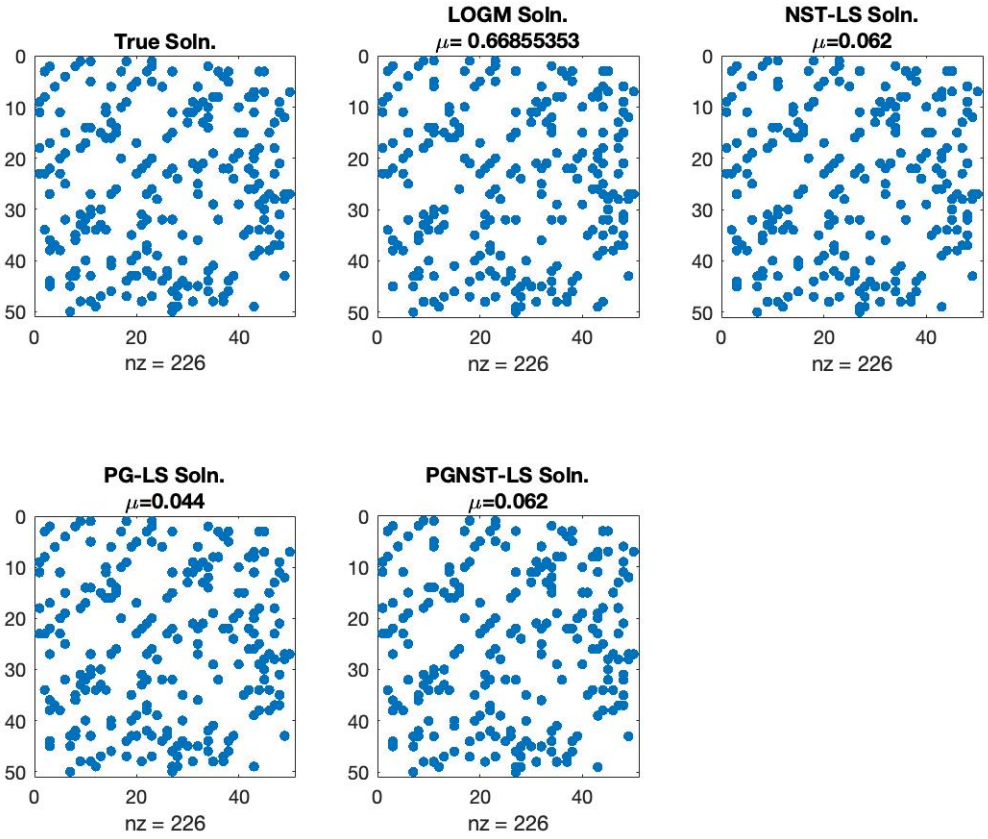

Figure 6: Counter-example: Comparison of the true and approximate solutions using the the LOGM, NST-LS, PG-LS, and PGNST-LS methods with their corresponding μ values. The heading of each subplot shows the corresponding μ value and the bottom indicates the number of ones in each solution.

8 Acknowledgment

The authors would like to thank the referees for comments that led to improvements of the presentation.

References

[1] A. Beck, *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*, SIAM, Philadelphia, 2014.

[2] A. Beck, *First-Order Methods in Optimization*, SIAM, Philadelphia, 2017.

[3] I. Borg and P. J. F. Groenen, *Modern Multidimensional Scaling Theory and Applications*, Springer, New York, 1997.

[4] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.

[5] Celegans neural data and Western US Power data, `https://sparse.tamu.edu`.

[6] O. De la Cruz Cabrera, M. Matar, and L. Reichel, *Analysis of directed networks via the matrix exponential*, J. Comput. Appl. Math., 355 (2019), pp. 182–192.

[7] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Philadelphia, 1996.

[8] L. Dieci, *Considerations on computing real logarithms of matrices, Hamiltonian logarithms, and skew-symmetric logarithms*, Linear Algebra Appl., 244 (1996), pp. 35–54.

[9] E. Estrada, *The Structure of Complex Networks: Theory and Applications*, Oxford University Press, Oxford, 2011.

[10] E. Estrada and J. A. Rodriguez-Velazquez, *Subgraph centrality in complex networks*, Phys. Rev. E, 71 (2005), Art. 056103.

[11] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, Philadelphia, 2008.

[12] P. Kandolf, A. Koskela, S. D. Relton, and M. Schweitzer, *Computing low-rank approximations of the Frèchet derivative of a matrix function using Krylov subspace methods*, Numer. Linear Algebra Appl., 28 (2020), Art. e2401.

[13] J. D. Lee, Y. Sun, and M. A. Saunders, *Proximal Newton-type methods for minimizing composite functions*, SIAM J. Optim., 24 (2014), pp. 1420–1443.

[14] H. Liu and R. F. Barber, *Between hard and soft thresholding: Optimal iterative thresholding algorithms*, Inf. Inference, 9 (2020), pp. 899–933.

[15] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, Philadelphia, 2000.

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

[17] A. Taylor and D. J. Higham, *CONTEST: A controllable test matrix toolbox for MATLAB*, ACM Trans. Math. Software, 35 (2009), Art. 26.

[18] Y. Zhang and W. Ye, *$L_{2/3}$ regularization: Convergence of iterative thresholding algorithm*, J. Vis. Commun. Image Represen., 33 (2015), pp. 350–357.