Electronic Transactions on Numerical Analysis.
Volume 53, pp. 439–458, 2020.
Copyright © 2020, Kent State University.
ISSN 1068–9613.
DOI: 10.1553/etna_vol53s439

ETNA
Kent State University and
Johann Radon Institute (RICAM)

PERFORMANCE AND STABILITY OF DIRECT METHODS FOR COMPUTING GENERALIZED INVERSES OF THE GRAPH LAPLACIAN[*]

MICHELE BENZI[†], PARASKEVI FIKA[†], AND MARILENA MITROULI[‡]

Abstract. We consider the computation of generalized inverses of the graph Laplacian for both undirected and directed graphs, with a focus on the group inverse and the closely related absorption inverse. These generalized inverses encode valuable information about the underlying graph as well as the regular Markov process generated by the graph Laplacian. In [Benzi et al., Linear Algebra Appl., 574 (2019), pp. 123–152], both direct and iterative numerical methods have been developed for the efficient computation of the absorption inverse and related quantities. In this work, we present two direct algorithms for the computation of the group inverse and the absorption inverse. The first is based on the Gauss-Jordan elimination and the reduced row echelon form of the Laplacian matrix and the second on the *bottleneck matrix*, the inverse of a submatrix of the Laplacian matrix. These algorithms can be faster than the direct algorithms proposed in [Benzi et al., Linear Algebra Appl., 574 (2019), pp. 123–152].

Key words. Laplacian matrix, group inverse, absorption inverse, Gauss-Jordan method

AMS subject classifications. 15A09, 05C50, 65F50

1. Introduction.

Let us consider a weighted simple graph $G = (V, E, w)$, directed or undirected, and let $A \in \mathbb{R}^{n\times n}$ be the adjacency matrix associated with G, whose non-zero entries a_{ij} correspond to the weight of the edge from node j to node i, $i \neq j$. We caution the reader that the convention "$a_{ij} \neq 0$ if there is an edge from node j to node i" differs from that adopted in most papers and books on graph theory. Here we adopt this convention in order to be consistent with the notation used in [5] and [18], on which this paper builds. The convention implies that for, any graph, L has zero column sums.

We assume that all weights are positive and therefore $a_{ij} \geq 0$. Also, $a_{ii} = 0$ since the graph contains no loops. We consider the associated outdegree Laplacian matrix $L \in \mathbb{R}^{n\times n}$ specified as $L = W - A$, where the matrix $W \in \mathbb{R}^{n\times n}$ is diagonal with each entry w_{ii} given by the sum of outgoing weights from the node i, i.e., $w_{ii} = \sum_{j=1}^{n} a_{ji}$; the weighted outdegree. We consider also the normalized Laplacian matrix $\tilde{L} \in \mathbb{R}^{n\times n}$ which is given by $\tilde{L} = (W - A)W^{-1} = I_n - AW^{-1}$. Note that both L and $\tilde{L}$ are singular M-matrices. We mention in passing that other definitions of graph Laplacians can be found in the literature; see, e.g., [8, 9].

It holds $\mathrm{rank}(L) = \mathrm{rank}(\tilde{L}) \leq n-1$ and equality holds if the graph is strongly connected. When $\mathrm{rank}(L) = n - 1$, the null space of L is spanned by a unique positive vector. The algorithms in this paper apply to strongly connected graphs, corresponding to the case where the adjacency matrix (and thus the Laplacian) is irreducible. In the general case, there exists a permutation matrix P such that

$$L = P \begin{bmatrix} L_{11} & L_{12} \\ 0 & L_{22} \end{bmatrix} P^T ,$$

where L_{11} is block triangular with irreducible, invertible diagonal blocks while L_{22} is block diagonal with irreducible, singular blocks. The block L_{11} (and therefore L_{12}) may be absent, for instance if the graph is strongly connected or undirected. As described for example in [7, Theorem 7.7.3], the group inverse of L (see the definition below) exists, and can be obtained

[*]Received October 19, 2019. Accepted April 1, 2020. Published online on June 10, 2020. Recommended by Giuseppe Rodriguez.

[†]Scuola Normale Superiore, Pisa, Italy ({michele.benzi, paraskevi.fika}@sns.it).

[‡]National and Kapodistrian University of Athens, Department of Mathematics, Panepistimiopolis, 15784 Athens, Greece (mmitroul@math.uoa.gr).

from the inverse of L_{11}, from L_{12}, and from the group inverses of the diagonal blocks of L_{22}. Therefore, in what follows we assume that G is strongly connected. For an undirected graph, this means that G is connected.

Both L and $\tilde{L}$ are column diagonally dominant matrices. More specifically, from the definition of L, it holds $l_{jj} = \sum_{k=1}^{n} a_{kj} > 0$ and $l_{ij} = -a_{ij}$ for $i \neq j$, which implies $l_{jj} = \sum_{k=1}^{n} a_{kj} = \sum_{k=1,k\neq j}^{n} |l_{kj}|$. Similarly, from the definition of $\tilde{L}$ it holds $\tilde{l}_{jj} = 1$ and $\tilde{l}_{ij} = \dfrac{-a_{ij}}{\sum_{k=1}^{n} a_{kj}}$ for $i \neq j$, which implies $\sum_{i=1,i\neq j}^{n} |\tilde{l}_{ij}| = \dfrac{\sum_{i=1}^{n} a_{ij}}{\sum_{k=1}^{n} a_{kj}} = 1 = \tilde{l}_{jj}$. The group generalized inverse of a square matrix A of index one is the unique matrix $A^{\#}$ such that $AA^{\#}A = A$, $A^{\#}AA^{\#} = A^{\#}$, and $AA^{\#} = A^{\#}A$ [7]. The group inverse of the Laplacian matrix encodes valuable information about the graph such as the distance in weighted trees [20], as well as the regular Markov process generated by the graph Laplacian. In particular, for a regular Markov process the ijth entry of $L^{\#}$ can be interpreted as the deviation from the expected time spent in vertex i due to starting from vertex j [18].

Let us now consider a graph $G = (V, E, w)$, directed or undirected, with a notion of absorption on its nodes, i.e., each node of the graph represents a transient state in a Markov process, and assume that each transient state comes with a transition rate $d_i > 0$ to an absorbing state, labeled $n + 1$ if n is the number of nodes in G. If we consider a vector $d = [d_1, d_2, \ldots, d_n]^T$ with the absorption rates as entries, a *graph with absorption* is denoted as the pair (G, d). Such graphs arise naturally in many applications, for example in the modeling of disease spreading in community networks [26]. Further discussion on absorption graphs and useful properties can be found in [18].

We consider a generalized inverse of the graph Laplacian L, denoted as L^d, first introduced in [18], which captures much valuable information about transient random walks on the graph by combining the known node absorption rates d_i with the structural properties of the underlying graph G encoded in the Laplacian matrix L.

We recall that a matrix X^- is said to be a $\{1\}$-inverse of a matrix X if it satisfies the condition $XX^-X = X$. If X^- satisfies additionally the condition $X^-XX^- = X^-$ then it is called a $\{1, 2\}$-inverse of the matrix X [7].

The matrix L^d is a $\{1, 2\}$-inverse X of L which satisfies the conditions

$$\begin{aligned} XLy &= y, \quad \text{for } y \in N_{1,0}, \text{ and} \\ Xy &= \mathbf{0}, \quad \text{for } y \in R_{1,0}, \end{aligned}$$

where

$$\begin{aligned} N_{1,0} &= \{x \in \mathbb{R}^n : Dx \in \text{Range}(L)\}, \\ R_{1,0} &= \{Dx : x \in \text{Ker}(L)\}, \end{aligned}$$

and D is the diagonal matrix with the absorption rates $d_1, d_2, \ldots, d_n$ as entries. This generalized inverse is called the *absorption inverse of* L *with respect to* d. It was proved in [18] that for strongly connected graphs with positive absorption vector d the absorption inverse exists and is unique. In fact, the absorption inverse can be considered as a generalization of the group inverse for graphs with absorption. It is of interest to compute the absorption inverse L^d, which provides a wealth of information on the structure of the underlying graph. For instance, the absorption inverse can be used to define a notion of distance for graph partitioning purposes, and provides centrality measures for ranking the nodes in an absorption graph [5, 18].

In [5] we addressed the efficient computation of the absorption inverse and of useful quantities related to it such as its action on a vector, its entries, and centrality measures associated with it. We also addressed other computational aspects including the problem of how to perform cheap updates of the quantities of interest when the underlying absorption graph undergoes a low-rank modification such as the addition/deletion of an edge. We considered direct and iterative numerical methods for the computation of the absorption inverse and related quantities; the direct algorithms employ the *LU* decomposition of the graph Laplacian and the iterative ones utilize the preconditioned conjugate gradient method or the preconditioned GMRES method.

In this work we revisit the direct computation of the absorption and group inverse matrices and study an alternative computational method that uses the reduced row echelon form of the Laplacian matrix L instead of its *LU* decomposition, employing a modified form of the Gauss-Jordan method. Moreover, we propose an additional approach that is based on the *bottleneck matrix* of the graph Laplacian; see Section 3. These approaches can be faster in some "difficult" cases where the direct algorithm proposed in [5] becomes slow, such as networks whose adjacency matrix is relatively dense. Furthermore, we present an error analysis of our variant of the Gauss-Jordan algorithm for Laplacian matrices.

We mention that a survey of a variety of computational procedures for finding the mean first passage times in Markov chains is presented in [17], from which the corresponding group inverse can be computed; cf., for instance, [17, Proc. 5, Proc. 11].

A variety of numerical methods for the computation of the group inverse for singular M-matrices, and therefore for Laplacian matrices, have been proposed in the literature and are surveyed in [19]. Most of them can be classified in the following categories: Cholesky-based methods in the symmetric case, techniques based on the *QR* factorization, and some other methods based on *LU* and Gauss-Jordan algorithms. Not all of these methods are guaranteed to yield accurate results; see [4] for examples on which methods based on the *QR* and *LU* factorization produce inaccurate solutions. We note that some of these methods proceed by computing the bottleneck matrix of the Laplacian; we also consider such an approach in this paper. For all of these methods, the dominant term of their complexity is of order $\mathcal{O}(n^3)$ for dense $n \times n$ matrices and is similar to the complexity of the algorithms investigated here. In this work we include additionally a treatment of numerical stability issues and present a detailed forward error analysis based on Wilkinson's approach [28].

Throughout the manuscript, the superscript T denotes the transpose and I_n stands for the identity matrix of dimension n. We denote by $\mathbf{1}$ and $\mathbf{0}$ the column vectors of length n of all ones and of all zeros, respectively. The $m \times n$ matrix with all zeros is denoted by $\mathbf{0}_{m,n}$. In the floating point analysis, $fl(x)$ denotes the floating point representation of a real number x, t and β are the number of significant digits and the base of the floating point arithmetic respectively, $u = \frac{1}{2}\beta^{1-t}$ is the unit round off error, $t_1 = t - log_\beta(1.01)$, and $u_1 = \frac{1}{2}\beta^{1-t_1}$.

2. Gauss-Jordan-based algorithms.

2.1. Preliminaries.

For the group inverse matrix it holds [4, Proposition 2]

$$L^{\#} = (I_n - v\mathbf{1}^T)Y(I_n - v\mathbf{1}^T), \tag{2.1}$$

where Y is any $\{1\}$-inverse of the Laplacian matrix L and v is a positive vector in $\mathrm{Ker}(L)$ with $\sum_{i=1}^n v_i = 1$. With $L^{\#}$ as in (2.1) it can be easily seen that $LL^{\#}L = L$, $L^{\#}LL^{\#} = L^{\#}$, and $LL^{\#} = L^{\#}L$. For the absorption inverse matrix it holds [18, Theorem 3]

$$L^d = (I_n - VD)Y(I_n - DV), \tag{2.2}$$

where Y is any $\{1\}$-inverse of the Laplacian matrix L, $V = v\mathbf{1}^T/\tilde{d}$, and $\tilde{d} = d^Tv$. Of course, since $L^{\#}$ and L^d are unique, they do not depend on the choice of the $\{1\}$-inverse Y in formulae (2.1) and (2.2), which form the basis for computational methods for the group and the absorption generalized inverses.

In [5], the generalized inverse L^- obtained by means of the *LDU* factorization is assumed as Y in formula (2.2). In particular, let us consider the triangular factorization

$$L = \mathcal{L}\mathcal{D}\mathcal{U},$$

where $\mathcal{L}$ and $\mathcal{U}$ are unit lower and upper triangular matrices, respectively, and $\mathcal{D}$ is a diagonal matrix of the form $\mathcal{D} = \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_{n-1}, 0)$, with $\delta_i > 0$ for $1 \leq i \leq n-1$. For undirected graphs, $L = L^T$ and thus $\mathcal{U} = \mathcal{L}^T$. We further note that any symmetric permutation PLP^T (with P a permutation matrix) admits a similar factorization.

Then, the following $\{1,2\}$-inverse of L is taken as Y in (2.2):

$$L^- = \mathcal{U}^{-1}\mathcal{D}^-\mathcal{L}^{-1},$$

where $\mathcal{D}^- = \operatorname{diag}(\delta_1^{-1}, \delta_2^{-1}, \ldots, \delta_{n-1}^{-1}, 0)$. Concerning the computation of a normalized vector v in the kernel of the Laplacian matrix, which is required in formula (2.2), we have $v = (1/n)\mathbf{1} \in \operatorname{Ker}(L)$ if the graph is balanced (for each node in the graph the indegree coincides with the outdegree) since in this case $L\mathbf{1} = \mathbf{0}$. In case of unbalanced graphs, when looking for a nonzero solution to $Lv = \mathbf{0}$, we factor L (or, in practice, a symmetric permutation PLP^T of it) as $L = \mathcal{L}\mathcal{D}\mathcal{U}$ and solve the equivalent system $\mathcal{D}\mathcal{U}v = \mathbf{0}$. Since the last equation of this system is the identity $0 = 0$, this is actually an underdetermined system of $n-1$ linear equations in n unknowns. Fixing the value of $v_n = 1$ yields the equivalent upper triangular system $\mathcal{U}v = \mathbf{e}_n$ which is easily solved by backsubstitution. Since $\mathcal{U}^{-1}$ is a nonnegative matrix, so is v. Normalization of v in the 1-norm produces the desired vector in $\operatorname{Ker}(L)$. In what follows, we will refer to the method described above (cf. [5, Algorithm 1]) as the $\mathcal{L}\mathcal{D}\mathcal{U}$-$L^d$ Algorithm.

2.2. The Gauss-Jordan algorithm for computing a $\{1\}$-inverse. In this section, we study an alternative way for obtaining a generalized inverse Y in formulae (2.2) and (2.1) by considering the Gauss-Jordan elimination of the graph Laplacian L applied to the augmented matrix $[L\ I_n]$. The employment of the Gauss-Jordan algorithm for the computation of generalized inverses has been also studied in [7, Ch.12], [27, Ch.5]; see also [2, 21, 22] and references therein. We have the following result.

PROPOSITION 2.1. *Let $L \in \mathbb{R}^{n\times n}$ be the Laplacian matrix. By applying row operations to the augmented matrix $[L\ I_n]$ we obtain the form $[R\ F]$, where $R \in \mathbb{R}^{n\times n}$ is the reduced row echelon form of the Laplacian matrix which is partitioned as*

$$R = \begin{bmatrix} I_{n-1} & u \\ \mathbf{0}^T & 0 \end{bmatrix},$$

$u \in \mathbb{R}^n$. Furthermore, it holds that $[-u^T,\ 1]^T \in \operatorname{Ker}(L)$ and F is a $\{1\}$-inverse of L.

Proof. Let v be the unique positive vector in the null space of L with entries adding up to 1, i.e., $Lv = \mathbf{0}$, and let R be the reduced row echelon form of the Laplacian matrix L. Therefore $Rv = \mathbf{0}$, which implies

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \end{bmatrix} = -v_n \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{bmatrix} \text{ and hence } \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = v_n \begin{bmatrix} -u_1 \\ -u_2 \\ \vdots \\ -u_{n-1} \\ 1 \end{bmatrix}.$$

Thus $[-u^T,\ 1]^T \in \operatorname{Ker}(L)$, and its normalization in the 1-norm gives the desired vector $v \in \operatorname{Ker}(L)$ with the normalization factor v_n.

Performing row operations on $[L\ I_n]$, we obtain the equivalent form $[R\ F]$. Therefore it is $FL = R$ which implies $LFL = LR$. It holds $LR = L$ as we see below. Suppose the Laplacian matrix L is partitioned as

$$L = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix},$$

where L_{n-1} is $(n-1)\times(n-1)$. Thus

$$LR = \begin{bmatrix} L_{n-1} & L_{n-1}u \\ w^T & w^T u \end{bmatrix}.$$

Furthermore, since $[-u^T,\ 1]^T \in \mathrm{Ker}(L)$, it holds

$$L\begin{bmatrix} -u \\ 1 \end{bmatrix} = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix}\begin{bmatrix} -u \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and hence $L_{n-1}u = z$. Moreover, from the (unique) block LU factorization of L,

$$L = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix} = \begin{bmatrix} I_{n-1} & \mathbf{0}_{n-1,1} \\ w^T L_{n-1}^{-1} & 1 \end{bmatrix}\begin{bmatrix} L_{n-1} & z \\ \mathbf{0}_{1,n-1} & 0 \end{bmatrix},$$

we obtain $l_{n,n} = w^T L_{n-1}^{-1} z$, i.e., $l_{n,n} = w^T u$. Therefore, we have

$$LR = \begin{bmatrix} L_{n-1} & L_{n-1}u \\ w^T & w^T u \end{bmatrix} = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix} = L.$$

Then, since $LR = L$, we have $LFL = L$ and hence the matrix F is a $\{1\}$-inverse of L. □

REMARK 2.2. The same result holds if we consider the normalized Laplacian $\tilde{L}$ instead of L as defined in Section 1. In what follows in this section, we refer to L as either of the Laplacian matrices L or $\tilde{L}$.

2.3. The algorithms.

Let us consider a column diagonally dominant M-matrix $L = [l_{ij}]$, with $l_{ij} \le 0$ if $i \ne j$ and $l_{ii} > 0$ where $l_{ii} = \sum_{k=1,k\ne i}^{n} |l_{ki}|$. We apply the Gauss-Jordan elimination process to this matrix and observe that the multipliers $m_{ik} = \frac{l_{ik}^{(k)}}{l_{kk}^{(k)}} \le 0$, since $l_{ik}^{(k)}$ is negative and $l_{kk}^{(k)}$ is positive. Also, for $i \ne j$, the relation $l_{ij}^{(k+1)} = l_{ij}^{(k)} - m_{ik} l_{kj}^{(k)}$ implies that the element $l_{ij}^{(k+1)}$ remains negative as addition of negative quantities. Furthermore, if $l_{ij}^{(k)}$ is the element l_{ij} at step k then $l_{ij}^{(k+1)} \le l_{ij}^{(k)}$ for $i \ne j$.

Because the property "the active $(n-k)\times(n-k)$ submatrix is an M-matrix with zero column sums" is preserved at each step of the row reduction process, for the diagonal elements we can compute the ith pivot as $l_{ii}^{(k+1)} = -\sum_{t=k+1,t\ne i}^{n} l_{ti}^{(k+1)}$, $i = k+1,\ldots,n$, adding in this way only numbers with the same sign (cf. [24]), which leads to a stable computation as proved in Section 2.5. This way of computing the diagonal elements is the one used in the well-known GTH algorithm [15].

The Laplacian matrix L possesses the property that no row interchanges are necessary at any step during the Gauss-Jordan elimination process. This is again due to the fact that at each step k of the process, the property of being an M-matrix with zero column sums is preserved in the active $(n-k)\times(n-k)$ submatrix. Indeed, in the first $n-1$ steps no pivots become zero and thus we conclude that pivoting is not needed. Next, the steps of the algorithms for computing the matrices $L^{\#}$ and L^d are presented. In what follows, the augmented matrix $B=[L\ I_n]$ is overwritten with the row transformed matrix $[R\ F]$.

Algorithm 1: Gauss-Jordan - $L^{\#}$.

```
Input: $L \in \mathbb{R}^{n\times n}$, a Laplacian matrix.
Output: $L^{\#}$.

Set $B = [L\ I_n]$, the augmented matrix.
For $k = 1 : n-1$
  $m_{ik} = b_{ik}/b_{kk}, \quad i = 1,\dots,n,\ i \neq k.$
  $b_{ij} = b_{ij} - m_{ik}b_{kj}, \quad i = 1,\dots,n,\ i \neq k,\ \ j = k,\dots,n+k,\ \ i \neq j.$
  $b_{ii} = -\sum_{t=k+1,t\neq i}^{n} b_{ti}, \quad i = k+1,\dots,n.$
  $b_{kj} = b_{kj}/b_{kk}, \quad j = k,\dots,n+k.$
end
$F = B(1:n, n+1:2n)$, a generalized inverse of the graph Laplacian.
$v_1 = [B(1:n-1,n);1]$, a vector in $\mathrm{Ker}(L)$.
Set $v = v_1/\|v_1\|_1$.
Compute $Y_1 = v\mathbf{1}^TF,\ \ Y_2 = Fv\mathbf{1}^T$, and $Y_3 = Y_1v\mathbf{1}^T$.
$L^{\#} = F - Y_1 - Y_2 + Y_3$, the group inverse of $L$.
```

Algorithm 2: Gauss-Jordan - L^d.

```
Input: $L \in \mathbb{R}^{n\times n}$, a Laplacian matrix and $d \in \mathbb{R}^n$, a vector of absorption rates.
Output: $L^d$.

Set $B = [L\ I_n]$, the augmented matrix.
For $k = 1 : n-1$
  $m_{ik} = b_{ik}/b_{kk}, \quad i = 1,\dots,n,\ i \neq k.$
  $b_{ij} = b_{ij} - m_{ik}b_{kj}, \quad i = 1,\dots,n,\ i \neq k,\ \ j = k,\dots,n+k,\ \ i \neq j.$
  $b_{ii} = -\sum_{t=k+1,t\neq i}^{n} b_{ti}, \quad i = k+1,\dots,n.$
  $b_{kj} = b_{kj}/b_{kk}, \quad j = k,\dots,n+k.$
end
$F = B(1:n, n+1:2n)$, a generalized inverse of the graph Laplacian.
$v_1 = [B(1:n-1,n);1]$, a vector in $\mathrm{Ker}(L)$.
Set $v = v_1/\|v_1\|_1$.
Compute $\tilde{d} = d^Tv,\ V = v\mathbf{1}^T/\tilde{d},\ D = \mathrm{diag}(d),$
 $Y_1 = VDF,\ Y_2 = FDV$, and $Y_3 = Y_1DV$.
$L^d = F - Y_1 - Y_2 + Y_3$, the absorption inverse of $L$.
```

2.4. The implementation of the algorithms and computational complexity. Algorithms 1 and 2 are implemented in MATLAB. The part of the algorithms concerning the Gauss-Jordan elimination is implemented in binary format, using LAPACK routines from Intel's MKL, because the internal function `rref` of MATLAB is quite slow. The matrices V and D in Algorithm 2 are not formed explicitly and the computations are done by performing successive matrix-vector products and diagonal scalings as follows, where $.*$ stands for the element-wise multiplication.

1. Let F and v be the quantities as described in Algorithm 2.
2. Set $Dv = d.*v$, $\tilde{d} = d^T * v$, and $Y_2 = (F * Dv) * \mathbf{1}^T/\tilde{d}$.

3. Evaluate the quantities $\tilde{Y_2} = d^T * Y_2$ and $Y_3 = v * \tilde{Y_2}/\tilde{d}$.
4. Obtain the quantities $\tilde{Y_1} = d^T * F$ and $Y_1 = v * \tilde{Y_1}/\tilde{d}$.
5. Set $L^d = F - Y_1 - Y_2 + Y_3$, the absorption inverse.

It is worth observing that the computations $b_{ij} = b_{ij} - m_{ik}b_{kj}$, $i = 1, \ldots, n$, $i \neq k$, $i \neq j$, $j \geq k$, needed to update the entries of the $n \times 2n$ augmented matrix, can be limited to $j = 1, \ldots, n + k$ instead of $j = 1, \ldots, 2n$, resulting in a lower computational complexity.

The main computational task of Algorithms 1 and 2 is governed by the Gauss-Jordan elimination of $L \in \mathbb{R}^{n\times n}$ applied to the augmented matrix $[L\ I_n]$. This computation requires $n - 1$ flops for the multipliers, $(n - 1)(n + 1) - (n - k)$ for the update of the nondiagonal elements of the matrix, $(n - k)(n - k - 1)$ for the separate update of the diagonal elements, and finally $(n + 1)$ flops for the division of the pivot row by the pivot element. Thus, the total complexity is $\sum_{k=1}^{n-1}\{(n-1)+(n-1)(n+1)-(n-k)+(n-k-1)(n-k)+(n+1)\}$ flops. The dominant term in this summation is $4n^3/3$, therefore $\mathcal{O}(4n^3/3)$ arithmetic operations are required. For the evaluation of the remaining part of the algorithm, i.e., steps 2-4, $\mathcal{O}(n^2)$ arithmetic operations are required.

REMARK 2.3. We remark that performing the computation of the diagonal (pivot) terms as described leads to a slight increase of the computational complexity of the algorithm, from $\mathcal{O}(n^3)$ to $\mathcal{O}\left(\frac{4}{3}n^3\right)$. However, this ensures that the subtraction of almost equal terms is avoided and yields highly accurate results.

2.5. Numerical stability.

In this section we will discuss the stability of the proposed algorithms. We will focus on the stability of the Gauss-Jordan process applied to the Laplacian matrix L, which is a singular M-matrix [6]. We recall (see [12]) that in the Gaussian elimination of an irreducible M-matrix, singular or nonsingular, no pivoting is necessary.

Concerning the stability of the Gauss-Jordan approach, in [16, Corollary 14.7] it is proved that for row diagonally dominant matrices the Gauss-Jordan elimination without pivoting is both forward stable and row-wise backward stable. Concerning the column diagonally dominant matrices, which is our case, the Gauss-Jordan reduction for these matrices is forward stable but in general not backward stable [23].

REMARK 2.4. We note that by applying the Gauss-Jordan approach to L^T, which is row diagonally dominant with row sums equal to zero, the backward stability is preserved. However, in this case the Algorithms 1 and 2 must be modified. In particular, setting $\hat{B} = [L^T\ I_n]$ and applying the Gauss-Jordan reduction to the augmented matrix $\hat{B}$, we obtain a generalized inverse of L by computing $\hat{B}(1 : n, n + 1 : 2n)^T$. A vector in $\mathrm{Ker}(L)$ is then given by $\hat{B}(n, n + 1 : 2n)^T$.

Next, we study the forward error analysis of the Gauss-Jordan elimination for Laplacian matrices as presented in Algorithms 1 and 2. We adopt the forward error analysis approach as it was introduced by Wilkinson in [28].

The following floating point computations $fl(\cdot)$ are performed:

```
For $k = 1 : n - 1$
  $m_{ik} = fl(b_{ik}/b_{kk}),\quad i = 1, \ldots, n,\ i \neq k.$
  $b_{ij} = fl(b_{ij} - m_{ik}b_{kj}),\quad i = 1, \ldots, n,\ i \neq k\ i \neq j,\quad j = k, \ldots, n + k.$
  $b_{ii} = -fl(\sum_{t=k+1,t\neq i}^{n} b_{ti}),\quad i = k + 1, \ldots, n.$
end
```

At step k of the algorithm, the matrix $B^{(k)}$ will have elements $b_{ij}^{(k)}$. In floating point arithmetic with unit round off u, the elements $b_{ij}^{(k+1)}$ are specified from the relations

$$m_{ik} = fl(b_{ik}^{(k)}/b_{kk}^{(k)}), \quad i = 1, \ldots, n, \; i \neq k,$$
$$b_{ij}^{(k+1)} = \begin{cases} 0 & i = 1, \ldots, n, \; i \neq k, \; j = k, \\ fl(b_{ij}^{(k)} - m_{ik} b_{kj}^{(k)}) & i = 1, \ldots, n, \; i \neq k, \; i \neq j, \;\; j = k, \ldots, n + k, \\ -fl(\sum_{t=k+1, t\neq i}^{n} b_{ti}^{(k+1)}) & i = k + 1, \ldots, n, \; j = i, \\ b_{ij}^{(k)} & \text{elsewhere.} \end{cases}$$

THEOREM 2.5 (Forward stability). *At step k of the row transformation process, the matrix $\tilde{B}^{(k)}$ computed using floating point arithmetic with unit round off u satisfies the relation*

$$\tilde{B}^{(k)} = B^{(k)} + E^{(k)},$$

where $E^{(k)}$ is the error matrix with $\|E^{(k)}\|_1 \leq (n-1)u_1 \|B^{(k)}\|_1$, $u_1 = 1.01u$. Therefore, the normwise relative error bound

$$Rel = \frac{\|\tilde{B}^{(k)} - B^{(k)}\|_1}{\|B^{(k)}\|_1} \leq (n-1)u_1$$

holds. Thus, each step of the process is forward stable.

Proof. Assume the first k steps of the algorithm have been completed. Firstly, for the multiplier m_{ik} we have $m_{ik} = fl(b_{ik}^{(k)}/b_{kk}^{(k)}) = (b_{ik}^{(k)}/b_{kk}^{(k)})(1+\epsilon)$, $|\epsilon| \leq u$, which implies $m_{ik} b_{kk}^{(k)} = b_{ik}^{(k)} + b_{ik}^{(k)}\epsilon$. Therefore, $\epsilon_{ik}^{(k)} = b_{ik}^{(k)}\epsilon$ is the error due to the multiplier, and

$$\begin{aligned} b_{ij}^{(k+1)} &= fl(b_{ij}^{(k)} - fl(m_{ik} b_{kj}^{(k)})) \\ &= fl(b_{ij}^{(k)} - m_{ik} b_{kj}^{(k)}(1+\epsilon)(1+\epsilon_1)) \\ &= (b_{ij}^{(k)} - m_{ik} b_{kj}^{(k)}(1+\epsilon)(1+\epsilon_1))(1+\epsilon_2), \qquad |\epsilon_1| \leq u, \; |\epsilon_2| \leq u. \end{aligned}$$

Set $(1+\epsilon)(1+\epsilon_1)(1+\epsilon_2) = 1 + \xi_1$, $|\xi_1| \leq 3u_1$, $u_1 = 1.01u$. Then

$$b_{ij}^{(k+1)} = b_{ij}^{(k)}(1+\epsilon_2) - m_{ik} b_{kj}^{(k)}(1+\xi_1).$$

Therefore,

$$b_{ij}^{(k+1)} = b_{ij}^{(k)} - m_{ik} b_{kj}^{(k)} + b_{ij}^{(k)}\epsilon_2 - m_{ik} b_{kj}^{(k)}\xi_1$$

and thus the errors at the generic step k satisfy

$$E_{ij}^{(k)} = b_{ij}^{(k)}\epsilon_2 - m_{ik} b_{kj}^{(k)}\xi_1, \;\; |\epsilon_2| \leq u, \;\; |\xi_1| \leq 3u_1.$$

For $i = j$, $i = k+1, \ldots, n$ the diagonal entries $b_{ii}^{(k+1)}$ are computed as off-diagonal column sums $b_{ii}^{(k+1)} = -fl(\sum_{t=k+1, t\neq i}^{n} b_{ti}^{(k+1)})$. In particular, we have

$$b_{ii}^{(k+1)} = -\sum_{t=k+1, t\neq i}^{n} b_{ti}^{(k+1)}(1+\eta_t),$$

where $(1-u)^{n+1-t} \le 1+\eta_t \le (1+u)^{n+1-t}$, which is replaced by $|\eta_t| \le (n+1-t)u_1$, $u_1 = 1.01u$. Hence

$$\left| fl\left(\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}\right) - \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)} \right|$$
$$= \left| \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}(1+\eta_t) - \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)} \right| = \left| \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}\eta_t \right|.$$

Therefore,

$$E_{ii}^{(k)} = \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}\eta_t, \;\; |\eta_t| \le (n+1-t)u_1 \le (n-k)u_1 \le (n-1)u_1.$$

Thus, we see that after step k of the elimination procedure we have computed

$$\tilde{B}^{(k)} = B^{(k)} + E^{(k)},$$

where for the error matrix $E^{(k)}$ it is

$$E_{ij}^{(k)} = \begin{cases} b_{ij}^{(k)}\epsilon_2 - m_{ik}b_{kj}^{(k)}\xi_1 & i = 1,\ldots,n,\ i\neq k,\ i\neq j,\ \ j = k,\ldots,n+k, \\ \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k)}\eta_t & i = k+1,\ldots,n,\ j = i, \\ 0 & \text{elsewhere.} \end{cases}$$

Setting $\delta = \max\{\eta_t, \xi_1, \epsilon_2\}$ we have $|\delta| \le (n-1)u_1$ for $n \ge 4$, and for the error matrix $E^{(k)}$ we obtain

$$\|E^{(k)}\|_1 \le \|B^{(k)}\|_1|\delta| \le \|B^{(k)}\|_1(n-1)u_1.$$

For the relative error it holds

$$Rel = \frac{\|\tilde{B}^{(k)} - B^{(k)}\|_1}{\|B^{(k)}\|_1} = \frac{\|E^{(k)}\|_1}{\|B^{(k)}\|_1} \le \frac{(n-1)u_1\|B^{(k)}\|_1}{\|B^{(k)}\|_1} = (n-1)u_1.$$

In the case $n \le 3$ we have $|\delta| \le 3u_1$. Therefore the bound for $\|E^{(k)}\|_1$ takes the form $\|E^{(k)}\|_1 \le \|B^{(k)}\|_1 3u_1$ and the relative error satisfies $Rel = \|E^{(k)}\|_1/\|B^{(k)}\|_1 \le 3u_1$. As the upper bound for the relative error depends linearly on n, each step is forward stable. ◻

Next, we demonstrate an element-wise forward error analysis for each entry of the computed matrix $B^{(k)}$, which reveals why the diagonal elements of the matrix must be computed as off-diagonal column sums $b_{ii}^{(k+1)} = -fl(\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)})$, $i = k+1,\ldots,n$.

COROLLARY 2.6. *The relative error for* $b_{ij}^{(k+1)} = fl(b_{ij}^{(k+1)} - m_{ik}b_{kj}^{(k+1)})$, $i \neq j$, *computed using floating point arithmetic with unit round off* u, *is bounded by*

$$Rel \le 4u_1, \;\; u_1 = 1.01u.$$

Proof. We have (cf. Theorem 2.5)

$$b_{ij}^{(k+1)} = fl(b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}) = b_{ij}^{(k)}(1+\epsilon_2) - m_{ik}b_{kj}^{(k)}(1+\xi_1)$$

with $|\epsilon_2| \leq u$, $|\xi_1| \leq 3u_1$, and $u_1 = 1.01u$. Therefore, for the relative error we have

$$\begin{aligned} Rel &= \frac{|fl(b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}) - (b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)})|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \\ &= \frac{|b_{ij}^{(k)}(1+\epsilon_2) - m_{ik}b_{kj}^{(k)}(1+\xi_1) - (b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)})|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \\ &= \frac{|b_{ij}^{(k)}\epsilon_2 - m_{ik}b_{kj}^{(k)}\xi_1|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \leq \frac{|b_{ij}^{(k)}||\epsilon_2| + |m_{ik}b_{kj}^{(k)}||\xi_1|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \leq \frac{|b_{ij}^{(k)}|u + |m_{ik}b_{kj}^{(k)}|3u_1}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \end{aligned} \tag{2.3}$$

which can be bounded since $b_{ij}^{(k)}$ and $-m_{ik}b_{kj}^{(k)}$ have both the same sign for $i \neq j$ and we avoid the cancellation error. In particular, concerning the left $n \times n$ part of the matrix B, i.e., the matrix L, in a previous discussion we remarked that both $b_{ij}^{(k)}$ and $-m_{ik}b_{kj}^{(k)}$ are negative since m_{ik} and $b_{kj}^{(k)}$ are both negative quantities. Concerning the right $n \times n$ part of the matrix B, i.e., the identity matrix I_n, we can see that at each step of the row reduction process its elements remain positive or zero. For instance, in the first step of the process, we have that any $b_{ij}^{(1)} = b_{ij} - m_{ik}b_{kj}$ is nonnegative since $-m_{ik}$ is positive and $b_{ij}, b_{kj} \geq 0$, $i = 1, \ldots, n$, $j = n+1, \ldots, 2n$. Hence, at any step we will have addition of positive numbers. Thus, we have

$$Rel \leq \frac{|b_{ij}^{(k)}|u + |m_{ik}b_{kj}^{(k)}|3u_1}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \leq \alpha u + 3\beta u_1,$$

where

$$\alpha = \frac{|b_{kj}^{(k)}|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \leq 1 \text{ and } \beta = \frac{|m_{ik}b_{kj}^{(k)}|}{|b_{ij}^{(k)} - m_{ik}b_{kj}^{(k)}|} \leq 1.$$

Therefore,

$$Rel \leq \alpha u + 3\beta u_1 \leq 4u_1. \qquad \square$$

REMARK 2.7. From Corollary 2.6 we can see that for $i = j$ the denominator of (2.3) can take small values since the formula $b_{ii}^{(k+1)} = b_{ii}^{(k)} - m_{ik}b_{ki}^{(k)}$ involves subtraction of positive numbers and thus may lead to loss of accuracy due to cancellation of significant digits. In order to avoid this cancellation error, the update formula $b_{ii}^{(k+1)} = b_{ii}^{(k)} - m_{ik}b_{ki}^{(k)}$ is not employed for $i = j$ and the diagonal entries $b_{ii}^{(k+1)}$ are computed as off-diagonal column sums as $b_{ii}^{(k+1)} = -fl(\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)})$, $i = k+1, \ldots, n$. In particular, we have (cf. Theorem 2.5)

$$b_{ii}^{(k+1)} = -\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}(1+\eta_t),$$

where $|\eta_t| \leq (n+1-t)u_1$, $u_1 = 1.01u$. This formula leads to the following error bound.

COROLLARY 2.8. *At step k, the relative error for the computation of*

$$b_{ii}^{(k+1)} = -fl(\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}), \quad i = k+1, \ldots, n,$$

computed using floating point arithmetic with unit round off u satisfies

$$Rel \leq (n-k)u_1, \;\; u_1 = 1.01u.$$

Proof.

$$\begin{aligned} Rel &= \frac{|fl(\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}) - \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|}{|\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|} \\ &= \frac{|\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}(1+\eta_t) - \sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|}{|\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|} \\ &\leq (n-k)u_1 \frac{\sum_{t=k+1,t\neq i}^{n} |b_{ti}^{(k+1)}|}{|\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|}. \end{aligned} \tag{2.4}$$

Therefore, as every $b_{ti}^{(k+1)}$ is negative, $\dfrac{\sum_{t=k+1,t\neq i}^{n} |b_{ti}^{(k+1)}|}{|\sum_{t=k+1,t\neq i}^{n} b_{ti}^{(k+1)}|} = 1$, which yields

$$Rel \leq (n-k)u_1. \qquad \square$$

REMARK 2.9. We see that formula (2.4) guarantees the stability of the computation of the diagonal elements, whereas formula (2.3) may lead to numerical instability in case of $i = j$.

3. The bottleneck approach .

In this section, an alternative approach is presented for the computation of the generalized inverse Y to be used in formulae (2.1) and (2.2). This is based on the bottleneck matrix of the graph Laplacian as described below; see also [5, 18].

LEMMA 3.1 ([5]). *Suppose the Laplacian matrix L is partitioned as*

$$L = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix},$$

where L_{n-1} is $(n-1)\times(n-1)$. Then

$$L^- = \begin{bmatrix} L_{n-1}^{-1} & \mathbf{0}_{n-1,1} \\ \mathbf{0}_{1,n-1} & 0 \end{bmatrix}$$

is a $\{1,2\}$-inverse of the graph Laplacian L.

REMARK 3.2. The positive matrix L_{n-1}^{-1} is called the bottleneck matrix of L based at vertex n in [18, p. 129]; see also [20]. Note that L^- coincides with the matrix M in [18, Proposition 3].

This approach leads to Algorithm 3 below.

```
Algorithm 3: Bottleneck Algorithm.
    Input: $L \in \mathbb{R}^{n\times n}$ a Laplacian matrix.
    Output: $L^{\#}$ or $L^d$.
    Partition $L = \begin{bmatrix} L_{n-1} & z \\ w^T & l_{n,n} \end{bmatrix}$ and compute the inverse of $L_{n-1}$.
    Consider the vector $\begin{bmatrix} -L_{n-1}^{-1}z \\ 1 \end{bmatrix}$ which gives a vector $v \in \mathrm{Ker}(L)$, which is then
      normalized in 1-norm.
    Compute the generalized inverse $L^- = \begin{bmatrix} L_{n-1}^{-1} & \mathbf{0}_{n-1,1} \\ \mathbf{0}_{1,n-1} & 0 \end{bmatrix}$.

    -For the group inverse matrix:
    Compute the matrices
      $Y_1 = v\mathbf{1}^T L^-$,  $Y_2 = L^- v\mathbf{1}^T$, and  $Y_3 = Y_1 v\mathbf{1}^T$.
    Return $L^{\#} = L^- - Y_1 - Y_2 + Y_3$.

    -For the absorption inverse matrix:
    Compute the matrices $V = v\mathbf{1}^T/\tilde{d}$, $D = \mathrm{diag}(d)$ where $\tilde{d} = d^T v$, and the matrices
      $Y_1 = VDL^-$,  $Y_2 = L^- DV$, and  $Y_3 = Y_1 DV$.
    Return $L^d = L^- - Y_1 - Y_2 + Y_3$.
```

REMARK 3.3. For an irreducible Laplacian matrix of order n there are n principal submatrices of order $n-1$, hence n different possible choices of a bottleneck matrix to work with. These are all nonsingular M-matrices, but some may be better conditioned than others. Therefore, in principle, care should be exercised as to the choice of a well-conditioned submatrix. For this purpose, some methods have been proposed in the literature; see, for instance, [14]. A simple rule could be to use Gerschgorin disks to find the "most diagonally dominant" submatrix of L, if any. Obviously, these heuristics add to the overall computational cost. In the context of the present work, for simplicity we always choose the leading principal submatrix of order $n-1$.

Implementation. Algorithm 3 is implemented in the MATLAB environment. For the matrix inverse computation the internal function `inv` of MATLAB was applied. The function `inv` performs an *LU* decomposition of the input matrix or an *LDL* decomposition if the input matrix is Hermitian. It then uses the results to form a linear system whose solution is the matrix inverse X^{-1} [25]. In particular, if $L_{n-1} = \hat{L}\hat{U}$, first the matrix inverse $\hat{U}^{-1}$ is computed and then the equation $X\hat{L} = \hat{U}^{-1}$ is solved for X [16]. Let $\tilde{X}$ be the computed solution of the equation. Then, the following residual bound holds (cf. [16, Ch.14])

$$|\tilde{X}L_{n-1} - I_{n-1}| \le c_n u|\tilde{X}||\hat{L}||\hat{U}|,$$

which gives the forward error bound

$$|\tilde{X} - L_{n-1}^{-1}| \le c_n u|\tilde{X}||\hat{L}||\hat{U}||L_{n-1}^{-1}|,$$

where c_n is a function of n. The evaluation of the inverse of the $(n-1)\times(n-1)$ matrix L_{n-1}, which is the main computational task of Algorithm 3, requires $\mathcal{O}(n^3)$ arithmetic operations. In particular, the *LU* decomposition requires $n^3/3 + \mathcal{O}(n^2)$ flops. The computation of $\hat{U}^{-1}$ (cf. [16, Ch.14]) requires $(n-1)^3/6 + (n-1)^2/2 + (n-1)/3 = n^3/6 + \mathcal{O}(n^2)$ flops and finally, assuming that the triangular solve from the right with L is done by back substitution,

$((n-1)^2(n-2))/2 = n^3/2+\mathcal{O}(n^2)$ flops are required for the solution of equation $X\hat{L} = \hat{U}^{-1}$. Thus, $n^3 + \mathcal{O}(n^2)$ flops are needed. For the evaluation of the remaining part of the algorithm, $\mathcal{O}(n^2)$ arithmetic operations are required.

4. Numerical examples.

In this section numerical experiments are presented for the computation of the absorption inverse matrix L^d via the direct methods of Algorithms 2 and 3. We include a comparison with the direct $\mathcal{LDU}$-L^d Algorithm proposed in [5, Algorithm 1]. For the sake of brevity we only show results for the absorption inverse. The results are very similar for the group inverse. In cases that the given graph is not strongly connected, i.e., $\text{rank}(L) < n-1$, we consider the largest connected graph component. In the experiments for the absorption inverse, random absorption rates are considered. The computations are performed in MATLAB R2015a, 64-bit, on an Intel Core i7 computer with 16 Gb DDR4 RAM.

In many cases, the Laplacian matrix L is sparse. In [5], in the implementation of the direct algorithms for the computation of L^d, a fill-reducing ordering is used when computing the triangular factorization $L = \mathcal{LDU}$. Hence, the matrix L is first permuted symmetrically and then factored. If P is the permutation matrix corresponding to the chosen ordering, then we compute the factorization $PLP^T = \mathcal{LDU}$. In practice, we made use of the `colamd` and `symamd` functions in MATLAB, for directed and undirected graphs respectively, as permutations.

Exploiting sparsity is crucial for the efficient implementation of the proposed algorithms in [5]. Nevertheless, in cases of a Laplacian matrix with sufficient density, the method proposed in [5] becomes slow. On the other hand, the methods proposed in the present manuscript provide faster algorithms for handling dense problems and constitute alternative methods even for sparse networks.

In the numerical experiments that follow, in test problems 1-3 we present the execution time in seconds for the computation of the absorption inverse matrix through the $\mathcal{LDU}$-L^d method presented in [5, Algorithm 1] and through the Gauss-Jordan - L^d Algorithm (Algorithm 2) and the Bottleneck Algorithm (Algorithm 3) averaged over 5 runs. The comparison of these two algorithms is fair if they are both applied to the Laplacian matrix stored in dense format. However, for completeness, the results of [5, Algorithm 1] applied to the Laplacian matrix stored in sparse format and those obtained by using sparse matrix reorderings are also included. In test problems 4-5 we present some numerical experiments that show the numerical stability of the algorithms introduced in this paper.

Test problem 1: The Twitter network. Let us consider the *Twittercrawl* network (with 3656 nodes and 188712 edges), which is a real world unweighted directed network; see [3]. The *Twittercrawl* network is part of the Twitter network [13]; the nodes are users and the directed edges are mentions and re-tweets between users. This network is not strongly connected and hence we consider its largest connected component, which consists of 3485 nodes and 184517 edges. Figure 4.1 depicts the sparsity pattern of the largest connected component of the network and those obtained by using the `colamd`, `symamd`, and `amd` functions as permutations.

In Table 4.1 we present the execution time in seconds for the computation of the absorption inverse considering random absorption rates. We also report the part of the execution time required for the Gauss-Jordan or *LU* part of the algorithm. Note that despite the fact that the *LU* method is faster than Gauss-Jordan one, the remaining part of the algorithm (cf. Algorithm 2) requires less computational effort than those of [5, Algorithm 1], leading to a faster total execution time. For this test problem, the bottleneck matrix approach slightly outperforms the one based on the Gauss-Jordan algorithm.

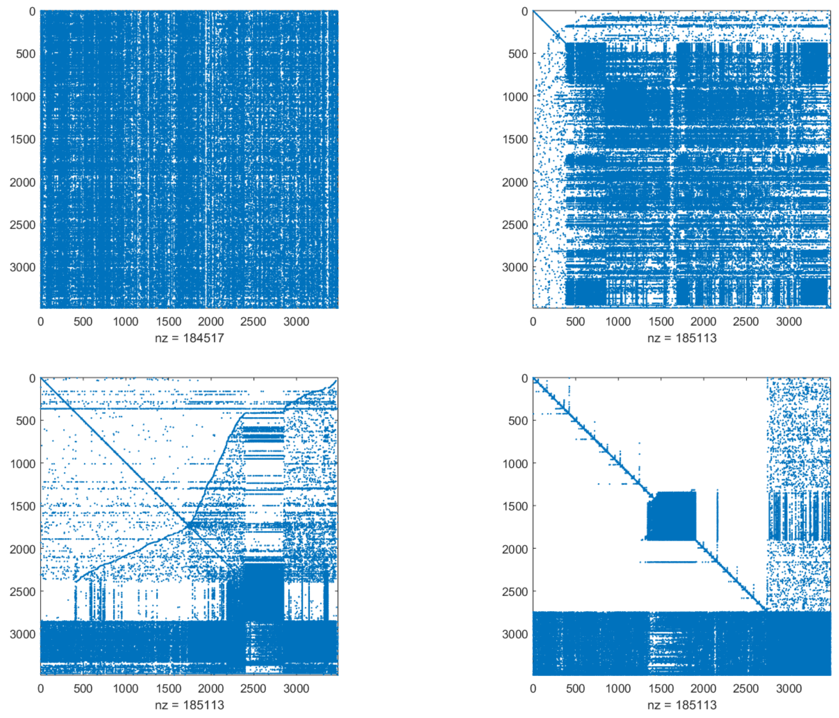

FIG. 4.1. *The sparsity pattern of the largest connected component of the Twittercrawl network without permutations (upper-left) and by using the* `colamd` *function (upper-right), the* `symamd` *function (down-left), and the* `amd` *function (down-right) as permutations.*

TABLE 4.1
Execution time in seconds for the computation of the absorption inverse of the Twittercrawl network.

Method	**Time**
Gauss-Jordan - L^d	1.04e0 sec [GJ: 8.88e-1 sec]
Bottleneck Algorithm	9.55e-1 sec [`inv`: 7.04e-1 sec]
$\mathcal{LDU}$ - L^d	2.11e0 sec [LU: 2.52e-1 sec]
$\mathcal{LDU}$-L^d sparse - no permut	2.19e0 sec [LU: 2.41e-1 sec]
colamd	2.68e0 sec [LU: 5.39e-1 sec]
symamd	2.39e0 sec [LU: 4.79e-1 sec]
amd	2.50e0 sec [LU: 4.04e-1 sec]

Test problem 2: The Cage network. Let us consider the *Cage 9* network (with 3534 nodes and 41594 edges), which is a real world unweighted directed network; see [10]. This is a biological network associated with DNA electrophoresis with 9 monomers in a polymer, and it is strongly connected. Figure 4.2 shows the sparsity pattern of the network and those obtained by using the `colamd`, `symamd`, and `amd` functions as permutations.

In Table 4.2 we present the execution time in seconds for the computation of the absorption inverse considering random absorption rates. Again, the newly proposed algorithms clearly outperform the previous ones, with the bottleneck approach having a slight edge over the one based on the Gauss-Jordan method.

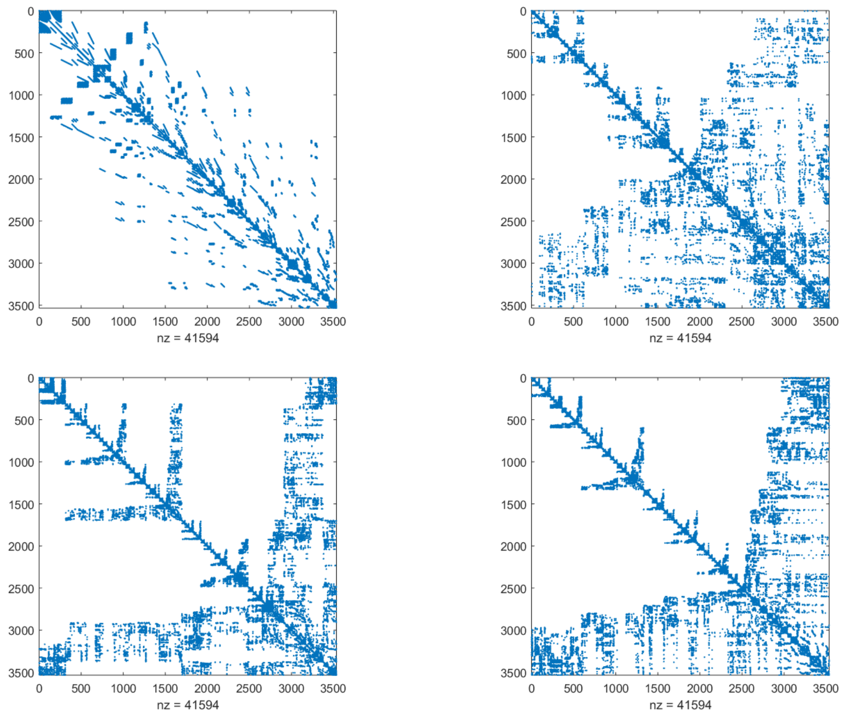

FIG. 4.2. *The sparsity pattern of the Cage 9 network without permutations (upper-left) and by using the* `colamd` *function (upper-right), the* `symamd` *function (down-left), and the* `amd` *function (down-right) as permutations.*

TABLE 4.2
Execution time in seconds for the computation of the absorption inverse of the Cage 9 network.

Method	**Time**
Gauss-Jordan - L^d	1.04e0 sec [GJ: 8.91e-1 sec]
Bottleneck Algorithm	9.56e-1 sec [`inv`: 6.87e-1 sec]
$\mathcal{LDU}$ - L^d	2.15e0 sec [LU: 2.77e-1 sec]
$\mathcal{LDU}$-L^d sparse - no permut	2.16e0 sec [LU: 2.51e-1 sec]
colamd	2.47e0 sec [LU: 4.42e-1 sec]
symamd	2.46e0 sec [LU: 3.90e-1 sec]
amd	2.21e0 sec [LU: 3.54e-1 sec]

Test problem 3: Miscellaneous networks. Next, we examine the behaviour of the algorithms on further numerical examples. These test networks are a representative collection of directed and undirected graphs of moderate size and various degrees of sparsity, and except for *twittercrawl* and *autobahn* they are obtained from the SuiteSparse matrix collection [10]. *Autobahn* network is available from [1].

In the first column of Table 4.3, we report the employed network. If it is undirected, it is marked with an asterisk *. The two next columns display the size of the test network n (or the size of its largest connected component in case that the network is not strongly connected) and the ratio of the number of nonzero elements nnz over the size of the network, i.e., nnz/n. In columns 4-10 we record the execution time in seconds for the computation

ETNA
Kent State University and
Johann Radon Institute (RICAM)

of the absorption inverse matrix through the $\mathcal{LDU}$-L^d method presented in [5, Algorithm 1], through the Gauss-Jordan - L^d Algorithm (Algorithm 2), and the Bottleneck Algorithm (Algorithm 3). In the last column of Table 4.3, the speedup factor (SPF): 'time with dense $\mathcal{LDU}$' divided by 'time with dense G-J' is reported.

TABLE 4.3
Execution time in seconds for the computation of the absorption inverse.

Network	n	nnz/n	**G-J**	**Bottl**	$\mathcal{LDU}$ **dense**	$\mathcal{LDU}$ **sparse** **no perm**	**col**	**sym**	**amd**	SPF
email*	1133	9.62e0	8.02e-2	8.21e-2	1.42e-1	1.52e-1	1.80e-1	1.68e-1	1.57e-1	1.77
data*	2851	1.06e1	6.78e-1	7.08e-1	1.97e0	1.92e0	2.03e0	1.41e0	1.37e0	2.91
uk*	4824	2.83e0	2.34e0	2.25e0	4.24e0	4.25e0	4.62e0	4.59e0	4.66e0	1.81
yeast*	1458	2.70e0	1.48e-1	1.38e-1	2.59e-1	2.39e-1	2.91e-1	3.06e-1	2.89e-1	1.75
power*	4941	2.67e0	2.39e0	2.33e0	4.67e0	4.68e0	4.92e0	5.00e0	5.14e0	1.95
Roget	904	5.34e0	4.90e-2	4.59e-2	8.97e-2	9.18e-2	1.05e-1	1.03e-1	1.00e-1	1.83
autobahn*	1168	2.13e0	8.06e-2	7.67e-2	1.49e-1	1.51e-1	1.65e-1	1.80e-1	1.68e-1	1.86
ca-GrQc*	4158	6.46e0	1.71e0	1.54e0	2.99e0	3.03e0	4.75e0	3.36e0	3.24e0	1.75
Erdos972*	4680	3.00e0	2.10e0	2.23e0	4.19e0	4.21e0	4.44e0	4.58e0	4.63e0	1.99
Erdos02*	5534	3.06e0	3.30e0	3.21e0	6.46e0	6.13e0	6.58e0	6.90e0	7.00e0	1.95
wing_nodal*	10937	1.38e1	1.83e1	1.86e1	4.30e1	4.20e1	4.33e1	4.52e1	4.38e1	2.35
Twittercrawl	3485	5.29e1	1.04e0	9.55e-1	2.11e0	2.19e0	2.68e0	2.39e0	2.50e0	2.03
cage9	3534	1.18e1	1.04e0	9.56e-1	2.15e0	2.16e0	2.47e0	2.46e0	2.21e0	2.06
wiki-Vote	1300	3.04e1	1.23e-1	1.05e-1	3.07e-1	3.17e-1	3.36e-1	3.46e-1	2.42e-1	2.50
hep-Th*	5835	4.74e0	3.70e0	3.49e0	7.39e0	7.57e0	1.21e1	1.25e1	1.20e1	2.00
Gnutella	3226	4.21e0	8.78e-1	7.88e-1	1.67e0	1.72e0	1.89e0	1.85e0	1.77e0	1.90
FA	4845	1.27e1	2.36e0	2.00e0	4.36e0	4.41e0	4.98e0	4.89e0	4.66e0	1.85
wiki-RfA	2449	4.22e1	4.43e-1	4.22e-1	8.47e-1	8.30e-1	9.79e-1	9.56e-1	9.35e-1	1.91
ca-HepTh*	8638	5.75e0	9.57e0	9.80e0	2.04e1	2.02e1	3.51e1	3.52e1	3.49e1	2.14

In Table 4.3 we notice that the Gauss-Jordan - L^d Algorithm (Algorithm 2) and the Bottleneck Algorithm (Algorithm 3) have comparable execution times and are faster than the $\mathcal{LDU}$-L^d method presented in [5, Algorithm 1]. Again, the bottleneck approach has a slight edge on a majority of test cases.

Now we present some tests aimed at assessing the numerical stability of the methods introduced in this paper; see [4]. We consider the computation of the stationary distribution of irreducible Markov chains with transition matrix P. This is the unique positive row vector with $\|u\|_1 = 1$ such that $u = uP$. While the matrix $I_n - P^T$ is not strictly speaking a (normalized) graph Laplacian, it is a singular M-matrix of rank $n-1$.

Test problem 4. Next, we examine the second test problem tested in [4], which is taken from [11]. This is a parameterized family of 10×10 matrices with varying degree of coupling. Let

$$T = \begin{bmatrix} 0.1 & 0.3 & 0.1 & 0.2 & 0.3 & \beta & 0 & 0 & 0 & 0 \\ 0.2 & 0.1 & 0.1 & 0.2 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.2 & 0.2 & 0.4 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0.2 & 0.1 & 0.2 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0.3 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & 0 & 0 & 0.1 & 0.2 & 0.2 & 0.4 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0.2 & 0.1 & 0.3 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0.3 & 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0.2 & 0.1 & 0.3 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0.7 & 0 & 0 & 0.2 \end{bmatrix}.$$

ETNA
Kent State University and Johann Radon Institute (RICAM)

If $\Delta = \operatorname{diag}\left(\frac{1}{1+\beta}, 1, 1, 1, 1, \frac{1}{1+\beta}, 1, 1, 1, 1\right)$, then $P = \Delta T$ is a nearly uncoupled stochastic matrix with degree of coupling $\gamma = \frac{\beta}{1+\beta}$. Let u be the stationary vector, i.e., a vector in $\operatorname{Ker}(I_{10} - P^T)$. Let $\tilde{u}$ be the computed stationary vector by using the Gauss-Jordan Algorithm or the Bottleneck Algorithm. Two values of β are used.
(**a**) For $\beta = 10^{-7}$ the exact stationary vector is given by (see [4])

$$u = \begin{bmatrix} 0.1008045195787271\text{e+0} \\ 0.8012666139606563\text{e-1} \\ 0.3015519514905696\text{e-1} \\ 0.6031039029811392\text{e-1} \\ 0.7926508439180686\text{e-1} \\ 0.1008045195787271\text{e+0} \\ 0.1967651659427390\text{e+0} \\ 0.7003949685919185\text{e-1} \\ 0.1619417899342436\text{e+0} \\ 0.1197871768713281\text{e+0} \end{bmatrix}.$$

(**b**) For $\beta = 10^{-14}$ the exact stationary vector is given by

$$u = \begin{bmatrix} 0.1008045115305868\text{e+0} \\ 0.8012666301149126\text{e-1} \\ 0.3015519575701284\text{e-1} \\ 0.6031039151402568\text{e-1} \\ 0.7926508598986232\text{e-1} \\ 0.1008045115305868\text{e+0} \\ 0.1967651699097019\text{e+0} \\ 0.7003949827125116\text{e-1} \\ 0.1619417931991359\text{e+0} \\ 0.1197871792863454\text{e+0} \end{bmatrix}.$$

In Table 4.4 we can see the residual $\|\tilde{u}^T - \tilde{u}^T P\|_1$ and the absolute error $\|u - \tilde{u}\|_1$ for $\beta = 10^{-7}$ and $\beta = 10^{-14}$.

TABLE 4.4
Results for test problem 4.

	Method	**Residual**	**Error**
$\beta = 10^{-7}$	Gauss-Jordan	1.2490e-16	2.0817e-16
	Bottleneck	1.1796e-16	3.6826e-10
$\beta = 10^{-14}$	Gauss-Jordan	1.1796e-16	2.0470e-16
	Bottleneck	6.9389e-17	3.7641e-3

Note that the errors using the Bottleneck Algorithm are large compared to those given by the Gauss-Jordan method proposed in this work. This is due to the fact that the `inv` function in the Bottleneck Algorithm employs the *LU* factorization of the explicitly computed matrix $I_{10} - P^T$, which is not accurate enough for this problem [4]. High accuracy cannot easily be attained with this approach, since it is not possible to compute the pivots using the off-diagonal entries in the active submatrix at each step.

Test problem 5. Next, we examine the third test problem of [4], which is taken from [24]. We consider the tridiagonal stochastic matrix P determined (uniquely) by

$$p_{i+1,i} = 0.8 \quad \text{and} \quad p_{i,i+1} = 0.1, \quad \text{for } i = 1, 2, \dots, n-1.$$

Thus $p_{ii} = 0.1$ for all i except for $p_{11} = 0.9$ and $p_{nn} = 0.2$. The exact stationary vector u is considered as the vector with analytically computed entries $u_i = 2^{3(n-i)}$.

In Table 4.5 we see the residual $\|\tilde{u}^T - \tilde{u}^T P\|_1$ and the absolute error $\|u - \tilde{u}\|_1$, where $\tilde{u}$ is computed by using the Gauss-Jordan Algorithm or the Bottleneck Algorithm.

TABLE 4.5
Results for test problem 5.

Method	Residual	Error
Gauss-Jordan	4.3616e-17	1.2688e-16
Bottleneck	2.4537e-17	1.2091e-15

In Table 4.5 we notice that the (absolute) normwise error using the Bottleneck Algorithm is small enough. However, in fact, through this algorithm the last components of the computed stationary vector are affected by large relative componentwise errors. In particular, the last entry of the computed stationary vector is negative, with no significant digit computed correctly. On the other hand, the components of the computed stationary vector via the Gauss-Jordan method are exact to working precision. In Table 4.6 we report the six last entries of the exact stationary vector and the computed ones via the Gauss-Jordan method and the Bottleneck Algorithm.

TABLE 4.6
The six last entries of the stationary vector for test problem 5.

Exact u	ũ via G-J	ũ via Bottl.
0.198951966012828e-12	0.198951966012828e-12	0.198917271543309e-12
0.024868995751604e-12	0.024868995751604e-12	0.024834301282084e-12
0.003108624468950e-12	0.003108624468950e-12	0.003073929999431e-12
0.000388578058619e-12	0.000388578058619e-12	0.000353883589099e-12
0.000048572257327e-12	0.000048572257327e-12	0.000013877787808e-12
0.000006071532166e-12	0.000006071532166e-12	-0.000028622937354e-12

5. Concluding remarks.

In this paper, we focused on two direct numerical methods for the computation of the group inverse and the absorption inverse of the Laplacian matrix. First, we studied a method based on a modification of the Gauss-Jordan algorithm adjusted to the case of the Laplacian matrix. This GTH-like variant is forward stable and results in computed solutions with high relative componentwise accuracy. In particular, using the fact that the M-matrix property is preserved throughout the steps of the algorithm, at each step the diagonal entries of the matrix are computed by adding the nondiagonal entries in the same column of the active submatrix and then changing the sign, avoiding in this way any cancellation error that may occur. Also, a second approach was proposed, which employs the bottleneck matrix. Forward error bounds were presented for each of these two approaches.

The developed methods exhibit comparable complexity with that of standard algorithms existing in the literature; furthermore, all the stability issues concerning the implementation of these algorithms are well understood.

The proposed methods have been compared in terms of their execution time with the algorithm presented in [5] on various networks, directed and undirected. We notice that the proposed algorithms display comparable execution times, which are better than those attained using the LU factorization. Comparing the Gauss-Jordan-based algorithm and the algorithm presented in [5], which is based on the LU factorization of the Laplacian matrix, we note that despite the fact that the computation of the LU factorization is faster than the Gauss-Jordan elimination process, the remaining part of the algorithm (Algorithm 2) requires less computational effort than those of [5, Algorithm 1], leading to a faster total execution time.

The numerical stability of the proposed algorithms was also studied experimentally and it was illustrated in several numerical examples. We conclude that the Gauss-Jordan-based algorithm is able to attain high relative componentwise accuracy, while the bottleneck matrix approach is not.

Acknowledgments. The authors would like to thank an anonymous referee for his/her valuable suggestions.

REFERENCES

[1] Biological Networks Data Sets of Newcastle University, available online at http://www.biological-networks.org/.
[2] K. M. ANSTREICHER AND U. G. ROTHBLUM, *Using Gauss-Jordan elimination to compute the index, generalized nullspaces, and Drazin inverse*, Linear Algebra Appl., 85 (1987), pp. 221–239.
[3] J. BAGLAMA, C. FENU, L. REICHEL, AND G. RODRIGUEZ, *Analysis of directed networks via partial singular value decomposition and Gauss quadrature*, Linear Algebra Appl., 456 (2014), pp. 93–121.
[4] M. BENZI, *A direct projection method for Markov chains*, Linear Algebra Appl., 386 (2004), pp. 27–49.
[5] M. BENZI, P. FIKA, AND M. MITROULI, *Graphs with absorption: numerical methods for the absorption inverse and the computation of centrality measures*, Linear Algebra Appl., 574 (2019), pp. 123–152.
[6] A. BERMAN AND R. J. PLEMMONS, *Nonnegative Matrices in the Mathematical Sciences*, SIAM, Philadelphia, 1994.
[7] S. L. CAMPBELL AND C. D. MEYER, *Generalized Inverses of Linear Transformations*, Pitman, Boston, 1979.
[8] F. CHUNG, *Spectral Graph Theory*, AMS, Providence, 1997.
[9] ———, *Laplacians and the Cheeger inequality for directed graphs*, Ann. Comb., 9 (2005), pp. 1–19.
[10] T. DAVIS AND Y. HU, *The SuiteSparse Matrix Collection*, https://sparse.tamu.edu/.
[11] T. DAYAR, *Stability and Conditioning Issues on the Numerical Solution of Markov Chains*, PhD. Thesis, Computer Science/Physical & Math Science, North Carolina State University Raleigh, Raleigh, 1994.
[12] R. E. FUNDERLIC AND R. J. PLEMMONS, *LU decomposition of M-matrices by elimination without pivoting*, Linear Algebra Appl., 41 (1981), pp. 99–110.
[13] GEPHI, *Gephi Sample Data Sets*. http://wiki.gephi.org/index.php/Datasets
[14] S. A. GOREINOV, I. V. OSELEDETS, D. V. SAVOSTYANOV, E. E. TYRTYSHNIKOV, AND N. L. ZAMARASHKIN, *How to find a good submatrix*, in Matrix Methods: Theory, Algorithms and Applications, V. Olshevsky and E. Tyrtyshnikov, eds., World Scientific, Singapore, 2010, pp. 247–256,
[15] W. K. GRASSMANN, M. I. TAKSAR, AND D. P. HEYMAN, *Regenerative analysis and steady state distributions for Markov chains*, Oper. Res., 33 (1985), pp. 1107–1116.
[16] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.
[17] J. HUNTER, *The computation of the mean first passage times for Markov chains*, Linear Algebra Appl., 549 (2018), pp. 100–122.
[18] K. A. JACOBSEN AND J. H. TIEN, *A generalized inverse for graphs with absorption*, Linear Algebra Appl., 537 (2018), pp. 118–147.
[19] S. J. KIRKLAND AND M. NEUMANN, *Group Inverses of M-matrices and Their Applications*, CRC Press, Boca Raton, 2013.
[20] S. J. KIRKLAND, M. NEUMANN, AND B. L. SHADER, *Distances in weighted trees and group inverse of Laplacian matrices*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 827–841.
[21] J. MA AND Y. LI, *Gauss–Jordan elimination method for computing all types of generalized inverses related to the $\{1\}$-inverse*, J. Comput. Appl. Math., 321 (2017), pp. 26–43.
[22] J. MA, F. GAO, AND Y. LI, *An efficient method for computing the outer inverse $A^{(2)}_{T,S}$ through Gauss-Jordan elimination*, Numer. Algorithms, online publication, 2019. https://doi.org/10.1007/s11075-019-00803-w.

[23] A. N. MALYSHEV, *A note on the stability of Gauss-Jordan elimination for diagonally dominant matrices*, Computing, 65 (2000), pp. 281–284.

[24] C. A. O'CINNEIDE, *Relative-error bounds for the LU decomposition via the GTH algorithm*, Numer. Math., 73 (1996), pp. 507–519.

[25] THE MATHWORKS, *Matlab Documentation*.
https://www.mathworks.com/help/matlab/ref/inv.html.

[26] J. H. TIEN, Z. SHUAI, M. C. EISENBERG, AND P. VAN DEN DRIESSCHE, *Disease invasion on community networks with environmental pathogen movement*, J. Math. Biol., 70 (2015), pp. 1065–1092.

[27] G. WANG, Y. WEI, AND S. QIAO, *Generalized Inverses: Theory and Computations*, Science Press, Beijing, 2004.

[28] J. H. WILKINSON, *Rounding Errors in Algebraic Processes*, Prentice Hall, Englewood Cliffs, 1963.