Science & Technology Facilities Council

Preprint

RAL-P-2018-004

Uncovering hidden block structure

IS Duff, PA Knight, L le Gorrec, S Mouysset, D Ruiz

August 2018

Submitted for publication in SIAM Journal on Mathematics of Data Science

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

Uncovering Hidden Block Structure

Iain S. Duff[1], Philip A. Knight[2], Luce le Gorrec[3], Sandrine Mouysset[3] and Daniel Ruiz[3]

ABSTRACT

We develop a multistage procedure for uncovering the block structure in a matrix. Our algorithm combines standard combinatorial techniques with a novel clustering approach that merges both numerical and structural analysis. A central part of our process is to use a doubly stochastic scaling. We illustrate the use of our algorithm in partitioning sparse matrices for constructing a preconditioner for iterative methods. We also show how we can use our approach in community detection in both undirected and directed networks.

Keywords: doubly stochastic scaling, edge detection, partitioning sparse matrices, block preconditioning, community detection.

AMS(MOS) subject classifications: 65F30, 65F50

---

[1]Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0QX, UK. iain.duff@stfc.ac.uk. Also Cerfacs, Toulouse, France. duff@cerfacs.fr.

[2]University of Strathclyde, Glasgow, Scotland, UK. p.a.knight@strath.ac.uk.

[3]Université de Toulouse - IRIT, 2 rue Camichel, Toulouse, France. Sandrine.Mouysset@irit.fr, ruiz@enseeiht.fr, luce.legorrec@enseeiht.fr.

Also published as CERFACS Technical Report TR/PA/18/90.

August 9, 2018

Contents

1 Introduction

Our aim is to discover block structure in a matrix by taking into account both the size of the entries and the structure of (significant) nonzeros. We develop a multistage process for doing this.

Our algorithm permutes and scales the matrix $\mathbf{A}$ so that it can be partitioned into the block triangular form

$$\begin{bmatrix} \mathbf{B}_1 & & * \\ & \ddots & \\ 0 & & \mathbf{B}_k \end{bmatrix}, \text{ with } \mathbf{B}_i = \begin{bmatrix} \mathbf{D}_i & \mathbf{G}_i \\ \mathbf{H}_i & \mathbf{C}_i \end{bmatrix} \tag{1.1}$$

where

- $\mathbf{D}_i$ is (strongly) diagonally dominant.
- $\mathbf{G}_i$ and $\mathbf{H}_i$ have very small norm.
- $\mathbf{C}_i$ is such that both $\mathbf{C}_i\mathbf{C}_i^T$ and $\mathbf{C}_i^T\mathbf{C}_i$ have a near block diagonal structure although the blocking will be different in each case.

Our method can be divided into three main phases: preprocessing and scaling, SVD and step detection, and clustering improvement together with postprocessing for particular applications, where we note that we might iterate over the last two phases. We describe each of these aspects in the subsequent sections before discussing the effectiveness on standard test problems and the applications for our matrix partitioning in Section 8. We present our conclusions in Section 9.

A crucial part of our approach is to use the scaling algorithms of [25] to scale the blocks in (1.1) so that $|\mathbf{B}_1|, |\mathbf{B}_2|, \ldots |\mathbf{B}_k|$ are all doubly stochastic, that is that all column and row sums are equal to one. We discuss this more in Section 2, where we highlight useful properties of the singular vectors of doubly stochastic matrices in identifying block structure.

Our aim is to be able to apply our algorithm to both symmetric and unsymmetric matrices. This leads to a number of challenging issues regarding the rearrangement of the blocks. These are discussed in Section 7. In particular, we develop a bespoke measure of block quality based on modularity to validate our proposed partitioning.

The link between spectral and structural properties of matrices is a motivating factor behind many existing clustering algorithms [15, 39], for example, the work on obtaining a partitioning using the signs of the entries of the Fiedler vector [32]. Other authors [18, 28] have proposed the use of singular vectors in (bi-)clustering but, by working with the SVD of a doubly stochastic scaling, we are able to develop a versatile method with many attractive features. In particular, we only need the computation of a few singular vectors (sometimes one is enough) to obtain significant information on structure, and we need no prior information on the number of blocks potentially hidden in the structure. We describe how this is done in Sections 5 and 6.

2 Scaling to doubly stochastic form

As we mentioned in the introduction, a central part of our algorithm and analysis is that we scale the matrices $|\mathbf{B}_i|$ so that they are doubly stochastic. That is for each block $\mathbf{B}_i$, we find diagonal matrices $\mathbf{D}$ and $\mathbf{F}$ so that all the row sums and column sums of the scaled matrix $\mathbf{D}|\mathbf{B}_i|\mathbf{F}$ are equal to one. By equilibrating the row and column sums of our input matrix we are able to make certain deductions about potential block structures. We note that there is no loss in generality in scaling $|\mathbf{B}_i|$ rather than $\mathbf{B}_i$ as we are only concerned with the structure of the matrix and the size of its entries and so the sign of an entry is immaterial.

While for positive matrices (that are of course dense) such a scaling exists, it may be impossible to scale a sparse matrix all of whose entries are non-negative to doubly stochastic form. Fortunately, the

conditions for existence of a doubly stochastic scaling with respect to the zero pattern are well understood, and can be checked relatively easily. Before continuing, we give some background properties concerning doubly stochastic scaling.

A square matrix, $\mathbf{A}$, is said to have *support* if there is a perfect matching in the associated bipartite graph and *total support* if each edge in its bipartite graph can be included in a perfect matching. It is called bi-irreducible or fully indecomposable if it is impossible to find permutation matrices $\mathbf{P_r}$ and $\mathbf{P_c}$ such that

$$\mathbf{P_r A P_c} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{O} \\ \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix}$$

with $\mathbf{A}_1$ and $\mathbf{A}_3$ square. Such a matrix has the property known as the strong Hall property. Fully indecomposable matrices have total support but the reverse is not true as a block diagonal matrix is not fully indecomposable but can have total support. More formal definitions of support, total support, and full indecomposability, can be found in the book by Brualdi and Ryser [6, Ch.3 and Ch. 4].

The following theorem is due to Sinkhorn and Knopp [40].

Theorem 1 *If $\mathbf{A} \in \mathbb{R}^{n\times n}$ is nonnegative then a necessary and sufficient condition that there exists a doubly stochastic matrix $\mathbf{P}$ of the form $\mathbf{DAF}$ where $\mathbf{D}$ and $\mathbf{F}$ are diagonal matrices with positive main diagonals is that $\mathbf{A}$ has total support. If $\mathbf{P}$ exists then it is unique. $\mathbf{D}$ and $\mathbf{F}$ are also unique up to a scalar multiple if and only if $\mathbf{A}$ is fully indecomposable.*

A matrix $\mathbf{A}$ that has support but not total support cannot be scaled to a doubly stochastic matrix. However, even in this case, we note that the classical scaling algorithms to doubly stochastic form produce a doubly stochastic matrix corresponding to the largest submatrix of $\mathbf{A}$ with total support. However, the scaling factors $\mathbf{D}$ and $\mathbf{F}$ have some entries that either converge to zero or to infinity [40]. This fact is also seen in more recent work [24, 26, 38]. We note that if $\mathbf{A}$ is symmetric then a doubly stochastic scaling is guaranteed if its diagonal contains no zeros (since we can symmetrically permute any nonzeros a_{ij} and a_{ji} onto the diagonal).

We have two principal reasons for applying a doubly stochastic scaling. The first, which is somewhat qualitative, is that the action of our scaling to doubly stochastic form can emphasise the interdependence between the size of an entry and the interconnections with respect to other entries. For example, a large entry after scaling implies that all other entries in its row and column are small. It is important that the scaling we use is global. In contrast, if we were just to balance row sums, say, we could do this in a localised manner by looking at each row of the matrix independently but the scaled matrix would not exhibit the properties that we observe above or that we need in the following. The second reason for the scaling is quantitative. As we show below, spectral properties of a doubly stochastic matrix can be used to determine structural properties.

Probably the best known algorithm for finding a scaling to doubly stochastic form is the Sinkhorn–Knopp algorithm [40][1] but this can be slow, especially for sparse matrices, and so we use the Newton method described in [25]. Typically, this is very cheap to do, requiring only matrix–vector products.

We now discuss some fundamental spectral properties of doubly stochastic matrices. The following theorem is a direct consequence of the Perron–Frobenius theorem [19, 35].

Theorem 2 *Suppose that $\mathbf{S} \in \mathbb{R}^{n\times n}$ is symmetric, irreducible, and doubly stochastic. The largest eigenvalue of $\mathbf{S}$ is 1, with multiplicity 1, and the associated eigenvector is equal to a multiple of $\mathbf{e}$, the vector of ones. Furthermore, if $\mathbf{S}$ is reducible (but has total support) and can be permuted symmetrically into a block diagonal structure with k blocks, then the eigenvalue 1 has multiplicity k. Assuming that these blocks are of size $n_1, n_2, \ldots, n_k$ then a basis for the corresponding eigenvectors is*

$$\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\},$$

[1] The algorithm has a long history that pre-dates Sinkhorn and Knopp but it is commonly known by this name in linear algebra circles. For more details see [24].

where v_{pq} for $p \in \{1, \ldots, k\}$ is given by

$$v_{pq} = \begin{cases} 1, & \text{for all } q \text{ in block } p \\ 0, & \text{otherwise.} \end{cases}$$

In other words, if we partition the rows and columns of $\mathbf{S}$ *according to the blocks, then the components of the pth eigenvector corresponding to those rows have constant value.*

As a corollary, consider an unsymmetric but still doubly stochastic matrix $\mathbf{P}$, then both normal equation matrices $\mathbf{P}\mathbf{P}^T$ and $\mathbf{P}^T\mathbf{P}$ are also doubly stochastic, and the results of Theorem 2 apply. Suppose then that we compute the principal left and right singular vectors of an unsymmetric doubly stochastic matrix. If any of the two normal equations matrices exhibit some block structure, then the computed singular vectors are likely to have contributions from each (or, at least, several) of the corresponding basis vectors, but since these vectors are formed by a disjoint partition of $\{1, \ldots, n\}$, we can identify the precise contribution from each block. Reordering the singular vectors according to the block structure will arrange them in a piecewise constant form. In theory, a single such vector could reveal the whole block structure and it is this property that we exploit in our algorithm to get this structure information from only an apparently small amount of information.

The rationale here is as follows. If we compute a singular vector associated with the (k times repeated) singular value 1 then it will be of the form

$$\mathbf{x} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k.$$

It is reasonable to assume that $\mathbf{x}$ is selected randomly from the space spanned by the $\mathbf{v}_i$ (for example, an iterative method to compute $\mathbf{x}$ might be seeded with a random initial vector). If this is the case it is also reasonable to assume that the a_i are distinct. At any rate, to avoid the trivial case given by the vector $\mathbf{e}$ of all ones, which is both a left and right maximal singular vector for a doubly stochastic matrix, we may work only with maximal singular vectors that are in the orthogonal complement of the constant vector $\mathbf{e}$. This will enforce conditions for the a_i to be almost surely distinct. We can then characterise the partitions exactly using the set

$$\{a_1, \ldots, a_k\}.$$

In practice we do not have matrices with perfect block structure (and indeed if they were so, our sophisticated machinery would not be needed to discover the blocking). However, if the matrix has a structure that is close to block diagonal, then the leading singular vectors can be assumed to have a structure similar to the piecewise constant form. If we then look for steps in the computed vectors we should be able to reveal some underlying near block structure, or at least some row blocks (respectively column blocks) that have low correlation with each other.

There are clear connections between the singular vectors we work with and the Fiedler vector/eigenvectors used in other spectral approaches. In fact, if our doubly stochastic matrix is $\mathbf{S}$ ($\mathbf{P}^T\mathbf{P}$ or $\mathbf{P}\mathbf{P}^T$) then its weighted Laplacian is defined as $\mathbf{L}$ where $l_{ij} = -s_{ij}, i \neq j$ and $l_{ii} = \sum_{i \neq j} s_{ij} = 1 - s_{ii}$ so that $\mathbf{L}$ and $\mathbf{S}$ are such that

$$\mathbf{L} = \mathbf{I} - \mathbf{S}$$

and the identification of our first non-trivial eigenvector and the Fiedler vector is clear.

The weighted Laplacian is also used by [30] to determine a weighted bandwidth minimisation and by [18], where the authors split rows and columns, as we do, but only into bipartitions in a style akin to the use of the Fiedler vector. An analogue of the Fiedler vector is also used in [42] to obtain a partitioning of the matrix which both provides a good block diagonal preconditioner and is easily parallelizable. But once again, the authors use this vector to separate the matrix into only two blocks. What is different in our approach is that we use the Fiedler vector not just to separate the rows or columns into two parts but to discover directly many piecewise constant subcomponents.

To conclude this discussion about the motivations for scaling, and the peculiarities of the doubly stochastic case we use a small example in Figure 2.1 to illustrate how the spectral information may vary

when extracting eigenvectors from either the Laplacian or from the doubly stochastic matrix. In this example, there is a graph with three distinct heterogeneous clusters that are loosely linked together. Our doubly stochastic matrix is obtained by setting the diagonal of the adjacency matrix of the graph to 10^{-8} times the identity matrix, before scaling to doubly stochastic form. Indeed, a zero diagonal may prevent the convergence of the scaling algorithm and since the graph Laplacian fills the diagonal, too, this is not an unreasonable approach. In the lower half of Figure 2.1 we see the numerical structure of the eigenvectors that can be used to identify the clusters. For the Laplacian we have illustrated the eigenvectors of the two smallest positive eigenvalues (the Fiedler vector in blue) and for the stochastic scaling we have used the two subdominant vectors. In the case of the Laplacian matrix, we can see that the eigenvectors cannot easily resolve the clusters (the same is true for the normalised Laplacian, too). On the other hand, for the doubly stochastic scaling we can use either of the vectors to resolve the clusters perfectly and unambiguously.

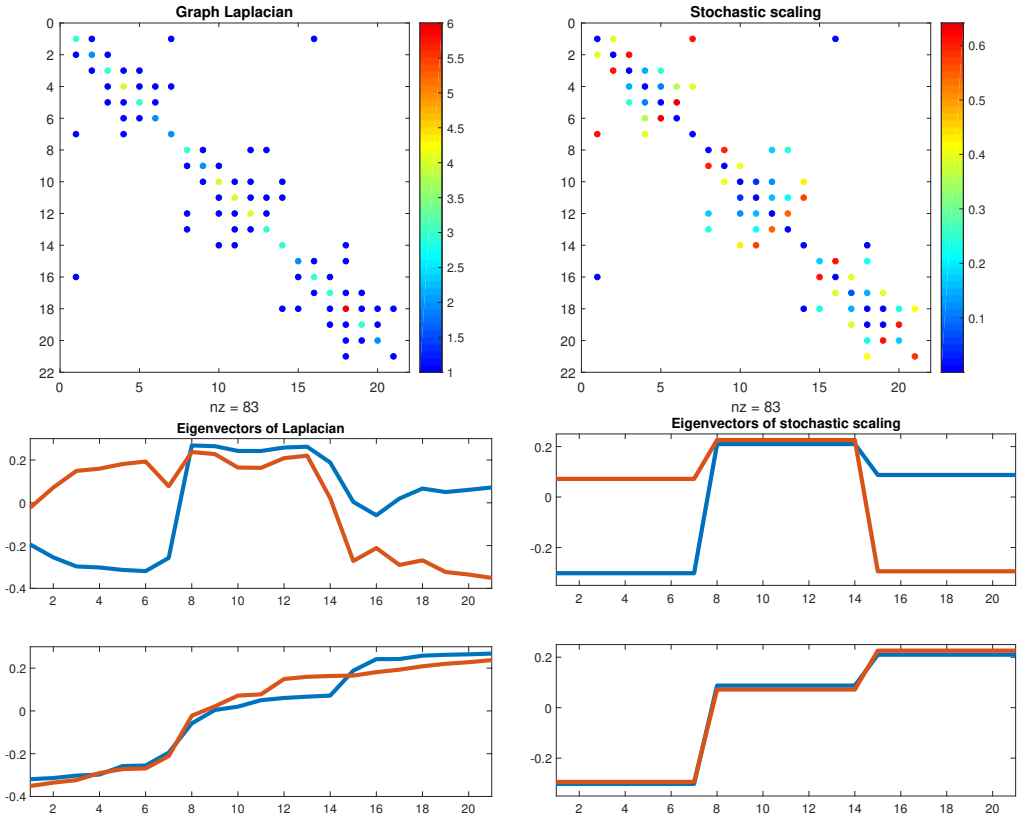


Figure 2.1: Two matrices corresponding to a graph with 3 clusters: the Laplacian of the graph (top left), and the corresponding doubly stochastic scaling (top right). The bottom left plot shows the numerical values of the two eigenvectors associated with the smallest positive eigenvalues of the Laplacian, and the bottom right plot shows the two subdominant eigenvectors of the scaled adjacency matrix (the top subplot shows the numerical values in natural order, and the bottom subplot after sorting the entries in increasing order).

2.1 Perturbation theory

To get a better understanding of the potential of the singular vectors when we have an imperfect block structure we examine the effects of small perturbations on the doubly stochastic form. When looking for hidden block structure it is appropriate to assume that $\mathbf{A}$ is a block diagonal matrix but this introduces some subtleties to the analysis. For clarity, we start by assuming that $\mathbf{A}$ is fully indecomposable (and

therefore is a single block), but we will then show that this can be relaxed to matrices with total support which means we can extend our analysis to include matrices with a perfect block structure.

Suppose then that $\mathbf{A} \geq 0$ can be diagonally scaled to doubly stochastic form according to the formula

$$\mathbf{P} = \mathbf{RAC}.$$

Now consider a small perturbation to $\mathbf{A}$, namely

$$\widetilde{\mathbf{A}} = \mathbf{A} + \epsilon\mathbf{H},$$

where ϵ is small and $\|\mathbf{H}\| = 1$ (it is convenient to use the 2-norm in this perturbation analysis but it can readily be adapted to other matrix norms). We assume that $\widetilde{\mathbf{A}}$ also has total support. Then we can give conditions that ensure that the scaling of $\widetilde{\mathbf{A}}$ to doubly stochastic form is close to $\mathbf{P}$. We use the notation $\mathcal{D}(\mathbf{v})$ to represent a diagonal matrix whose diagonal entries are given by the vector $\mathbf{v}$.

Theorem 3 *Let $\mathbf{A}, \widetilde{\mathbf{A}}, \mathbf{R}, \mathbf{C}, \mathbf{H}$ and $\mathbf{P}$ be as described above and let $\widetilde{\mathbf{P}} = \mathbf{P} + \epsilon\mathbf{Q}$ where $\mathbf{Q} = \mathbf{RHC}$. Then we can find vectors $\mathbf{f}$ and $\mathbf{g}$ such that $\|\mathbf{f}\| = O(\epsilon)$, $\|\mathbf{g}\| = O(\epsilon)$ and, if $\mathbf{F} = \mathcal{D}(\mathbf{e} + \mathbf{f})$, $\mathbf{G} = \mathcal{D}(\mathbf{e} + \mathbf{g})$, then*

$$\begin{bmatrix} \mathbf{F}\widetilde{\mathbf{P}}\mathbf{G}\mathbf{e} \\ \mathbf{G}\widetilde{\mathbf{P}}^T\mathbf{F}\mathbf{e} \end{bmatrix} = \mathbf{e} + \mathbf{m}, \tag{2.1}$$

where $\|\mathbf{m}\| = O(\epsilon^2)$.

Proof We simply need to multiply out the left-hand side of (2.1). We do this one half at a time. Noting that $\mathbf{Pe} = \mathbf{e}$ we have

$$\begin{aligned} \mathbf{F}\widetilde{\mathbf{P}}\mathbf{G}\mathbf{e} &= \widetilde{\mathbf{P}}\mathbf{e} + \widetilde{\mathbf{P}}\mathbf{g} + \mathcal{D}(\mathbf{f})\widetilde{\mathbf{P}}\mathbf{e} + \mathcal{D}(\mathbf{f})\widetilde{\mathbf{P}}\mathbf{g} \\ &= \mathbf{e} + \epsilon\mathbf{Q}\mathbf{e} + \widetilde{\mathbf{P}}\mathbf{g} + \mathcal{D}(\mathbf{f})\mathbf{e} + \mathcal{D}(\mathbf{f})(\widetilde{\mathbf{P}}\mathbf{g} + \epsilon\mathbf{Q}\mathbf{e}) \\ &= \mathbf{e} + \epsilon\mathbf{Q}\mathbf{e} + \mathbf{f} + \mathbf{P}\mathbf{g} + \mathcal{D}(\mathbf{f})(\widetilde{\mathbf{P}}\mathbf{g} + \epsilon\mathbf{Q}\mathbf{e}) + \epsilon\mathbf{Q}\mathbf{g}. \end{aligned}$$

Note that the last two terms in the final line of these equalities are of size $O(\epsilon^2)$ and will form the top half of the vector $\mathbf{m}$ in (2.1). We need to show that there is a choice of $\mathbf{f}$ and $\mathbf{g}$ such that $\mathbf{f} + \mathbf{P}\mathbf{g} = -\epsilon\mathbf{Q}\mathbf{e}$. We can manipulate the bottom half of (2.1) similarly and we see that our theorem is proved so long as we can find $\mathbf{f}$ and $\mathbf{g}$ so that

$$\begin{bmatrix} \mathbf{I} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix} = -\epsilon \begin{bmatrix} \mathbf{Q}\mathbf{e} \\ \mathbf{Q}^T\mathbf{e} \end{bmatrix}. \tag{2.2}$$

This is not automatic as we know that the matrix on the left-hand side of (2.2) is singular (we call this matrix $\mathbf{P}_{sym}$). Its eigenvalues are of the form $1 \pm \sigma$ where σ is a singular value of $\mathbf{P}$ and since $\mathbf{P}$ is doubly stochastic it has a unit singular value. Now because $\mathbf{A}$ (and hence $\mathbf{P}$) is fully indecomposable, this unit singular value has multiplicity 1 and so $\mathbf{P}_{sym}$ has a one dimensional kernel whose entries are multiples of

$$\begin{bmatrix} \mathbf{e} \\ -\mathbf{e} \end{bmatrix}.$$

But

$$\begin{bmatrix} \mathbf{e} \\ -\mathbf{e} \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}\mathbf{e} \\ \mathbf{Q}^T\mathbf{e} \end{bmatrix} = \mathbf{e}^T\mathbf{Q}\mathbf{e} - \mathbf{e}^T\mathbf{Q}^T\mathbf{e} = 0$$

so (2.2) is a consistent system and a solution exists.

□

Remark The choice of $\mathbf{f}$ and $\mathbf{g}$ is motivated by the result of applying a step of the Newton method for balancing [25] to $\widetilde{\mathbf{P}}$ with initial vector $\mathbf{e}$.

The consequence of this theorem is that we can give precise conditions under which $\mathbf{A}$ and $\widetilde{\mathbf{A}}$ have nearby doubly stochastic scalings. We observe that $\mathbf{F}\widetilde{\mathbf{P}}\mathbf{G}$ is essentially doubly stochastic and $\|\mathbf{F}\widetilde{\mathbf{P}}\mathbf{G} - \mathbf{P}\|$ is small when $\|\mathbf{f}\|$ and $\|\mathbf{g}\|$ are small. This is true if $\|\mathbf{Q}\|$ is constrained and the smallest nonzero eigenvalue of $\mathbf{P}_{sym}$ is bounded away from zero. We argue that these are reasonable expectations in the context of uncovering block structure.

$\mathbf{P}_{sym}$ has a small nonzero eigenvalue if and only if the second largest singular value of $\mathbf{P}$ is close to 1. Now if $\mathbf{P}$ had a second singular value equal to 1 then this is indicative of a second block and we argue that if $\mathbf{P}$ has a singular value close to 1 then it is a small perturbation of a matrix with a competing block structure to that of our original matrix and so we can't expect small perturbations to give stable results.

$\|\mathbf{Qe}\| = \|\mathbf{RHCe}\|$ is large only if we can find values of the scaling matrices such that $r_i c_j \gg 1$. This is often observed if $\mathbf{A}$ is close to losing total support (in which case some of the scaling factors become extreme) but our preprocessing steps mitigate against this and in practice we do not expect $\mathbf{R}$ and $\mathbf{C}$ to blow up sufficiently to make $\|\mathbf{Qe}\|$ problematic in practice.

As stated earlier, we can extend our analysis to cover the case when $\mathbf{A}$ is not fully indecomposable but does have total support. This is precisely what we observe when $\mathbf{A}$ has perfect block structure and so it is vital that our perturbation analysis can handle such a case. We treat it separately as we note that the unit singular value of a doubly stochastic matrix has a multiplicity equal to the number of its diagonal blocks. Thus if $\mathbf{A}$ has several blocks the matrix $\mathbf{P}_{sym}$ in (2.2) has a kernel of dimension larger than one and so we need to do more work to show that the system is consistent. It is sufficient to consider the case when $\mathbf{A}$ has two blocks to describe how this is done. Suppose then that

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{P}_2 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{bmatrix}.$$

The kernel of $\mathbf{P}_{sym}$ is now spanned by the vectors

$$\mathbf{e}_x = \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{0} \\ -\mathbf{e}_1 \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{e}_y = \begin{bmatrix} \mathbf{0} \\ \mathbf{e}_2 \\ \mathbf{0} \\ -\mathbf{e}_2 \end{bmatrix},$$

where $\mathbf{e}_1$ and $\mathbf{e}_2$ are vectors of ones conforming with $\mathbf{P}_1$ and $\mathbf{P}_2$. We want to show that the right-hand side of (2.2) is still orthogonal to the kernel but

$$\begin{bmatrix} \mathbf{e}_x^T \\ \mathbf{e}_y^T \end{bmatrix} \begin{bmatrix} \mathbf{Qe} \\ \mathbf{Q}^T\mathbf{e} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1^T\mathbf{Q}_{12}\mathbf{e}_2 - \mathbf{e}_1^T\mathbf{Q}_{21}^T\mathbf{e}_2 \\ \mathbf{e}_2^T\mathbf{Q}_{21}\mathbf{e}_1 - \mathbf{e}_2^T\mathbf{Q}_{12}^T\mathbf{e}_1 \end{bmatrix}.$$

To make the right-hand side zero we need $\mathbf{e}_1^T\mathbf{Q}_{12}\mathbf{e}_2 = \mathbf{e}_2^T\mathbf{Q}_{21}\mathbf{e}_1$. Luckily, we have a degree of freedom which allows us to enforce this: in scaling $\mathbf{A}$ the diagonal factors for $\mathbf{P}_1$ and $\mathbf{P}_2$ are completely decoupled. These are each unique only up to a scaling factor (we can multiply the row scalings by a positive constant α and divide the column scalings by the same constant without affecting the row and column sums of the doubly stochastic matrix). Note that $\mathbf{Q}_{12}$ and $\mathbf{Q}_{21}$ are necessarily nonnegative. We can therefore pick a constant for the scaling of $\mathbf{P}_2$ independently from that of $\mathbf{P}_1$ to ensure that the right-hand side of (2.2) is orthogonal to the kernel of $\mathbf{P}_{sym}$ even when $\mathbf{A}$ is block diagonal. If there are more than two blocks then we can establish consistency recursively a block at a time.

In summary, we have shown that if we make a small perturbation to a matrix with block structure then it is reasonable to assume that the associated doubly stochastic matrices are also close together. We would like to draw corresponding conclusions about the corresponding singular vectors but so far we can only provide empirical evidence of this as exhibited in our examples (see, for example, Figure 4.1).

3 Preprocessing

Our algorithm is designed to be used on matrices which have an underlying block structure. Clearly, not every matrix has this property. Nevertheless, it may still be the case that we can find an effective block preconditioner for such cases. However it is essential to take some precautions to enhance the algorithm's chances of success. As we mention in the introduction, we first need to find the block triangular form (BTF) of our input matrix $\mathbf{A}$ to ensure that its diagonal blocks $\mathbf{B}_i$ are scalable to doubly stochastic form. To do this, we form a Dulmage–Mendelsohn decomposition [14] of the input matrix $\mathbf{A}$ and then work on its indecomposable diagonal blocks. The BTF can be computed efficiently by first finding a perfect matching and then finding the strongly connected components of a directed graph [37]. Algorithms for computing perfect matchings in shared memory systems have also been proposed [2].

Once we have identified and scaled the $\mathbf{B}_i$, we look for very large entries in order to create $\mathbf{D}_i$, $\mathbf{G}_i$ and $\mathbf{H}_i$ as defined in (1.1). The rationale for looking for these is that very dominant entries will result in blocks containing only one row or column. We would like to avoid spending time in discovering these very tiny blocks by means of spectral tools, first of all because these can be detected more or less easily at this stage. On top of that, such elementary blocks are represented by canonical vectors in terms of the Singular Value Decomposition, and hence can become hard to detect with signal processing tools such as those discussed in Section 5. In the general case, a simple approach for identifying the block $\mathbf{D}_i$ is to select entries in the scaled matrix which are larger than some threshold (0.75 for instance) and permute them to the diagonal. In the symmetric case, if we wish to preserve symmetry, we need to gather these entries around the diagonal and $\mathbf{D}_i$ will be block diagonal with blocks of order one or two. Nevertheless, in both cases, $\mathbf{G}_i$ and $\mathbf{H}_i$ are thus guaranteed to have small values relative to $\mathbf{D}_i$. The choice for the threshold value is not immutable but note that because of the doubly stochastic scaling, a node with a weight greater than 0.75 forces the sum of all remaining entries in its row and column sum below 0.25. We note also that a weight threshold value of 0.75 implies that the normal equations matrices (either $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$) will have a corresponding leading block with a weight of the order $0.75^2 \simeq 0.56$, which ensures diagonal dominance.

At the end of the preprocessing stage, we thus have clustered the matrix $\mathbf{A}$ into bi-irreducible blocks $\mathbf{B}_i$. Each of these doubly stochastic blocks $\mathbf{B}_i$ is itself split into two diagonal subblocks $\mathbf{C}_i$ and $\mathbf{D}_i$, where $\mathbf{D}_i$ is strongly diagonally dominant.

4 Projected SVD

As mentioned in Section 2, because the matrices we work on have imperfect block structure, it is important to consider the leading singular vectors in the orthogonal subspace of $\mathbf{e}$. If not, one may spend an iteration working on a vector which will not provide any information about the block structure of the matrix. By taking leading singular vectors of $\mathbf{P} - \frac{1}{n}\mathbf{e}\mathbf{e}^T$, we ensure that we work with the leading singular vectors of $\mathbf{P}$ which lie in the subspace orthogonal to $\mathbf{e}$. Additionally, if we have detected some dominant entries (as described in the previous section), we incorporate the corresponding set of canonical vectors into a basis $\mathbf{V}$ (that also includes the vector $\mathbf{e}$ in its range), and we project $\mathbf{P}$ onto the orthogonal complement of this basis $\mathbf{V}$ when computing the leading singular vectors.

If we consider now that we have fairly well defined blocks then the first k singular vectors should nearly span the subspace defined by the $\mathbf{v}_i$. In fact, we can find many blocks from only one singular vector. But we can also iterate with additional information to improve our proposed clustering. Each iteration requires new spectral information otherwise we will repeatedly discover the same blocks. An *a priori* approach is to take additional singular vectors of $\mathbf{P}$. However if we have missed some blocks[2] there is no guarantee that these blocks will reveal themselves as a matter of course. To ensure that the new vectors bring additional information we work instead with the leading eigenvectors of the projection of the normal equations of $\mathbf{P}$ into the subspace orthogonal to that of the blocks we have already identified. Accordingly, we augment

[2]See section 6 for ways to mitigate against this.

the span of the basis $\mathbf{V}$ with the set of characteristic vectors associated with the already identified clusters. We motivate this approach by considering an idealised example.

Let $\mathbf{S}$ be a doubly stochastic matrix with a perfect block diagonal structure ($\mathbf{S} = \mathbf{P}\mathbf{P}^T$ or $\mathbf{S} = \mathbf{P}^T\mathbf{P}$) so that

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & & \\ & \ddots & \\ & & \mathbf{S}_p \end{bmatrix},$$

where each $\mathbf{S}_k$ is doubly-stochastic and bi-irreducible.

Assume that we have found a partition of $\mathbf{S}$ into q blocks of respective size $c_1, \ldots, c_q$, which corresponds well with the real structure of $\mathbf{S}$, each block consisting of a number of distinct $\mathbf{S}_k$. Without loss of generality, we can assume that the ordering of these blocks matches the natural order of $\mathbf{S}$. Let us label these blocks $\mathbf{T}_1, \ldots, \mathbf{T}_q$, where each of the $\mathbf{T}_i$ is doubly stochastic. The projection of $\mathbf{S}$ into the orthogonal subspace of the blocks is given by

$$\mathbf{QSQ} = \begin{bmatrix} \mathbf{I}_1 - \frac{1}{c_1}\mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{I}_q - \frac{1}{c_q}\mathbf{J}_q \end{bmatrix} \begin{bmatrix} \mathbf{T}_1 & & \\ & \ddots & \\ & & \mathbf{T}_q \end{bmatrix} \begin{bmatrix} \mathbf{I}_1 - \frac{1}{c_1}\mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{I}_q - \frac{1}{c_q}\mathbf{J}_q \end{bmatrix},$$

where $\mathbf{I}_i$ and $\mathbf{J}_i = \mathbf{e}\mathbf{e}^T$ are of dimension c_i.

For each block i, we then have

$$\begin{aligned} \left(\mathbf{I}_i - \frac{1}{c_i}\mathbf{J}_i\right)\mathbf{T}_i\left(\mathbf{I}_i - \frac{1}{c_i}\mathbf{J}_i\right) &= \left(\mathbf{I}_i - \frac{1}{c_i}\mathbf{J}_i\right)\left(\mathbf{T}_i - \frac{1}{c_i}\mathbf{J}_i\right) \\ &= \mathbf{T}_i - \frac{1}{c_i}\mathbf{J}_i - \frac{1}{c_i}\mathbf{J}_i + \frac{1}{{c_i}^2}{\mathbf{J}_i}^2 \\ &= \mathbf{T}_i - \frac{1}{c_i}\mathbf{J}_i, \end{aligned}$$

using the fact that $\mathbf{T}_i$ is doubly stochastic and thus $\mathbf{T}_i\mathbf{J}_i = \mathbf{J}_i$ and $\mathbf{J}_i\mathbf{T}_i = \mathbf{J}_i$. Thus for each block $\mathbf{T}_i$, 1 is an eigenvalue with a multiplicity equal to the number of blocks of $\mathbf{S}$ which lie in $\mathbf{T}_i$. This leaves two possibilities.

- $\mathbf{T}_i$ coincides with only one block of $\mathbf{S}$. In this case, the largest eigenvalue of $\mathbf{T}_i - \frac{1}{c_i}\mathbf{J}_i$ is strictly less than 1 (see Theorem 2). Hence this block will not contribute to the leading eigenvector of $\mathbf{QSQ}$ and so we will not waste time re-identifying this block.
- $\mathbf{T}_i$ contains several blocks of $\mathbf{S}$. In this case, the principal eigenvector of $\mathbf{T}_i - \frac{1}{c_i}\mathbf{J}_i$ is the leading eigenvector of $\mathbf{T}_i$ that lies in the subspace orthogonal to $\mathbf{e}$, and it reveals each individual block in $\mathbf{S}$.

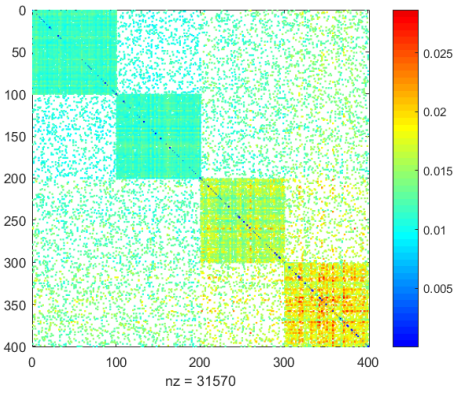
In summary, the leading eigenvector of the projection $\mathbf{QSQ}$ completes the recovery of all of the blocks whereas without the projection we would stagnate.

But we do not need such a perfect situation for our approach to work. Figure 4.1 presents a symmetric matrix $\mathbf{P}$ with 4 diagonal blocks. This is the doubly stochastic scaling of the adjacency matrix of a simple graph with 4 communities. These communities having intra-densities equal to 0.7, 0.7, 0.5 and 0.4, respectively and an inter-density of 0.07. Small ($O(10^{-8})$) diagonal entries are added before scaling to ensure that the Dulmage–Mendelsohn permutation maintains symmetry.

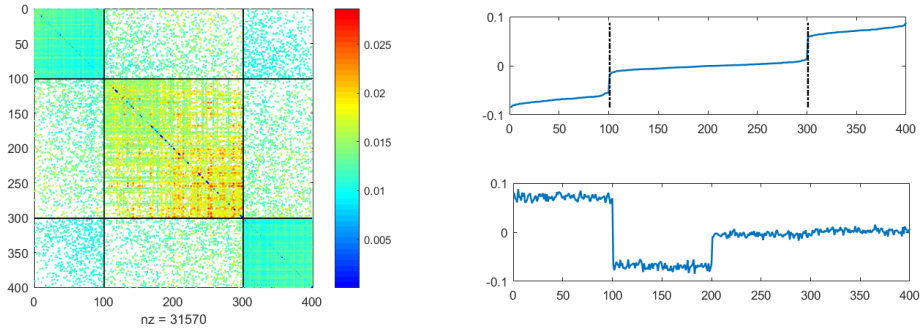
Using the subdominant eigenvector[3] of $\mathbf{P}$ we detect 3 blocks. The two blocks with the lowest intra-density remain conjoined (Figure 4.1 (b)). We then project $\mathbf{P}$ into the orthogonal subspace of the three blocks. The principal eigenvector of the projection is shown in Figure 4.1 (c). This clearly separates the two conjoined blocks from the previous iteration.

In Section 7 we describe how we can merge the two patterns illustrated in the left-hand sides of Figure 4.1(b) and (c) to uncover the true block structure.

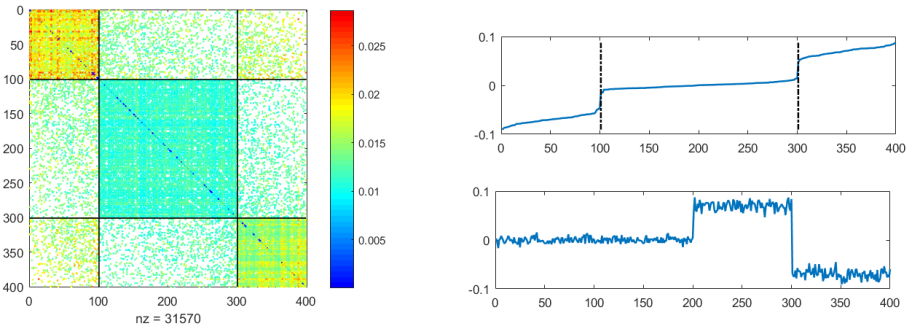
[3]The dominant eigenvector is $\mathbf{e}$.



(a)



(b)



(c)

Figure 4.1: Example of new information obtained from additional vectors. (a) Matrix with a block structure. In its natural order, we can see 4 blocks. (b) Bottom right : the second eigenvector in its natural order. Top right: the same vector sorted in ascending order. It detects two of the true blocks but merges the other two together. Left: the matrix permuted accordingly. (c) Bottom right : leading eigenvector of the projection of the initial matrix into the subspace orthogonal to the blocks highlighted in (b) in its natural order. Top right: the same vector sorted in ascending order. It separates the two blocks which were merged together in the previous step, while amalgamating the other two.

5 Cluster identification

A key stage in our algorithm is to identify clusters using the largest singular vectors of the doubly stochastic bi-irreducible matrix $\mathbf{P}$. If there are well defined clusters then we should be able to divide these vectors into nearly piecewise constant pieces, each equivalent to an underlying nearly block diagonal structure within one or other of the normal equation matrices $\mathbf{P}\mathbf{P}^T$ or $\mathbf{P}^T\mathbf{P}$. We note that a priori we are assuming no knowledge regarding the number of clusters nor the row and column permutations that may reveal the underlying block structure. Furthermore, we need to be aware that each singular vector we examine may suggest either a different number of clusters or clusters with different members. We now describe what we do to overcome these issues.

A first step is to reorder the computed singular vectors according to the size of their entries. Indeed, if our blocks were perfectly defined the piecewise constant steps would identify them unambiguously. Our problem is thus to detect steps in the reordered vector. A k-means approach [22] could be used to apply thresholding on the singular vector values but the performance of this method depends crucially on having the correct value of the parameter k to avoid overlooking some steps and also to avoid finding false positives. We have had much greater success in detecting edges by applying tools from signal processing, that means doing a convolution product between the current singular vector, seen as a signal where we want to detect the steps, and a filter. The peaks in the convolution product correspond to edges in the signal. These tools have the added bonus of not needing prior knowledge of the number of clusters.

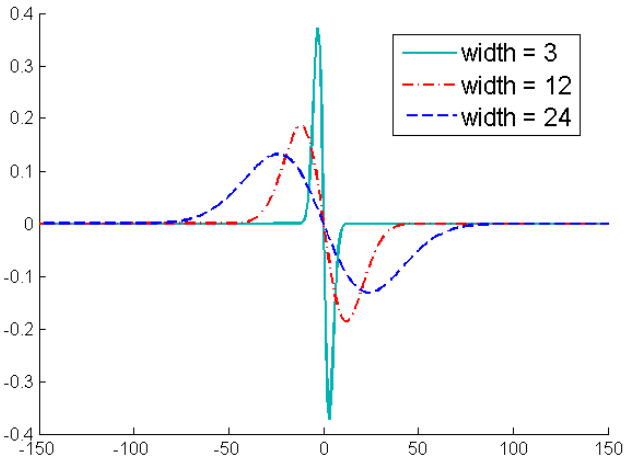
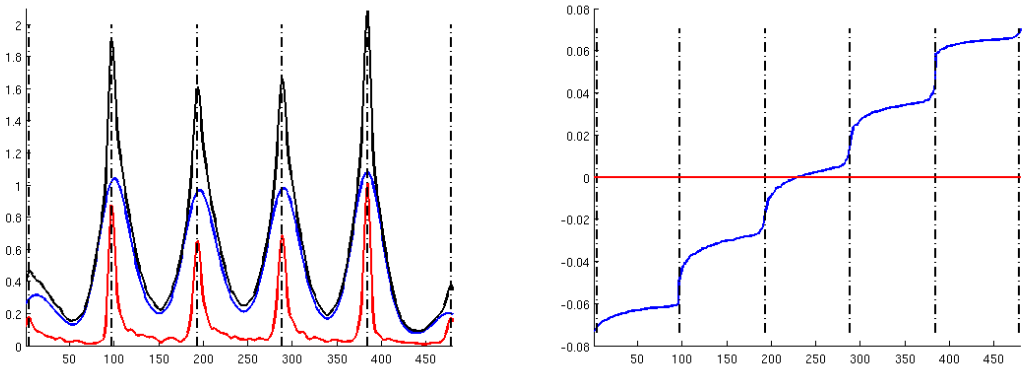


Figure 5.1: The first derivative of a Gaussian kernel for different sliding window sizes.

There are many types of filters. We have chosen to use the Canny filter [7], which is widely used in both signal and image processing for edge detection. To optimise edge detection it combines three performance criteria, namely good detection, good localization, and the constraint that an edge should correspond to a peak in an appropriate convolution. In order to satisfy these criteria the Canny filter uses an operator constructed from the convolution product of the output Signal-to-Noise Ratio (SNR) and a localization function (the filter). The optimal localization function is the first derivative of a Gaussian kernel [7].

Much effort has gone into refining our implementation of edge detection to make it optimal for our particular application. When using the filter for the convolution we can vary a parameter that we call the sliding window size. The window size determines whether the filter has a narrow support and has a steep profile (as happens with a small window) or a broad support and a smoother profile (with a big window), as shown in Figure 5.1. A small window can create many peaks whereas a larger one will detect fewer peaks.

To avoid the side effects of the window size on the convolution product the step detection is performed



(a) (b)

Figure 5.2: Example of step detection. (a) Convolution products with small (in red) and large (in blue) sliding window sizes for the Canny filter and their sum (in black). (b) Step detection on the right singular vector: in black dotted line the peaks found with the Canny filter. The red horizontal line indicates the partitioning found by the Fiedler vector.

with two window widths, large and small, with respect to the size of the vector. We then sum both convolution results in order to detect the principal peaks and to make sure that our steps are both sharp and of a relatively significant height.

Figure 5.2(b) shows an example of how the sliding window can affect the step detection using the 480×480 matrix `rbsb480` from the SuiteSparse collection [13]. We can see from this figure that the peaks corresponding to steps in the vector are much better identified through using this sum. In Figure 5.2(b), the associated right singular vector is plotted indicating the quality of the steps detected by the filters.

In contrast, the horizontal red line in Figure 5.2(b) indicates the bipartitioning suggested by the Fiedler vector. This presumes that there are precisely two clusters and the separation is not aligned with any of the steps indicated by the singular vector.

The figure indicates that the method has detected spurious clusters. Indeed, the first and the last are parasites caused by our prolongation of the vector. This prolongation is needed to avoid convolution product border effects. We also observe parasitic clusters if the matrix has a noisy block structure. In Section 6 we describe how to deal with this issue by validating peaks and how we incorporate additional information to resolve the boundaries between blocks with more precision.

6 Edge refinement

Once the convolution products of the singular vectors have been used to uncover a potential block structure we need to refine it to ensure the edges of our blocks are resolved sharply and are well localized. We also need to avoid including spurious artefacts. It is unreasonable to expect the filtering process to be completely effective in resolving edges as it assumes Gaussian noise and we compromise this when we reorder the singular vectors so that their entries are monotone increasing. Additionally, matrices from real-world applications do not exhibit the random deviation from block structure that we assume.

6.1 Iterative projection process

We note that the filtering process provides three distinct types of separation. The first corresponds to sharp edges in the vector. Unfortunately it is possible that the filter places these edges erroneously (though hopefully not far from their true location). This phenomenon can be seen in Figure 6.1(b). The second

type corresponds to smooth edges in the vector, and comes from the fact that we reorder a noisy vector. An example is provided in Figure 6.1(c). The separations of the third type are purely artefacts arising from local phenomena in the singular vector or from prolongation of a vector to apply a convolution product. Such a separation is presented in Figure 6.1(d).

Hence, we have designed a process to

- move the separations corresponding to the sharpest edges to their correct place,
- sharpen the smoother edges using additional information contained in other singular vectors,
- and remove spurious separations by means of a measure that evaluates the sharpness of an edge.

The goal of the process is to find a vector, $\mathbf{u}$, in the span of the singular vectors which is both close to the model staircase vector, $\mathbf{v}$, and to the initial vector. We achieve this with an iterative projection method between two spaces: the first space is fixed and is the span of the singular vectors at our disposal. The second is the span of the characteristic vectors of the current clustering—which are piecewise constant—and is modified through the iterations.

Projecting our initial vector $\mathbf{u}$ into the space of the proposed clustering gives us a staircase vector $\mathbf{v}$ where the ith constant part matches the ith cluster. This initial vector could be any of the singular vectors we have calculated. The value at each step in the staircase vector, α_i, is used as a characteristic value of the corresponding cluster. The mean of the characteristic values of adjacent clusters is then used as a separation threshold to reassign entries from the singular vector and this in turn gives updated clusters and corresponding basis vectors $\mathbf{v}_k$. If some entries have moved to another cluster we need to recompute the corresponding α_i and update $\mathbf{v}$ respectively. If we have computed only one singular vector we stop the process. Now, if there is more than one singular vector available, we project the updated $\mathbf{v}$ into the span of these singular vectors. We expect the contributions from other singular vectors to sharpen the steps in the resulting projected vector $\mathbf{u}$. Then the newly obtained projected vector $\mathbf{u}$ is normalised, and we iterate this process between $\mathbf{u}$ and $\mathbf{v}$ until the edges stabilise, which we recognise when changes in the norm of consecutive $\mathbf{u}$ vectors fall below a prescribed tolerance.

6.2 Spurious edge removal

At the end of the previous process we should obtain a vector which is closer to the staircase model than the initial singular vector. However, some unwanted edges may remain as presented in Figure 6.1 and we need to avoid introducing too many clusters, in particular falsely identifying clusters that have nothing to do with the true structure. In Section 7.2 we present a process to merge blocks in order to regulate their number. This process employs a greedy algorithm to maximise a certain modularity measure. This heuristic works best when provided with a starting configuration that conforms well with the block structure of the matrix. If there are too many false blocks, we have no guarantee that such a process will merge these blocks in a way that allows a correct detection in subsequent steps. However, if we do not keep an edge which corresponds to a separation between two blocks of the matrix, we could possibility re-detect it later, as shown in Section 4.

Because we expect the iterative projection process to increase the aspect ratio of the true edges in the vector, it is natural to use an SNR. In such an SNR, the signal is the staircase model $\mathbf{v}$, hence the amplitude of an edge is given by the height of the jump in $\mathbf{v}$. The amplitude of the background noise is measured by the difference in $\mathbf{u}$ and $\mathbf{v}$ on either side of an edge. Hence the SNR of an edge e_k is given by :

$$r(e_k) = \frac{\alpha_{k+1} - \alpha_k}{\varepsilon_{k+1} + \varepsilon_k}, \tag{6.1}$$

where α_k and α_{k+1} are the characteristic values of the blocks on either side of e_k and the noise ε_k in cluster C_k is given by

$$\varepsilon_k = \frac{1}{|C_k|} \sum_{i \in C_k} |\mathbf{u}(i) - \alpha_k|. \tag{6.2}$$

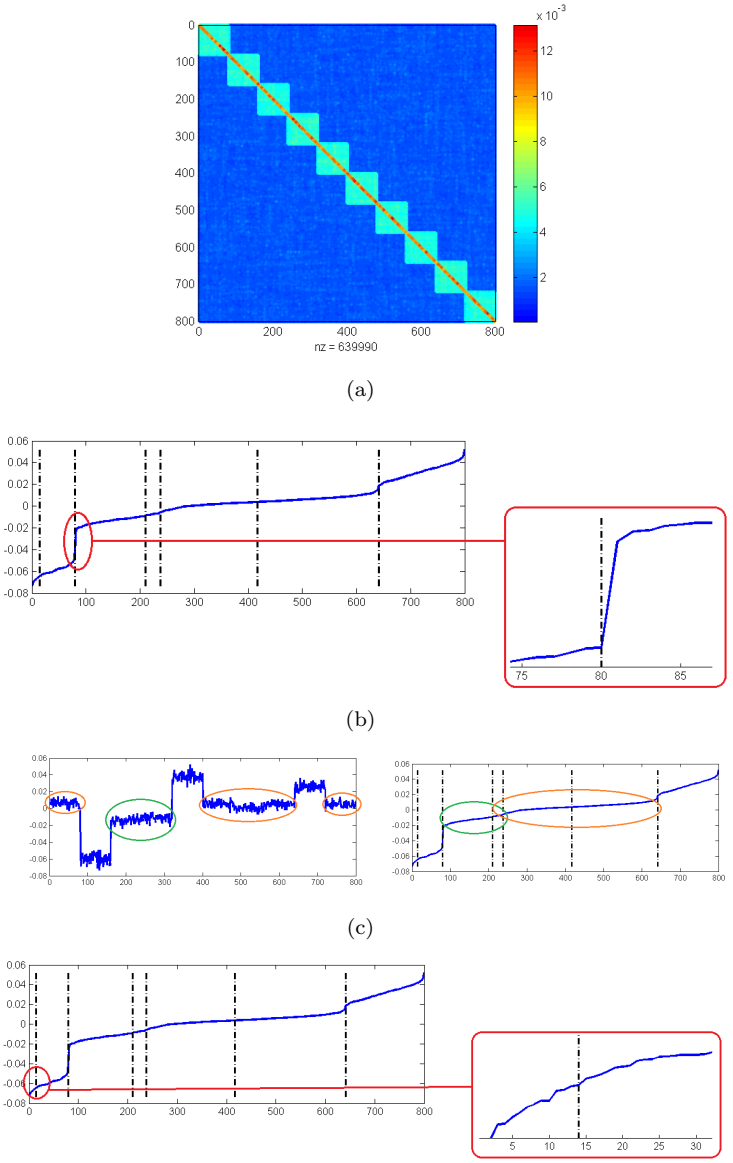
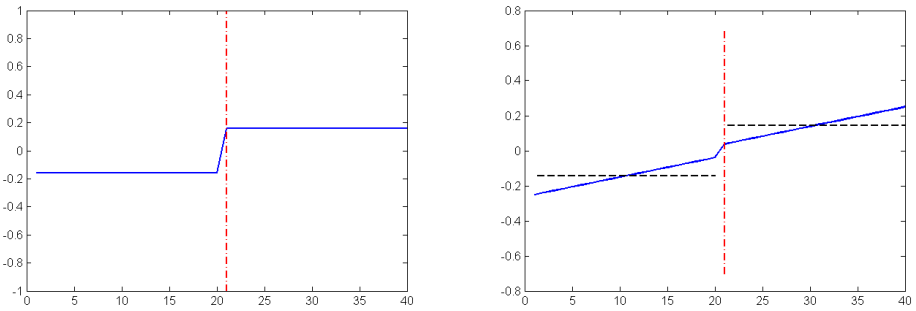
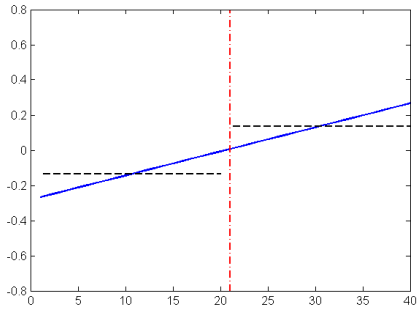


Figure 6.1: Different types of separation that the filtering process can cause. (a) Representation of the symmetric matrix from which the singular vector is extracted. This matrix has ten diagonal blocks along with some small perturbations. Its natural order fits with its block structure. (b) Left: in blue, a singular vector sorted in increasing order, in dashed black the separations found by the filter. Right: zoom in on one jump. The index of the separation should coincide with the first entry of the block to the right, whereas it is clear that it coincides here with the last entry of the block to the left. (c) Left: the singular vector in its natural order. Note the difference between the block circled in green and the blocks circled in orange. Right: the vector's entries in ascending order. The separation between the orange and the green blocks is blurred, and some entries from the two sets may be interlaced. The filter has divided this smooth slope in two places. (d) Zoom in on the first separation of the singular vector: a parasite introduced by the prolongation employed to avoid boundary effects in the convolution product.



(a) (b)



(c)

Figure 6.2: Three examples of vectors (solid blue line) with a detected edge (red dashed line). The black dotted lines represent the characteristic values of the clusters on each side of the edge (omitted in (a)). In (a) we want to keep the edge whatever the (positive) height of the jump. In (b) a priori we want to keep the edge, but it can depend on the effective height of the jump. In (c) we definitely want to discard the edge.

This SNR is infinite in the case of an edge perfectly fitting with the staircase model, as shown in Figure 6.2 (a), and so such an edge should not be removed. An example of a spurious edge is illustrated in Figure 6.2 (c). This has nothing to do with the staircase model but can occur when a constant section is subject to uniform noise and then sorted in increasing order to give an edge that divides a well-defined block in two. Basic arithmetic shows that the SNR of this edge, e, can be written as

$$r(e) = 2(1 + p(c, d)), \tag{6.3}$$

where c and d are the order of the blocks on each side of the edge e, and $p(c, d)$ is a simple function of c and d. For $c, d \geq 2$, we can show that $p(c, d) \in [0, 0.125]$.

Hence, we consider an edge as spurious and remove it if, after the process described in Section 6.1, its SNR is below the threshold $\eta = 2(1 + p(c, d))$. We can also choose $\eta \geq 2.5$ to be more selective. At any rate, this threshold value should be a trade-off between the confidence we want to have in an edge we keep, and the number of iterations we are prepared to carry out.

7 Cluster improvement

As mentioned before, cluster identification depends on how well the steps are represented by the entries of the singular vectors, and we may therefore apply the Canny filter to several singular vectors in turn, and merge those clusters identified from each vector independently in an embedded manner.

This is done by means of a lexicographical ordering of the labels assigned during independent cluster identification. This process is illustrated in Figure 7.1 (again on `rbsb480`), where the first detection step identifies only 4 row clusters and 5 column clusters, while the second left and right singular vectors suggest a different picture with 3 row clusters and 4 column clusters. These new clusters are complementary to the first set and the merged clustering gives 12 row clusters and 20 column clusters with fairly well separated subblocks in the resulting reordered matrix, as shown in the left-hand plot in Figure 7.2.

As we can see from our tests on `rbsb480`, there can be good reasons to analyse several vectors in turn and merge these results. However, this can be overdone and we may produce a very fine partitioning of the original matrix. This is illustrated in the right-hand plot of Figure 7.2, where 3 steps have been merged to give a fine grain clustering (with 48 by 100 clusters). Another issue is that because the row and column clusters are identified independently, the resulting clusters in the original matrix are potentially rectangular and may be scattered seemingly arbitrarily. Note that merging the steps of clustering (and labelling them in lexicographical order) favours neither bandwidth minimisation nor the movement of larger numerical entries towards the diagonal.

We therefore need to develop an effective method for amalgamating blocks—particularly tiny ones—and to perform a block-reordering, too. A further issue is to derive a stopping criterion with respect to this recursive clustering analysis.

7.1 Quality measure

We base our cluster analysis on modularity, first introduced in [34] for the case of undirected unweighted networks with binary adjacency matrices, and extended to weighted networks in [31]. Modularity is defined to be the quantity

$$Q = \frac{1}{2m} \sum_{i,j} \left(\mathbf{A}_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \tag{7.1}$$

where $\mathbf{A}$ is the $n \times n$ weighted adjacency matrix of an undirected weighted graph, $m = \frac{1}{2}\Sigma_{ij}\mathbf{A}_{ij}$ the sum of the edge weights, $k_i = \Sigma_j \mathbf{A}_{ij}$ the weighted degree of the node i, c_i the cluster to which node i belongs, and δ is the Kronecker function, so that $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise.

Modularity can be interpreted as the sum over all partitions of the difference between the fraction of links inside each partition and the expected fraction, by considering a random network with the same

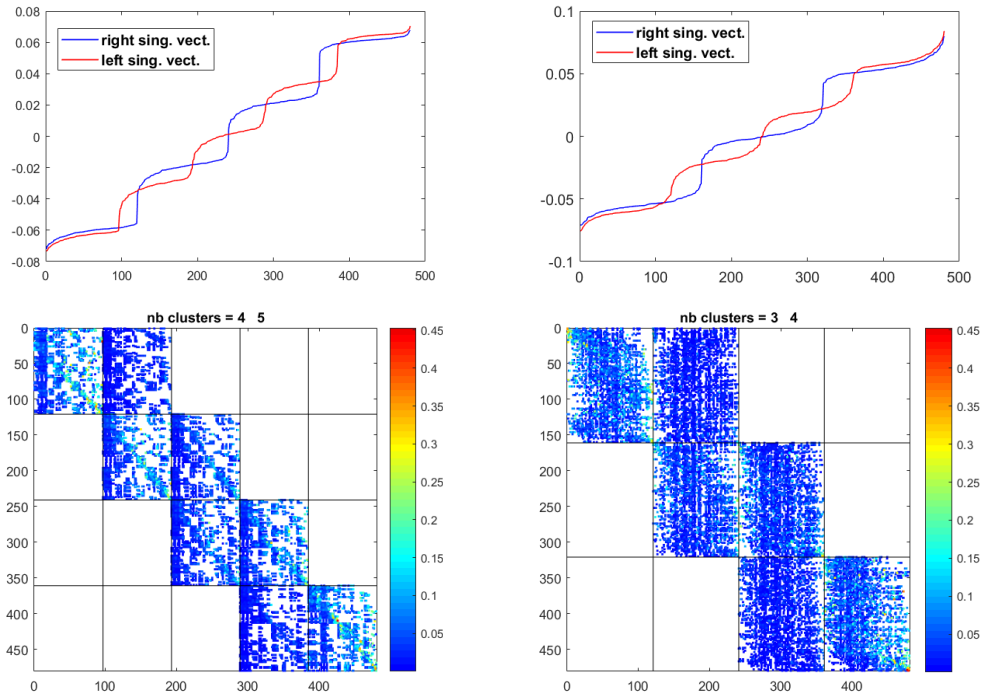


Figure 7.1: Example of cluster identification obtained by analysing different sets of singular vectors. Upper-left and upper-right plots show the left and right singular vectors for the first and second singular values respectively, with entries in ascending order. The lower-left plot shows the permuted matrix with its 4 by 5 clusters identified from the structure of the first vectors. The lower-right plot shows the permuted matrix and the 3 by 4 clusters extracted from the second vectors.

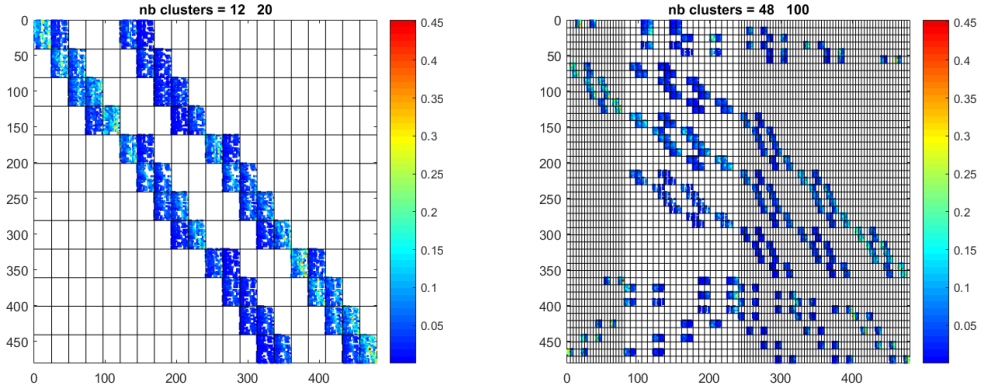


Figure 7.2: Example of recursive cluster identification obtained by analysing several singular vectors in turn. Left plot shows the permuted matrix and the 12 by 20 clusters obtained after merging the first clustering (4 by 5 clusters) together with that obtained from the second set of vectors (3 by 4 clusters). The row clustering has a modularity measure equal to 0.5034 (see subsection 7.1) and the column clustering has a modularity measure equal to 0.3085. The right plot illustrates the situation that is reached (48 by 100 blocks) after merging 3 clustering steps. On this configuration, the modularity of the row clustering is equal to 0.3625, whereas the column clustering modularity is equal to 0.1721.

degree for each node. This measure has proved popular with many authors. It was initially developed to choose the most coherent refinement level from the output of a hierarchical clustering algorithm and it has also been proposed as a quantity to maximise to find good quality partitions of matrices. This can be supported by local [4] or global [33] heuristics. Thus we feel it meets our needs for quality evaluation.

The modularity is rewritten in [3] as

$$Q = \frac{1}{2m} \sum_{k=1}^{r_C} \mathbf{v}_k^T \left(\mathbf{A} - \frac{\mathbf{r}\mathbf{r}^T}{2m} \right) \mathbf{v}_k, \tag{7.2}$$

where r_C is the number of clusters, $\mathbf{v}_k$ is a vector whose ith entry equals 1 if node i is in cluster k and 0 otherwise, and $\mathbf{r}$ is the vector of row sums of $\mathbf{A}$. Even if this formulation was originally designed for unweighted matrices, we note it stays consistent with the measure for weighted graphs.

In the case where the generic matrix $\mathbf{A}$ is the doubly stochastic matrix $\mathbf{P}\mathbf{P}^T$, we can evaluate our row clustering using the formula

$$\mathcal{Q}_r = \frac{1}{n} \sum_{k=1}^{r_C} \left(\mathbf{v}_k^T \mathbf{P}\mathbf{P}^T \mathbf{v}_k - \frac{1}{n} |\mathcal{J}_k|^2 \right). \tag{7.3}$$

easily derived from (7.2), noticing that $2m = n$ in the doubly stochastic case. Here, n is the number of rows in $\mathbf{P}$ and $|\mathcal{J}_k|$ is the cardinality of cluster k. In a similar way, we derive a quality measure $\mathcal{Q}_c$ for the column clustering by considering the matrix $\mathbf{P}^T\mathbf{P}$ and summing over the sets of column clusters.

If we only have a single cluster incorporating the entire matrix the quality measure is zero, simply because $\mathbf{P}\mathbf{P}^T$ is doubly stochastic and the sum of all its entries is equal to its size n, and the cardinality of the single cluster incorporating all the entries will also be equal to n. If we introduce $\mathbf{V}_{r_C}$ the matrix formed by gathering all the $\mathbf{v}_k$, $k = 1, \ldots, r_C$, we can easily see that

$$\mathcal{Q}_r = \frac{1}{n} \operatorname{trace} \left(\mathbf{V}_{r_C}^T \left(\mathbf{P}\mathbf{P}^T - \frac{\mathbf{e}\mathbf{e}^T}{n} \right) \mathbf{V}_{r_C} \right), \tag{7.4}$$

because $|\mathcal{J}_k|^2 = \mathbf{v}_k^T \left(\mathbf{e}\mathbf{e}^T \right) \mathbf{v}_k$. Since $\mathbf{P}\mathbf{P}^T$ is doubly stochastic, the vector $\mathbf{e}/\sqrt{n}$ is a normalized eigenvector corresponding to the dominant eigenvalue so $(\mathbf{P}\mathbf{P}^T - \mathbf{e}\mathbf{e}^T/n)$ is also positive semi-definite ensuring that the quality measure $\mathcal{Q}_r$ is nonnegative.

Additionally, since $\mathbf{v}_k/\sqrt{|\mathcal{J}_k|}$ is a vector with 2-norm equal to one, we observe that the value of $\mathbf{v}_k^T \mathbf{P}\mathbf{P}^T \mathbf{v}_k$ in (7.3) is bounded above by the corresponding cardinality value $|\mathcal{J}_k|$, and therefore

$$\mathcal{Q}_r \leq 1 - \sum_{k=1}^{r_C} \frac{|\mathcal{J}_k|^2}{n^2}.$$

Denoting the value of $|\mathcal{J}_k|/n$ by α_k, the Cauchy-Schwarz inequality tells us that

$$1 = \left(\sum_{k=1}^{r_C} \alpha_k \right)^2 \leq r_C \sum_{k=1}^{r_C} \alpha_k^2,$$

and consequently that the quality measure falls between the bounds

$$0 \leq \mathcal{Q}_r \leq 1 - \frac{1}{r_C},$$

with the upper bound being reached only when the r_C clusters have the same cardinality. Using similar arguments, this upper bound has previously been established for unweighted graphs [12].

Despite issues with the so-called resolution limit [16], modularity is still one of the most widely used measures in the field of community detection and we are comfortable in employing it here. Furthermore, in the specific case of k-regular graphs[4], many measures—the Newman and Girvan modularity, balanced modularity as well as deviation to indetermination and deviation to uniformity measures—are standardised and behave similarly [9]. Since doubly stochastic matrices can be considered as adjacency matrices of weighted 1-regular graphs most common quality measures can be expected to give similar results.

[4]Graphs in which all the nodes have the same degree k

7.2 Merging the clusters

If we introduce the condensed matrix $\mathbf{S}_{r_C} = \mathbf{V}_{r_C}^T \mathbf{P}\mathbf{P}^T \mathbf{V}_{r_C}$, the quality measure, $\overline{\mathcal{Q}_r}(k,\tilde{k})$, that we would get by merging together the kth and $\tilde{k}$th clusters is given by

$$\overline{\mathcal{Q}_r}(k,\tilde{k}) = \mathcal{Q}_r + \frac{2}{n}\mathbf{S}_{r_C}(k,\tilde{k}) - \frac{2}{n^2}|\mathcal{J}_k| \cdot |\mathcal{J}_{\tilde{k}}|\,. \tag{7.5}$$

From this we can see that if $\mathbf{S}_{r_C}(k,\tilde{k})$, the $(k,\tilde{k})$th entry of $\mathbf{S}_{r_C}$, is very small, indicating that the two row clusters with entries in $\mathcal{J}_k$ and $\mathcal{J}_{\tilde{k}}$ are almost numerically independent, merging the pair of clusters $(k,\tilde{k})$ will result in a decrease in this quality measure. We can look for a local maximum of this quality measure by merging together the pair of clusters $(k,\tilde{k})$ that maximise the value of $\mathcal{Q}_r$. To do so we evaluate the value of $\overline{\mathcal{Q}_r}(k,\tilde{k})$ over all the possible merged pairs of clusters (a total of $r_C(r_C-1)/2$ possibilities) and merge the pair that give the maximum increase. We can iterate this process. Since the condensed matrix $\mathbf{S}_{r_C}$ is available to us, testing the various pairwise combinations of clusters and evaluating the corresponding value of the quality measure is relatively cheap in computational terms. It is useful to further condense $\mathbf{S}_{r_C}$ by summing together its kth and $\tilde{k}$th rows and columns to form $\mathbf{S}_{r_C-1}^{k,\tilde{k}}$, before proceeding with the iteration. An analogous algorithm is applied independently to the c_C column clusters to test for pairwise column clusters amalgamation. The algorithm stops naturally when there is no further improvement to be made by merging pairs of clusters together.

To test for improvements in the quality measure it is sufficient to consider merging only pairs of clusters. This follows from the observation that if no pair improves the quality measure then from (7.5) we know that $n\mathbf{S}_{r_C}(k,\tilde{k}) \leq |\mathcal{J}_k| \cdot |\mathcal{J}_{\tilde{k}}|$ for all $k \neq \tilde{k}$, and we conclude that any amalgamated subset of clusters will have a local contribution to the quality measure that is less than the sum of contributions from each individual cluster within that subset. Since each pairwise amalgamation iteration changes the current clustering state, we cannot guarantee that we will stop at a global optimum for the value of the quality measure. However we can at least guarantee a local optimum in the final state.

The process of recursively amalgamating the pair of clusters which most improves the modularity is widely used in community detection where it is known as the greedy algorithm—see for instance [8] and [5]. But it is usually initiated with the trivial clustering where each node is a cluster. A proof that this method always provides a local maximum can be found in [8] but our demonstration is more general, because it also proves that once we reach a local maximum, no cluster amalgamation of any kind—e.g. of three or more clusters instead of two—will improve the quality measure.

The amalgamation process prevents an explosion in the number of clusters and gives some control over the amount of work for testing the $r_C(r_C-1)/2$ possibilities and is applied after merging the various clusters for each singular vector.

Table 7.1: Quality measure values, before and after pairwise cluster amalgamation, if we take only one vector at each iteration.

Clustering step	r_C	$\mathcal{Q}_r$	c_C	$\mathcal{Q}_c$
Step 1	4	0.5021	5	0.4026
Step 2 (before amalg.)	12	0.5033	20	0.3088
Step 2 (after amalg.)	7	0.5574	5	0.4967
Step 3 (before amalg.)	28	0.4239	10	0.4761
Step 3 (after amalg.)	7	0.5592	6	0.5004
Step 4 (before amalg.)	24	0.4477	10	0.4706
Step 4 (after amalg.)	7	0.5592	6	0.5004

The amalgamation process is illustrated in Figure 7.3. After three recursive clustering steps together with a final pairwise amalgamation, we reach a situation with only 7 row and 5 column clusters instead of

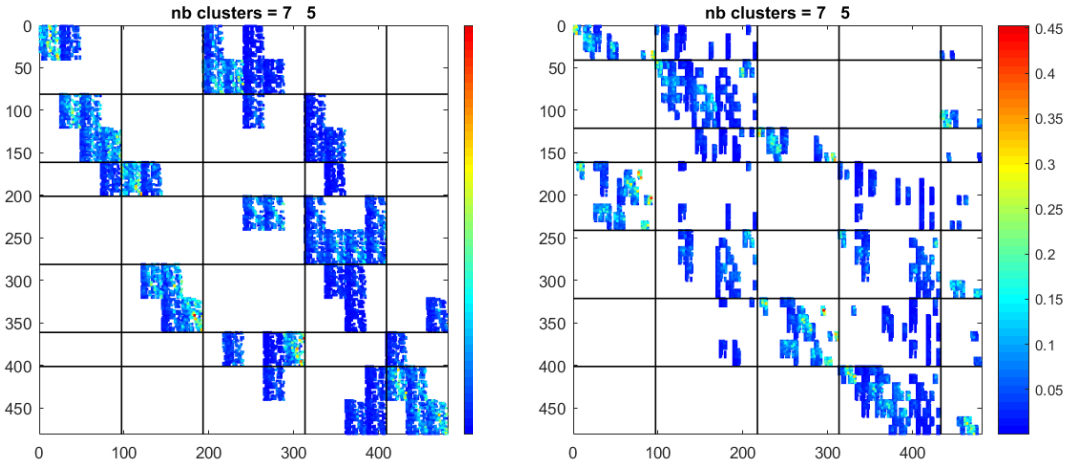


Figure 7.3: An example of pairwise cluster amalgamation. The left plot shows the result immediately after the second stage, in which 12 row and 20 column clusters are identified initially. This is reduced to a 7×5 block structure after amalgamation, with a quality measure of 0.5574 for the row clustering and 0.4932 for the columns. The right-hand plot shows the situation after 3 iterations of recursive clustering together with amalgamation. The final 7×5 block structure has a quality measure of 0.5574 on the rows and 0.502 on the columns.

the 48 by 100 blocks that we would obtain without amalgamation. Table 7.1 shows the evolution of the quality measure after each amalgamation step, which clearly limits the number of clusters.

We note that this amalgamation process is strongly linked to the numerical values within the blocks, as well as their cardinality. An example is provided in Figure 7.4, where the amalgamation process behaves differently with respect to varying numerical values within some small diagonal blocks which we believe is an example of resolution limit.

Because we can only reach a local optimum, we compare the quality of the clustering updated by the amalgamation process, $\mathcal{Q}_r^{upd}$ with that obtained immediately beforehand, $\mathcal{Q}_r^{ref}$. There are four possibilities to consider.

- If $\mathcal{Q}_r^{upd} < \mathcal{Q}_r^{ref}$ we reject the update.
- If $\mathcal{Q}_r^{upd} > \mathcal{Q}_r^{ref}$ but the number of clusters increases we accept the update only if there is a sizeable jump in modularity. Recall that we know the upper bound $\mathcal{Q}_r \leq 1 - \frac{1}{r_C}$. We compare the increase in the quality measure increase with its theoretical potential increase using the ratio

$$\rho = \frac{\mathcal{Q}_r^{upd} - \mathcal{Q}_r^{ref}}{\frac{1}{nbClust^{ref}} - \frac{1}{nbClust^{upd}}}.$$

  The threshold can be tuned to control the number of iterations we take and the granularity of the clustering. In tests, a threshold value between 0.01 and 0.1 seems to be effective.
- If $\mathcal{Q}_r^{upd} > \mathcal{Q}_r^{ref}$ and $nbClust^{upd} \leq nbClust^{ref}$ we accept the update.
- If the clusterings are identical up to a permutation then we can terminate the algorithm. For simplicity we reject the update.

The reasoning behind the second case is inspired by numerical optimisation. For instance, in the trust-region method [10], when looking for a minimum (or maximum) of a complicated function f it is preferable to determine the behaviour of an approximation for f (for instance its second order Taylor

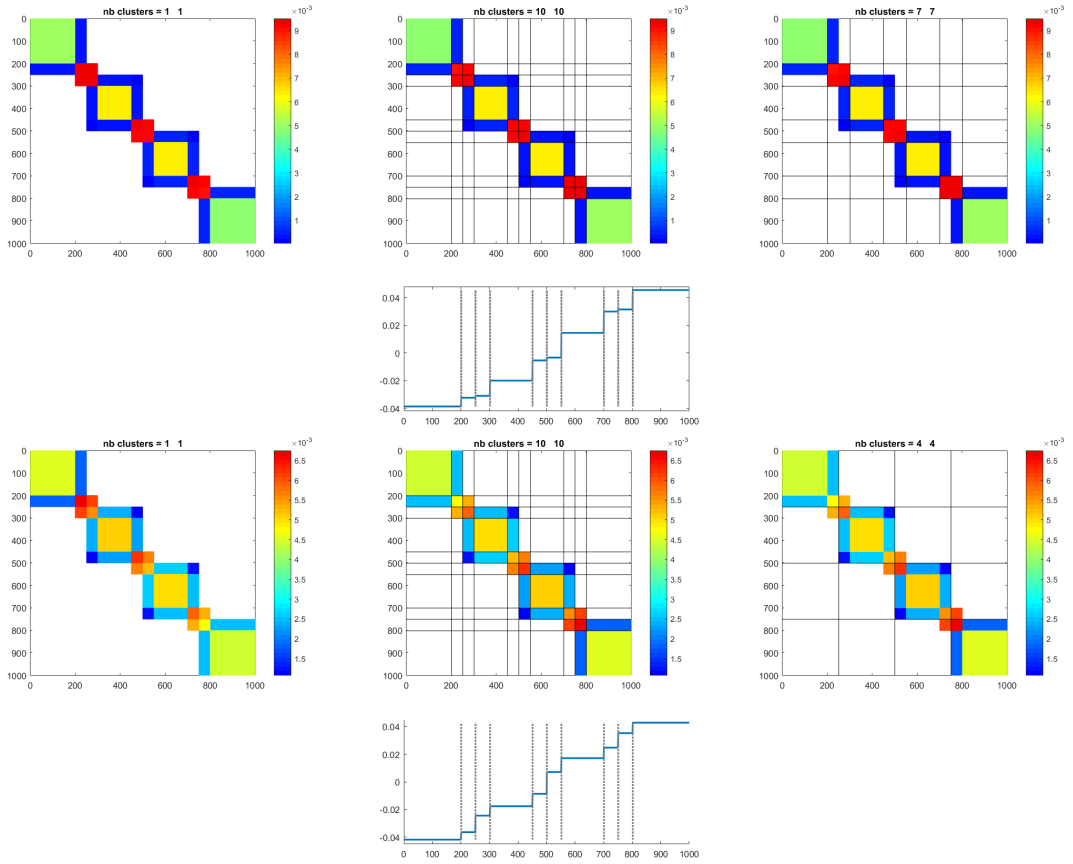


Figure 7.4: Two matrices (top and bottom) with perfect block structure. Before scaling the size of the entries in the large blocks are $O(1)$ in both matrices. In the top matrix the sizes of the entries in the small blocks are $O(10^3)$ but $O(10)$ in the bottom. Left: doubly-stochastic scalings. Middle: filtering and edge refinement detect all ten blocks perfectly. Right: after amalgamation. In the bottom matrix the small blocks are incorporated into bigger blocks. In the top matrix small blocks are grouped together. For both matrices we show below the middle plot the eigenvector from which the algorithm has derived the partitioning. The dashed lines correspond to the edges returned by the algorithm.

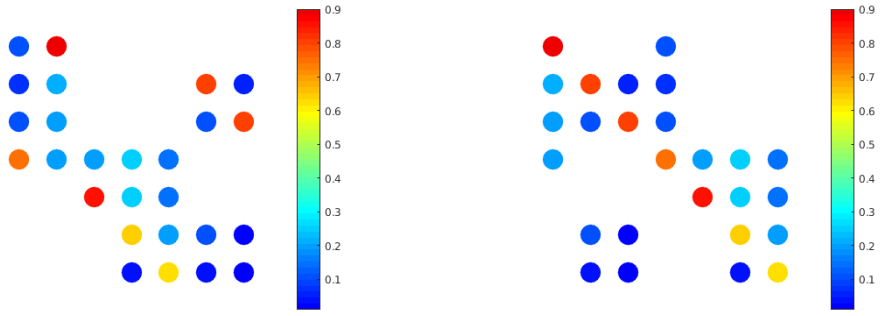


Figure 8.1: Moving weight to the diagonal.

expansion). We then compare the actual change in the function at a proposed new point with the change in the approximation. The new iterate is accepted if these changes are within a predetermined ratio.

Note that the algorithm only stops if the new clusterings on the rows and the columns are both rejected on the same iteration. So long as we find the singular vectors deterministically, if row and column clusterings are the same in successive iterations then this will be the case for all subsequent iterations and so we can safely terminate the algorithm. Notice that in the example given in Table 7.1 the amalgamation process stops after the third step.

8 Applications

In this section we present the behaviour of our algorithm on two applications: as a preconditioner for linear systems and to detect communities in networks.

8.1 Preconditioning of linear systems

Our algorithm identifies blocks in the matrix which can be used to construct a preconditioner for iterative solvers. Indeed, by applying a post-processing, we should be able to obtain a matrix which can be used as an "Enhanced block Jacobi preconditioner", in the sense introduced by Zhu and Sameh in [45], where the authors obtain efficient block Jacobi preconditioners that have dominant blocks along the diagonal and only a few entries elsewhere. They create their own block Jacobi preconditioner for a matrix $\mathbf{A}$ by finding a permutation $\tilde{\mathbf{A}}$ that maximises the product of diagonal entries. Then they partition $\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^T$ into 8 parts in which they both balance the size of the blocks and minimise the weighted cut between the blocks.

In the partitioning from our algorithm, we get a number of rectangular blocks of varying size which can be permuted into a checkerboard pattern. We can transform it into a configuration with square diagonal blocks of maximum weight along the diagonal. The problem of permuting rectangular tiles with varying row and column sizes is not trivial, but we can perform a bipartite matching to make sure that the diagonal has as much weight as possible. An illustrative example is shown in Figure 8.1. After identifying the entries that give a diagonal with the maximum product, we can then further divide rectangular tiles which contain a set of these entries, and obtain square blocks incorporating these specific entries that can be permuted to the diagonal. As we see in the bottom left plot of Figure 8.2, this process also creates tiny blocks. To mitigate against this, we first perform a symmetric version of the pairwise amalgamation process of Section 7.2. This symmetric amalgamation uses the sum of the row and the column quality measures $(\mathcal{Q}_r + \mathcal{Q}_c)$. Each time a maximising pair $(k, \tilde{k})$ is found, we merge both the corresponding row and column clusters. The bottom right plot in Figure 8.2 and the bottom left plot in Figure 8.3 illustrate the benefit of this final amalgamation stage. We have generated a symmetric permutation from the different row and column permutations so that the blocks on the diagonal are those with larger numerical weights.

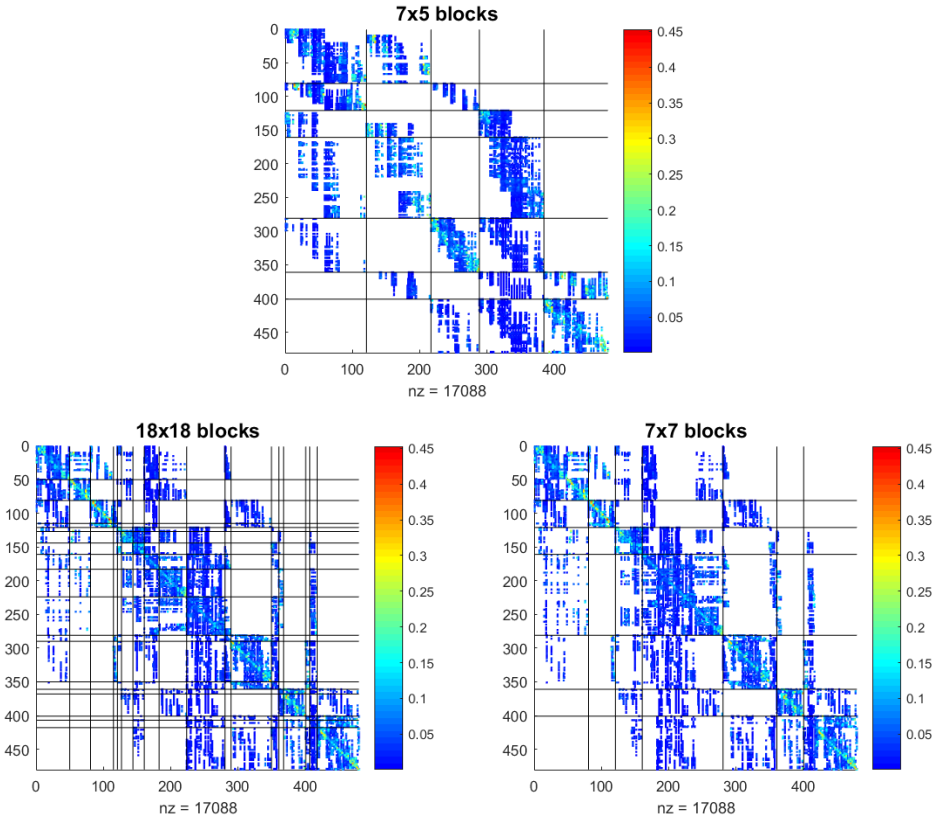


Figure 8.2: Chasing the maximum diagonal: the upper plot is the output of our algorithm, before postprocessing; the bottom left plot shows the partitioning after chasing the maximum diagonal and the bottom right plot is the result after the symmetric amalgamation.

Table 8.1: Matrix statistics and the number of blocks in the preconditioners V0 and V1. We remark that in V1, the aim of 8 blocks is not always feasible because condensed matrices present have many fewer nodes than the initial matrix.

Matrices	Size	Number of nonzeros	Condition number	V0: nb of blocks	V1: nb of blocks
`685_bus`	685	3249	4.23×10^5	25	8
`af23560`	23560	460598	1.99×10^4	27	8
`cage8`	1015	1103	1.14×10^1	19	8
`cage9`	3534	41594	1.26×10^1	25	8
`e40r0100`	17281	553562	1.51×10^8	27	8
`epb1`	14734	95053	5.94×10^3	35	8
`gre_1107`	1107	5664	3.19×10^7	10	6
`Hamrle2`	5952	22162	6.69×10^4	617	8
`Ill_Stokes`	20896	191368	2.25×10^9	18	8
`ns3Da`	20414	1679599	7.07×10^2	46	8
`nemeth26`	9506	1511760	1.00×10^0	13	7
`rbsb480`	480	17088	1.20×10^4	7	3
`sme3Da`	12504	874887	5.12×10^7	15	5

We have tested our algorithm with this postprocessing—denoted in the following by V0—as a preconditioner for `gmres` on some matrices from the SuiteSparse Matrix Collection [13] whose characteristics are given in Table 8.1. As we see when looking at the blue and green bars in Figure 8.4, the process does not succeed in creating an efficient block Jacobi preconditioner in comparison with our coding of the algorithm of Zhu and Sameh in [45] (denoted by ZS). In our version of ZS, we use the scaling generated when the diagonal is maximised.

There are two main drawbacks to using the symmetric amalgamation for this application. First, we are not able to control the number of blocks in the partitioning although it is clear, by looking at the number of blocks for V0 in Table 8.1, that this has a significant impact on the performance of the preconditioner. Second, the amalgamation is based on a modularity measure that merges the blocks according to a random graph model which is not relevant to our application.

With these observations, we provide another preconditioner—denoted by V1—based on a postprocessing using the same tools as ZS. That is, after having chased the maximum diagonal and getting many more partitions in order to obtain square diagonal blocks (as the first steps of the V0 postprocessing), we use this partitioning to create a condensed matrix $\mathbf{A}_c$ from the initial matrix by taking the weight of a diagonal entry equal to 1, and the weight of an off-diagonal entry (i, j) as the sum of the entries in the corresponding block (i, j) of the initial doubly-stochastic matrix. An example of this condensed matrix is shown in the bottom right plot of Figure 8.3. We then apply the `kway` graph partitioning process from `Metis` [23] to partition $\mathbf{A}_c + \mathbf{A}_c^T$, into 8 blocks for compatibility with ZS. We note that it is not always possible to get 8 blocks, and we show the actual number of blocks in V1 in the last column of Table 8.1.

As we can see in Figure 8.4, this V1 preconditioner has comparable performance to that of ZS. But we would like to emphasise that this preconditioner is only a first attempt at evaluating the potential of our algorithm for this application. In future work specifically focused on preconditioning, it would be worth using a cut function as in `Metis` instead of the modularity for the amalgamation performed after each iteration in the algorithm, as it seems that cut functions result in objective functions better adapted for this application. Finally we remark that the condensed matrix can be used in a block iterative solver such as block Cimmino [1, 41], and so offers many exciting possibilities.

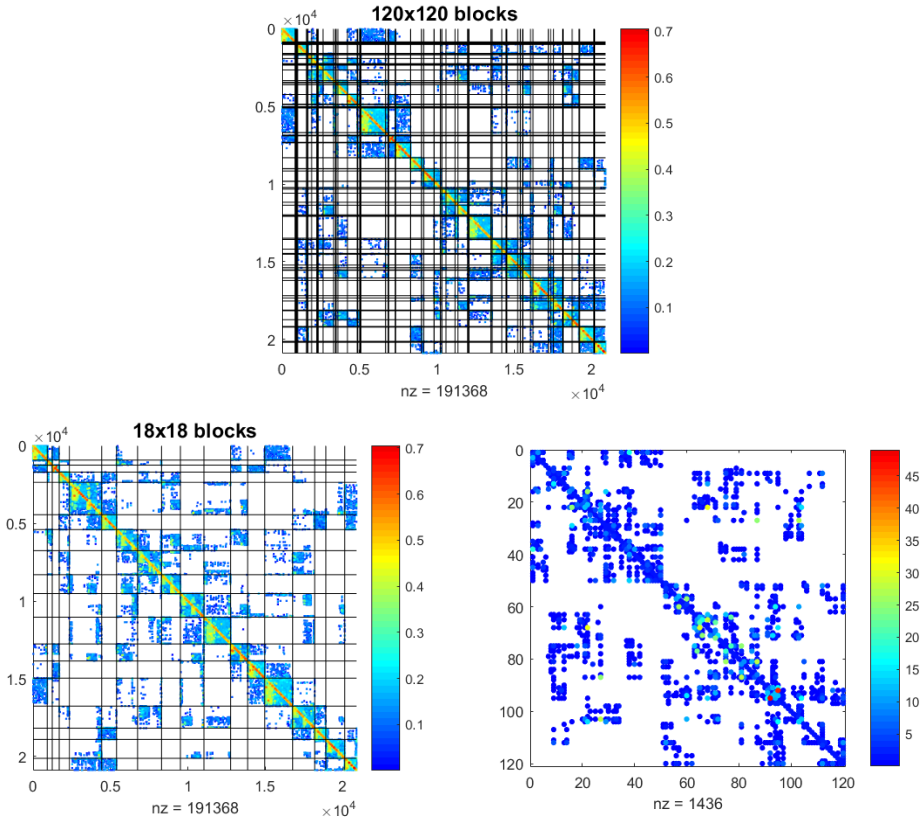


Figure 8.3: The upper plot is the partitioning of `Ill_Stokes` matrix after chasing the maximum diagonal; the bottom left plot coincides with the result after the symmetric amalgamation and the bottom right plot is the condensed matrix from the upper partitioning.

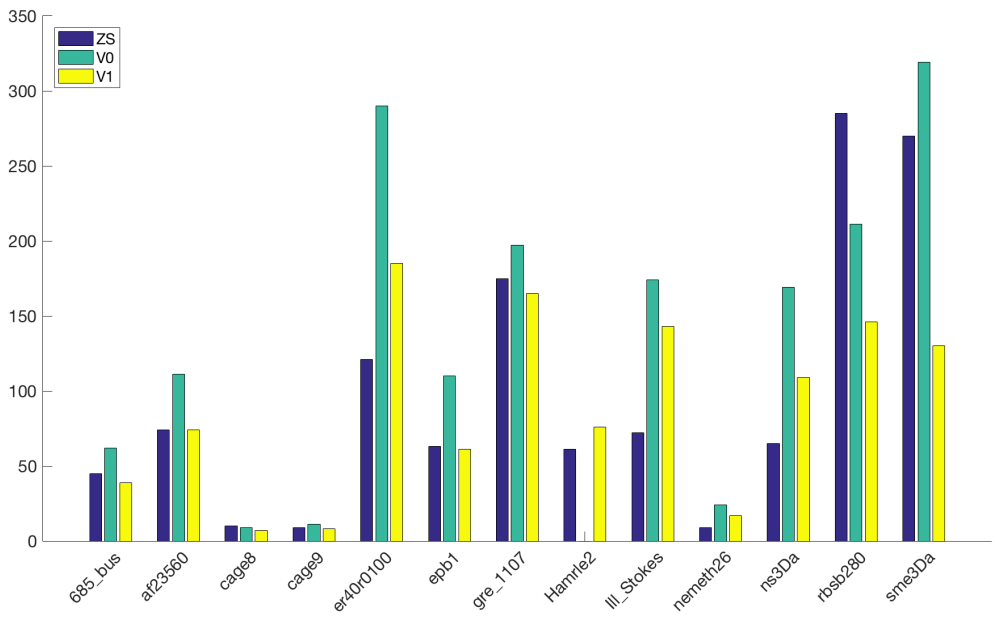


Figure 8.4: Comparison between the number of iterations to solve different linear systems with a `gmres` solver, with a tolerance 10^{-6} for the different preconditioning methods ZS, V0 and V1. For `Hamrle2`, V0 fails to converge.

8.2 Community detection

Because our algorithm is designed to detect dense diagonal blocks in sparse matrices it is well-suited to detect communities in networks by manipulating their adjacency matrices.

As for preconditioning, we need to adapt our algorithm for this specific application. Here we need to restrict ourselves to use a symmetric permutation for identifying communities. We add a small multiple of the identity matrix that has the double benefit of ensuring that we to use only symmetric permutations to identify irreducible blocks while also guaranteeing the convergence to a doubly stochastic matrix. Normally we set the diagonal to values around 10^{-8}. Although this does not avoid large diagonal entries appearing in the scaled matrix it reduces the risk considerably. We can use the quality measure described in Section 7.1 without the need to work on the normal equations.

During the preprocessing stage described in Section 3, we remove dominant entries in the scaled matrix. In networks, these entries are associated with structures not strongly associated with communities, such as pendant nodes. Once our algorithm has converged, we incorporate these entries using our amalgamation process to optimise modularity.

We have used our algorithm on some classic examples of clusters in undirected graphs—such as the Zachary karate club [44, 20] and the dolphins of Doubtful Sound [29]—and have recovered the ground truth communities. For a more comprehensive evaluation of our algorithm we have tested it on artificial benchmarks as described in [43]. We compare our algorithm against representative community detection algorithms from the `igraph` package [11] using two metrics. First, Normalised Mutual Information (NMI) from information theory (see for instance [17]). This measures the deviation between two candidate partitions with no requirement that they have the same number of blocks. For two partitions of a set of m elements, $\mathcal{C} = \{C_1, ...C_p\}$ and $\mathcal{K} = \{K_1, ..., K_q\}$, we can define the confusion matrix, $N \in \mathbb{N}^{p \times q}$, by

$$\forall i \in \{1...p\}, \forall j \in \{1...q\}, N(i,j) = |\{C_i \cap K_j\}|C. \tag{8.1}$$

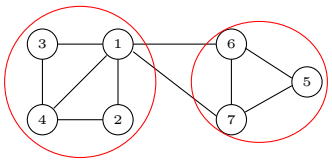


Figure 8.5: Graph to illustrate mixing parameter. Here, the node 1 has a mixing parameter equal to $\mu(1) = \frac{k_1^{out}}{k_1} = \frac{2}{5}$.

We can then define the NMI as

$$I(\mathcal{C},\mathcal{K}) = \frac{-2\sum_{k=1}^{p}\sum_{l=1}^{q} N(k,l)\log_2\left(mN(k,l)/\left(|C_k||K_l|\right)\right)}{\sum_{k=1}^{p}|C_k|\log_2(|C_k|/m) + \sum_{l=1}^{q}|K_l|\log_2(|K_l|/m)}, \tag{8.2}$$

where $|\cdot|$ is the cardinality. A complementary metric is the ratio of the number of blocks in our computed partitioning to the true number of clusters. We call this the cluster number ratio, $\overline{C}/C$.

The strength of a community structure can be quantified by the so-called mixing parameter. The mixing parameter of a node i, $\mu(i)$, measures the strength of a node's community membership by computing the ratio between its links outside the community and its degree. An example is shown in Figure 8.5. The greater the mixing parameter for each node, the weaker the community structure. The network mixing parameter, μ, is the mean of all the $\mu(i)$.

We have tested our algorithm on two widely-used benchmarks. The first is the Newman and Girvan (NG) benchmark, which provides networks of 128 nodes with 4 communities of 32 nodes each [21]. This is a historical benchmark for community detection algorithms, but the networks lack some of the features typical of real-world networks. Nevertheless it is still used to ensure that a community detection algorithm behaves as expected on a simple model. The second is the Lancichinetti–Fortunato–Radicchi benchmark (LFR) [27], widely used because its characteristics are close to those of real-world networks. For these two benchmarks, we have generated 100 networks for each μ value, and displayed the mean value of both NMI and number cluster ratio in each case.

We have compared our algorithm against four established community detection algorithms from the `igraph` package. Specifically the algorithms Walktrap (WT) [36], Louvain (LV) [4], Fast and Greedy (FG) [8], and Leading Eigenvector (LE) [32]. WT and LV are specifically designed to work on graph structures directly whereas LE focus (like ours) on exploiting spectral information. FG is the greedy algorithm described in the section 7.2.

The results for the NG benchmark are presented in Figure 8.6. In the top picture we show the results for NMI, and in the bottom picture the results for the cluster number ratio. We see that all the algorithms generally succeed in finding correctly the communities for μ less than or equal to 0.4.

The results for the LFR benchmark are presented in Figure 8.7. Here we can pair the existing algorithms in terms of performance: LV and WT, where NMI is close to 1 for $\mu < 0.65$, and FG and LE where NMI decreases rapidly and the cluster number ratio quickly moves away from 1. Our algorithm falls between the two groups.

These results are very encouraging. Even though our algorithm is not designed specifically for community detection, it is able to correctly detect communities better than some widely-used purpose-built algorithms. Moreover, our algorithm is not constrained to work with symmetric matrices and so provides a versatile tool for community detection in directed graphs whereas WT and LV—which both symmetrize the matrix by working on $A + A^T$—can provide spurious results. An example is shown in Figure 8.8, where two communities are highly connected in an asymmetric fashion. By working on the normal equations of the scaled matrix we are able to perfectly detect the community structure of the graph, while both WT and LV fail.

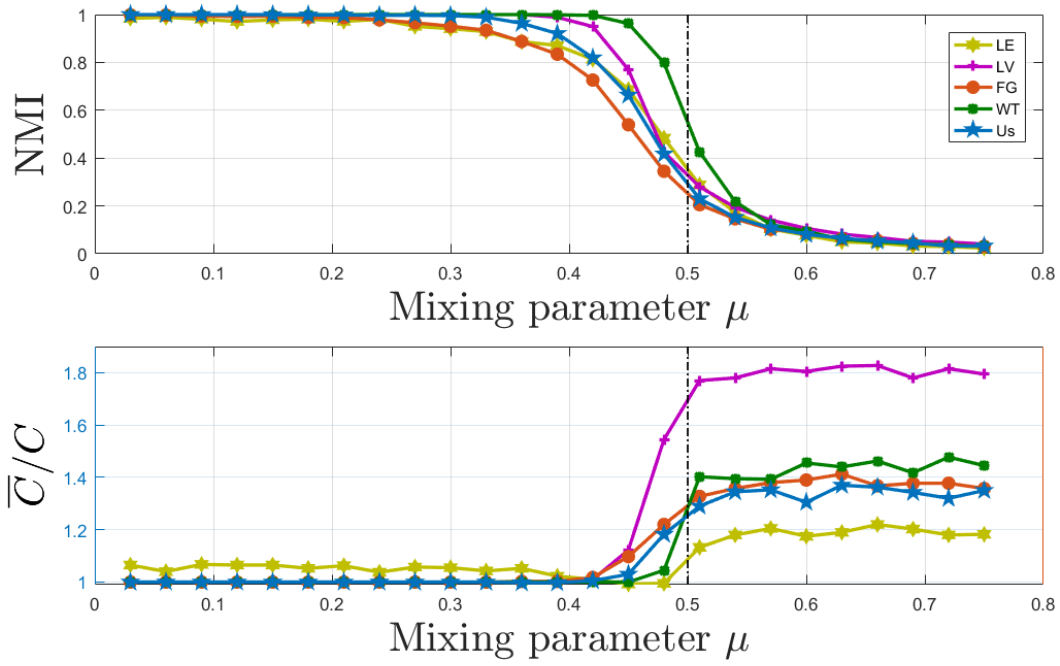


Figure 8.6: Community detection using the NG Benchmark: 128 nodes and 4 communities of 32 nodes (100 instances for a given μ value). Each curve colour corresponds to a specific algorithm given by the legend. The mean NMI values are plotted at the top, mean cluster number ratios at the bottom.

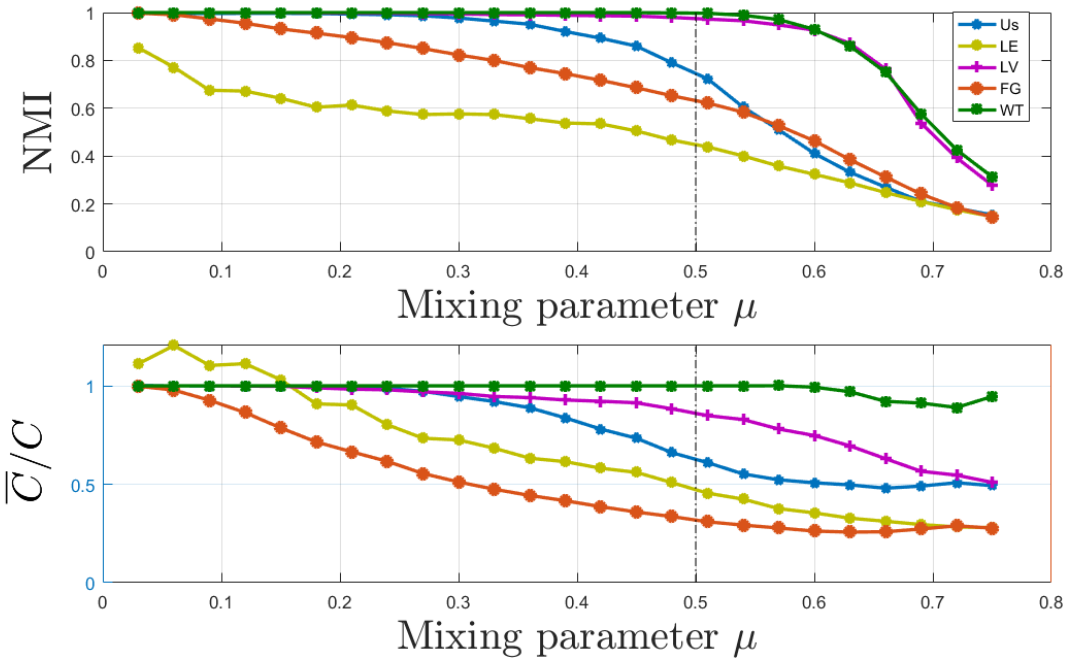


Figure 8.7: Community detection using the LFR Benchmark: 482 nodes (100 instances for a given μ value). The colour of each curve corresponds to a specific algorithm given by the legend. The mean NMI values are plotted at the top, the mean cluster number ratios are at the bottom.

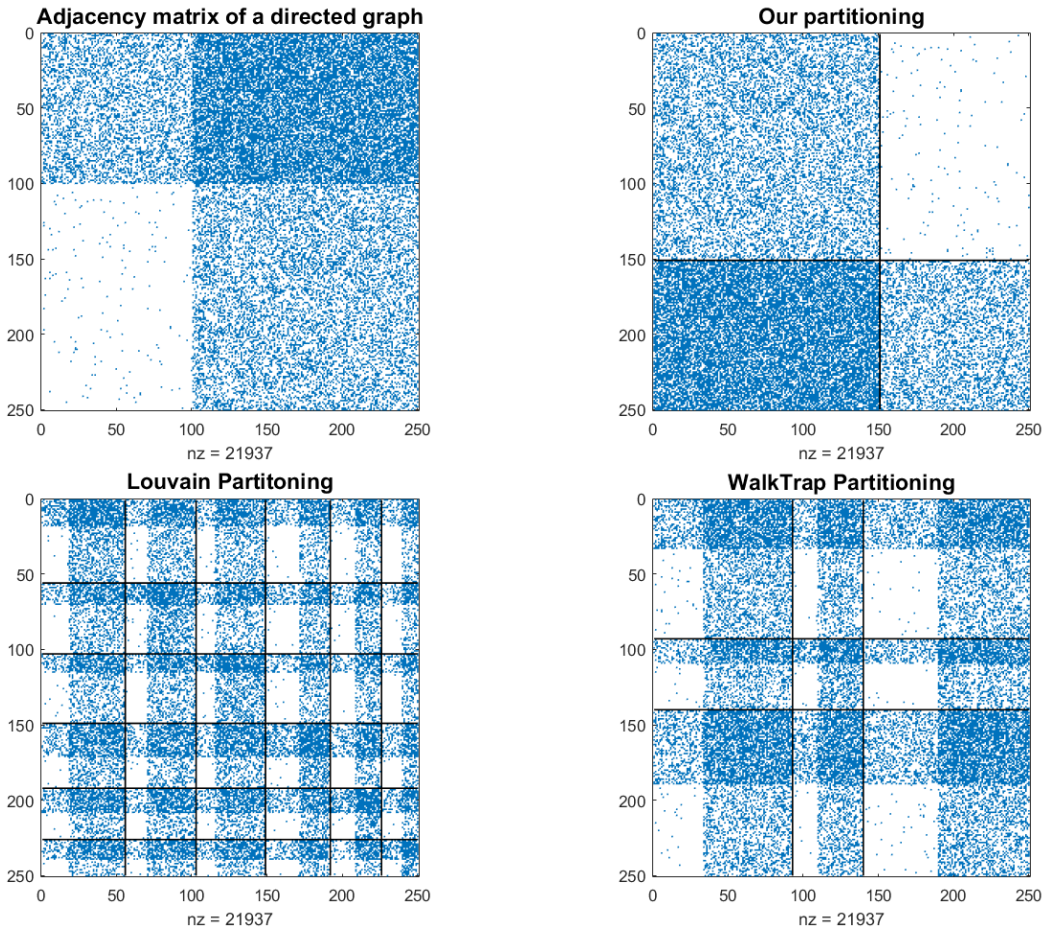


Figure 8.8: Community detection when communities are connected asymmetrically.

9 Conclusions

We have developed an algorithm that permutes and scales a matrix $\mathbf{A}$ so that it can be partitioned into the following block triangular form:

$$\begin{bmatrix} \mathbf{B}_1 & & * \\ & \ddots & \\ 0 & & \mathbf{B}_k \end{bmatrix}, \text{ with } \mathbf{B}_i = \begin{bmatrix} \mathbf{D}_i & \mathbf{G}_i \\ \mathbf{H}_i & \mathbf{C}_i \end{bmatrix}$$

where

- $\mathbf{D}_i$ is (strongly) diagonally dominant.
- $\mathbf{G}_i$ and $\mathbf{H}_i$ have very small norm.
- $\mathbf{C}_i$ is such that both $\mathbf{C}_i\mathbf{C}_i^T$ and $\mathbf{C}_i^T\mathbf{C}_i$ have a near block diagonal structure although the blocking will be different in each case.

We use this partitioning to construct a preconditioner for solving sparse linear systems using the GMRES iterative solver. We also illustrate the use of this partitioning in community detection. In both cases, we are competitive with methods specifically designed for these applications. Some steps of our algorithm can be adapted depending on the application, for example, the amalgamation step, modularity measure, preprocessing.

However, our approach is far more versatile since it can also be extended to rectangular matrices, and we plan to look at future applications including developing novel partitioning strategies for block iterative methods like block Cimmino and bi-clustering.

Acknowledgements

We would like to thank F. Sukru Torun for his help with software and general assistance, Yao Zhu for his enlightening explanations about the methodology they used to create their block Jacobi preconditioner, and Bora Uçar for many fruitful discussions, particularly on matching.

References

[1] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for sparse matrices*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 47–70.

[2] A. Azad, A. Buluç, and A. Pothen, *A parallel tree grafting algorithm for maximum cardinality matching in bipartite graphs*, in Proceedings of the IPDPS, 2015.

[3] J. P. Bagrow, *Communities and bottlenecks: Trees and treelike networks have high modularity*, Physical Review E, 85 (2012), p. 066118.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, *Fast unfolding of communities in large networks*, Journal of Statistical Mechanics: Theory and Experiment, 2008 (2008), p. P10008.

[5] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner, *On modularity clustering*, IEEE Transactions on Knowledge and Data Engineering, 20 (2008), pp. 172–188.

[6] R. A. Brualdi and H. J. Ryser, *Combinatorial Matrix Theory*, vol. 39 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, Cambridge, UK; New York, USA; Melbourne, Australia, 1991.

[7] J. Canny, *A computational approach to edge detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8 (1986), pp. 679–698.

[8] A. Clauset, M. E. J. Newman, and C. Moore, *Finding community structure in very large networks*, Phys. Rev. E, 70 (2004), p. 066111.

[9] P. Conde-Cespedes, *Modelisation et extension du formalisme de l'analyse relationnelle mathematique a la modularisation des grands graphes.*, PhD Thesis, Universite Pierre et Marie Curie, (2013).

[10] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, MPS/SIAM Series on Optimization, SIAM, Philadelphia, 2000.

[11] G. Csardi and T. Nepusz, *The igraph software package for complex network research*, InterJournal, Complex Systems (2006), p. 1695.

[12] B. DasGupta and D. Desai, *On the complexity of newmans community finding approach for biological and social networks*, Journal of Computer and System Sciences, 79 (2013), pp. 50 – 67.

[13] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–14.

[14] A. L. Dulmage and N. S. Mendelsohn, *Coverings of bipartite graphs*, Canadian Journal of Mathematics, 10 (1958), pp. 517–534.

[15] S. Fortunato, *Community detection in graphs*, Physics reports, 486 (2010), pp. 75–174.

[16] S. Fortunato and M. Barthlemy, *Resolution limit in community detection*, Proceedings of the National Academy of Sciences, 104 (2007), pp. 36–41.

[17] A. L. N. Fred and A. K. Jain, *Robust data clustering.*, in CVPR (2), IEEE Computer Society, 2003, pp. 128–136.

[18] D. Fritzsche, V. Mehrmann, D. B. Szyld, and E. Virnik, *An SVD approach to identifying metastable states of Markov chains*, Electronic Transactions on Numerical Analysis, 29 (2008), pp. 46–69.

[19] G. Frobenius, *Ueber matrizen aus nicht negativen elementen*, Sitzungsber. Königl. Preuss. Akad. Wiss, (1912), p. 456477.

[20] M. Girvan and M. E. J. Newman, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 7821–7826.

[21] ——, *Community structure in social and biological networks*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 7821–7826.

[22] J. A. Hartigan and M. A. Wong, *Algorithm as 136: A k-means clustering algorithm*, Journal of the Royal Statistical Society. Series C (Applied Statistics), 28 (1979), pp. 100–108.

[23] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[24] P. A. Knight, *The Sinkhorn–Knopp algorithm: Convergence and applications*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 261–275.

[25] P. A. Knight and D. Ruiz, *A fast algorithm for matrix balancing*, IMA Journal of Numerical Analysis, 33 (2013), pp. 1029–1047.

[26] P. A. Knight, D. Ruiz, and B. Uçar, *A symmetry preserving algorithm for matrix scaling*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 931–955.

[27] A. Lancichinetti, S. Fortunato, and F. Radicchi, *Benchmark graphs for testing community detection algorithms*, Physical Review E, 78 (2008), p. 046110.

[28] M. Lee, H. Shen, J. Z. Huang, and J. S. Marron, *Biclustering via sparse singular value decomposition*, Biometrics, 66 (2010), pp. 1087–1095.

[29] D. Lusseau, K. Schneider, O. Boisseau, P. Haase, E. Slooten, and S. Dawson, *The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations*, Behavioral Ecology and Sociobiology, 54 (2003), pp. 396–405.

[30] M. Manguoglu, M. Koyutürk, A. H. Sameh, and A. Grama, *Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers*, SIAM J. Scientific Computing, 32 (2010), pp. 1201–1216.

[31] M. Newman, *Analysis of weighted networks*, Physical Review E, 70 (2004), p. 056131.

[32] M. E. J. Newman, *Finding community structure in networks using the eigenvectors of matrices*, Physical review E, 74 (2006), p. 036104.

[33] M. E. J. Newman, *Modularity and community structure in networks*, Proceedings of the National Academy of Sciences of the United States of America, 103 (2006), pp. 8577–8582.

[34] M. E. J. Newman and M. Girvan, *Finding and evaluating community structure in networks*, Phys. Rev. E, 69 (2004), p. 026113.

[35] O. Perron, *Zur theorie der matrices*, Mathematische Annalen, 64 (1907), pp. 248–263.

[36] P. Pons and M. Latapy, *Computing communities in large networks using random walks (long version)*, ArXiv Physics e-prints, (2005).

[37] A. Pothen and C.-J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Transactions on Mathematical Software, 16 (1990), pp. 303–324.

[38] D. Ruiz, *A scaling algorithm to equilibrate both rows and columns norms in matrices*, Tech. Rep. RAL-TR-2001-034 and RT/APO/01/4, Rutherford Appleton Laboratory, Oxon, UK and ENSEEIHT-IRIT, Toulouse, France, 2001.

[39] S. E. Schaeffer, *Graph clustering*, Computer Science Review, 1 (2007), pp. 27–64.

[40] R. Sinkhorn and P. Knopp, *Concerning nonnegative matrices and doubly stochastic matrices*, Pacific J. Math., 21 (1967), pp. 343–348.

[41] F. Sukru Torun, M. Manguoglu, and C. Aykanat, *A Novel Partitioning Method for Accelerating the Block Cimmino Algorithm*, ArXiv e-prints, (2017).

[42] E. Vecharynski, Y. Saad, and M. Sosonkina, *Graph partitioning using matrix values for preconditioning symmetric positive definite systems*, SIAM J. Scientific Computing, 36 (2014), pp. A63–A87.

[43] Z. Yang, R. Algesheimer, and C. J. Tessone, *A comparative analysis of community detection algorithms on artificial networks*, Scientific Reports, 6 (2016), pp. 30750 EP –. Article.

[44] W. W. Zachary, *An information flow model for conflict and fission in small groups*, Journal of Anthropological Research, 33 (1977), pp. 452–473.

[45] Y. Zhu and A. H. Sameh, *How to generate effective block jacobi preconditioners for solving large sparse linear systems*, in Advances in Computational Fluid-Structure Interaction and Flow Simulation: New Methods and Challenging Computations, Y. Bazilevs and K. Takizawa, eds., Cham, 2016, Springer International Publishing, pp. 231–244.