Department of Business and Management

Teaching: Social Network Analysis [MAT/09]

Spectral Methods for Network Generation

SUPERVISOR

Prof. Xavier Mathieu Raymond Venel

CANDIDATE

Maria Chiara Lischi

ID: 271281

Academic Year 2023/2024

Abstract

This work delves into the mathematical foundations and elaborates on a methodology for network generation based on spectral properties of graph-associated matrices. First, a comprehensive mathematical framework focusing on eigenvalues, eigenvectors, and matrix diagonalization was established. Next, network theory was explored, detailing basic concepts, standard models for network generation, and the spectral properties of matrices related to graphs.

Then, the Spectral Graph Forge (SGF) method was introduced, leveraging these spectral properties to generate synthetic networks with the same community structure of a target one. Detailed mathematical formulations and coding implementations are provided, demonstrating the application of the method to both small and large networks. Finally, the algorithm was extended by introducing new transformations, with the goal to broaden the range of global properties that can be preserved in the synthetic network.

Acknowledgements

This work would not have been possible without the guidance, dedication, and patience of my supervisor, Professor Venel. I am deeply grateful for his support in developing my research in the area that sparked my curiosity the most. Thank you for the countless hours of discussion, the ones spent reading my drafts, and for making sure that I say what I mean and mean what I say.

I extend my gratitude to the administration, faculty, and staff at LUISS Guido Carli for providing a stimulating academic environment and encouraging my intellectual growth during my years here. In particular, I wish to thank President Gubitosi, President Lucchini, Rector Professor Prencipe, General Director. Lo Storto, Director Lubicz, Director Capone, and Director Professor Finocchi for taking me under their wings and teaching me to be a caring and responsible professional.

Words cannot express my gratitude to my family for allowing me to pursue my dream. To my parents Alessandro and Sabina, my relatives Graziella, Ivo, Marisa, Giuseppe, Lisa, Laura, Marzia, Gino, and Lara, and to my parents-in-law Raffaela, Giulio, and Adriana, thank you. You have helped me more than you know.

This page would be incomplete without thanking my dearest friends: Rocco, for being my friend of the heart since the day I was born; Sofia and Ilaria for their support and presence despite the long distance; Veronica for believing in my ideas and understanding my heart; and Matteo, for the three years spent studying together, from which an heartfelt friendship was born.

Above all, thank you to my person, Federico, for being the most loving, determined, caring, and tenacious soul. We have followed each other across Europe, and you have been my home no matter where we were. I see you growing every day, chasing the best version of yourself, and inspiring me to do the same. Thank you for being my anchor, my confidant, and my greatest source of strength.

Amsterdam, Rome, Oslo, Valencia, Copenhagen

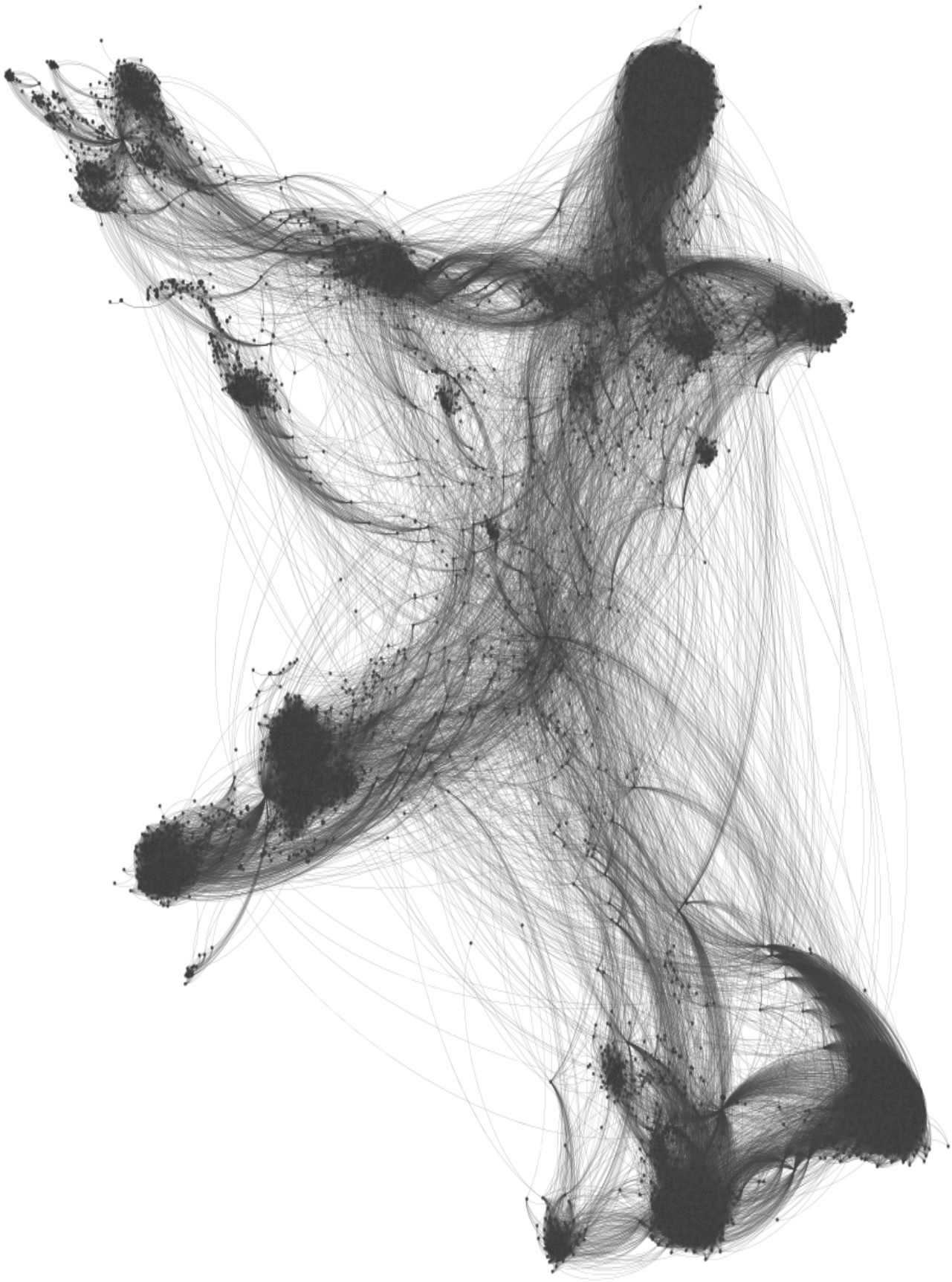

Figure 1: A synthetic network

Contents

Chapter 1

Introduction

Network science is a field that focuses on the study of complex networks, aiming to analyze and understand the structure and dynamics of interconnected systems. This work delves into the mathematical and theoretical frameworks underlying network generation and analysis, with a specific focus on spectral methods. By leveraging the properties of matrices associated with graphs, it aims to dissect and extend a spectral-based methodology for generating synthetic networks that closely resembles community structures, or other global properties, of real world networks. The work is structured to provide an understanding of both the foundational mathematics and the applications in network science of the spectra of matrices.

1.1 Research scope

This work aims to thoroughly study the spectra of matrices from a mathematical standpoint and to understand what the spectrum of both well-known and lesser-known matrices in network theory represents and what information it encodes. We then apply this knowledge to the generation of synthetic networks.

1.2 Outline

This dissertation is organized into four chapters, each addressing a specific aspect of our research. Following this introduction, the work proceeds with the study of the mathematics behind the spectra of matrices in Chapter 2. A primer on linear algebra is provided (Section 2.2) followed by the derivation of eigenvalues and eigenvectors of matrices, analysis of their properties, interpretation and application (Section 2.3). The chapter concludes with a dissertation on diagonalization and orthogonal diagonalization (Section 2.4). Chapter 3 delves into network theory, starting from an overview of foundational concepts (Section 3.2) and an introduction to standard methods for network

generation (Section 3.3), moving to the derivation of different matrices associated with graphs and, for the more interesting ones, the study of their spectral properties (Section 3.4). The final section of Chapter 3 focuses on the global properties of networks and provides some metrics to measure them (Section 3.5). In Chapter 4 the mathematical knowledge acquired in Chapter 2 is applied to the field of network theory, and more in detail to the generation of synthetic networks. The Spectral Graph Forge Method is introduced, discussed, analysed and implemented (Section 4.2), and it is then extended (Section 4.3). The conclusions of the work are provided in chapter 5

1.3 Symbols

Throughout this work we will adhere to the following notation unless otherwise stated.
Matrices are denoted by capital letters (e.g. $A, Q, \cdots$). Vectors are denoted by bold lowercase letters (e.g. $\mathbf{v}, \mathbf{w}, \mathbf{q}, \cdots$). Scalars and variables are denoted by lowercase letters (e.g. $a, i, j, \alpha, \lambda, \cdots$). Sets in linear algebra are denoted by uppercase calligraphic letters (e.g. $\mathbb{V}, \mathbb{R}, \mathbb{M}_{n,m}, \cdots$), while in network theory are denoted by uppercase calligraphic bold letters (e.g. $\mathcal{G}, \mathcal{N}, \mathcal{E}, \cdots$). Functions are denoted by lowercase (sometimes greek) letters followed by parenthesis (e.g. $\sigma(\cdots), \rho(\cdots), \cdots$). Distributions are denoted by uppercase calligraphic italic letters (e.g. $\mathscr{D}$, $\mathscr{E}, \cdots$).

Linear algebra

$\mathbb{R}$	$\triangleq$	set of real numbers.
$\mathbb{R}^n$	$\triangleq$	n-dimensional real vector space.
$\mathbb{C}$	$\triangleq$	set of complex numbers.
$\mathbf{w}^T$	$\triangleq$	transpose of vector $\mathbf{w}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	$\triangleq$	inner product of vectors $\mathbf{v}, \mathbf{w}$
$d(\mathbf{v}, \mathbf{w})$	$\triangleq$	euclidean distance between vectors $\mathbf{v}, \mathbf{w}$
$\|\mathbf{w}\|$	$\triangleq$	norm of vector $\mathbf{w}$
$\mathbb{M}_{n,m}$	$\triangleq$	set of matrices of the form $\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$. Note that: $\mathbb{M}_{n,m} \equiv \mathbb{R}^{m,n}$
$\det A$	$\triangleq$	determinant of a square matrix
$\text{diag}(\alpha_1, \cdots, \alpha_n)$	$\triangleq$	diagonal matrix whose entries on the main diagonal are listed, with all off-diagonal elements being zero.
$\dim(\mathbb{V})$	$\triangleq$	dimension of vector space $\mathbb{V}$
I	$\triangleq$	identity matrix. i.e. diagonal matrix whose entries on the main diagonal are all equal to one, with all off-diagonal elements being zero.
A^{-1}	$\triangleq$	inverse of matrix A
A^T	$\triangleq$	transpose of matrix A
$\sigma(A)$	$\triangleq$	spectrum of matrix A
$\rho(A)$	$\triangleq$	spectral radius function of matrix A
$c_A(\lambda)$	$\triangleq$	characteristic polynomial of matrix A
$E_\lambda(A)$	$\triangleq$	eigenspace of matrix A associated to eigenvalue λ
$m_a(\lambda)$	$\triangleq$	algebraic multiplicity of eigenvalue λ
$m_g(\lambda)$	$\triangleq$	geometric multiplicity of eigenvalue λ

Network theory

$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	$\triangleq$	graph $\mathcal{G}$ with set of nodes $\mathcal{N}$ and set of edges $\mathcal{E}$
$\mathcal{N}$	$\triangleq$	set of nodes of graph $\mathcal{G}$
$\mathcal{E}$	$\triangleq$	set of edges of graph $\mathcal{G}$
$e_{u,v}$	$\triangleq$	edge connecting nodes u and v
$u \sim v$	$\triangleq$	u and v are adjacent nodes
δ_u	$\triangleq$	degree of node u
δ_{max}	$\triangleq$	maximum degree of a graph $\mathcal{G}$

Chapter 2

Mathematical Framework

In this chapter we delve into the mathematics underlying the spectra of matrices that will serve as foundation for the subsequent work. This chapter draws upon the theoretical framework and methodologies delineated in (Nicholson, 2019), (Axler, 1995), (Solomon, 2015), and (Simon & Blume, 1994).

2.1 Structure of the chapter

The structure of the chapter is as follows. An introduction to the essential premises of the discussion is provided in Section 2.2. For other basic concepts that are not covered in this Section or in the main text, the reader is referred to Appendix A. Section 2.3 is dedicated to the study of eigenvalues and eigenvectors, which are at the core of the spectral analysis of networks, addressed in later chapters. The study of eigenvalues and eigenvectors begins with their definition and derivation in Subsection 2.3.1, followed by an analysis of their properties in Subsection 2.3.2 and a discussion on their interpretation and application in Subsection 2.3.3. The concept of diagonalization is elaborated in Section 2.4, beginning with an deep introduction on the topic in Subsection 2.4 which frames diagonalization under the lens of eigenvalues and eigenvectors showing how diagonalization can be interpreted as a representation of the matrix in the basis of the eigenvectors. This is followed by an analysis of orthogonal diagonalization in Section 2.4.2, which also summarize the properties of symmetric matrices in the Real Spectral Theorem and its implications.

2.2 Primer on linear algebra

2.2.1 Linear independence, inner product and euclidean distance

Definition 1 (Linear independence)**.** Let $\mathbf{v}_1, \cdots, \mathbf{v}_k \in \mathbb{R}^n$. Then, the set $\{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ is linearly independent if and only if:

$$\alpha_1\mathbf{v}_1 + \cdots + \alpha_k\mathbf{v}_k = \mathbf{0} \quad \Leftrightarrow \quad \alpha_1 = \cdots = \alpha_k = 0$$

As a result, none of these vectors can be expressed as a linear combination of the others.

Definition 2 (Inner product)**.** Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. Then, the inner product of $\mathbf{v}$ and $\mathbf{w}$, denoted as $\langle \mathbf{v}, \mathbf{w} \rangle$, is the real number:

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^T\mathbf{w} = \sum_{i=1}^{n} v_i w_i$$

Definition 3 (Orthogonality)**.** Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. Then, $\mathbf{v}, \mathbf{w}$ are orthogonal if and only if:

$$\langle \mathbf{v}, \mathbf{w} \rangle = 0$$

We write $\mathbf{v} \perp \mathbf{w}$. Two vectors that are orthogonal are always linearly independent.

Theorem 1 (Carnot Theorem)**.** *Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ and θ be the angle between them. Then, we have:*

$$\langle u, v \rangle = \|\mathbf{v}\| \cdot \|\mathbf{w}\| \cdot \cos(\theta)$$

Definition 4 (Euclidean distance)**.** Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. Then, the euclidean distance between $\mathbf{v}$ and $\mathbf{w}$, denoted as $d(\mathbf{v}, \mathbf{w})$, is the real number:

$$d(\mathbf{v}, \mathbf{w}) = \|\mathbf{v} - \mathbf{w}\| = \sqrt{\langle \mathbf{v} - \mathbf{w}, \mathbf{v} - \mathbf{w} \rangle} = \sqrt{\sum_{i=1}^{n} (v_i - w_i)^2}$$

In particular, we have that $d(\mathbf{v}, \mathbf{0}) = \|\mathbf{v}\| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle} = \sqrt{\sum_{i=1}^{n} v_i^2}$

Definition 5 (Euclidean norm)**.** Let $\mathbf{v} \in \mathbb{R}^n$. Then, the euclidean norm of $\mathbf{v}$, denoted as $\|\mathbf{v}\|$, is the real number:

$$\|\mathbf{v}\| = d(\mathbf{x}, \mathbf{0}) = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

It correspond to the length of the vector $\mathbf{v}$ seen as an oriented segment.

Definition 6 (Vector space)**.** A vector space is a non-empty set $\mathbb{V}$ of objects, called vectors, such that it is endowed with the two operations of vector addition, expressed as $\mathbf{v} + \mathbf{w} \quad \forall \mathbf{v}, \mathbf{w} \in \mathbb{V}$, and scalar multiplication, expressed as $\alpha\mathbf{v} \in \mathbb{V} \quad \forall \mathbf{v}, \mathbf{w} \in \mathbb{R} \times \mathbb{V}$, that adhere to the following axioms:

- $\mathbf{v} + \mathbf{w} \in \mathbb{V} \quad \forall\, \mathbf{v}, \mathbf{w} \in \mathbb{V}$

- $\mathbf{v} + \mathbf{w} = \mathbf{w} + \mathbf{v} \quad \forall\, \mathbf{v}, \mathbf{w} \in \mathbb{V}$
- $\mathbf{v} + (\mathbf{w} + \mathbf{z}) = (\mathbf{v} + \mathbf{w}) + \mathbf{z} \quad \forall\, \mathbf{v}, \mathbf{w}, \mathbf{z} \in \mathbb{V}$
- $\mathbf{0} \in \mathbb{V} : \mathbf{v} + \mathbf{0} = \mathbf{v} \quad \forall \mathbf{v} \in \mathbb{V}$
- $\exists \mathbf{w} \in \mathbb{V} : \mathbf{v} + \mathbf{w} = \mathbf{0} \quad \forall \mathbf{v} \in \mathbb{V}$
- $\alpha\mathbf{v} \in \mathbb{V} \quad \forall\, \alpha \in \mathbb{R},\ \mathbf{v} \in \mathbb{V}$
- $\alpha(\mathbf{v} + \mathbf{w}) = \alpha\mathbf{v} + \alpha\mathbf{w} \quad \forall\, \alpha \in \mathbb{R},\ \mathbf{v}, \mathbf{w} \in \mathbb{V}$
- $(\alpha + \beta)\mathbf{v} = \alpha\mathbf{v} + \beta\mathbf{v} \quad \forall\, \alpha, \beta \in \mathbb{R},\ \mathbf{v} \in \mathbb{V}$
- $\alpha(\beta\mathbf{v}) = (\alpha\beta)\mathbf{v} \quad \forall\, \alpha, \beta \in \mathbb{R},\ \mathbf{v} \in \mathbb{V}$
- $\mathbf{v} = \mathbf{v} \quad \forall\, \mathbf{v} \in \mathbb{V}$

Definition 7 (Vector subspace)**.** Let $\mathbb{V}$ be a vector space. Then, $\mathbb{U}$ is a subspace of $\mathbb{V}$ if $\mathbb{U} \subset \mathbb{V}$ is a set containing $\mathbf{0}_{\mathbb{V}}$, closed under the operations of addition and scalar multiplication of $\mathbb{V}$, and hence a vector space itself.

Definition 8 (Spanning set)**.** Let $\{\mathbf{v}_1, \cdots, \mathbf{v}_k\} \subset \mathbb{V}$. Then, we define the span of this set as the set generated by all the linear combinations of such vectors, i.e. the set:

$$\text{span}(\mathbf{v}_1, \cdots, \mathbf{v}_k) = \{\mathbf{w} \in \mathbb{V} \mid \exists(\alpha_1, \cdots, \alpha_k) \in \mathbb{R}^k, \mathbf{w} = \sum_{i=1}^{k} \alpha_i v^i\}$$

Which is is a subspace of $\mathbb{V}$.

Definition 9 (Basis)**.** Let $\mathbb{V}$ be a vector space. Then, a set $\mathbb{B} = \{\mathbf{b}_1, \cdots, \mathbf{b}_k\} \subset \mathbb{V}$ is a basis for $\mathbb{V}$ if $\mathbb{V} = \text{span}(\mathbf{b}_1, \cdots, \mathbf{b}_k)$ and $\mathbf{b}_1, \cdots, \mathbf{b}_k$ are linearly independent.

Proposition 1. *Let $\mathbb{V}$ be a subspace of $\mathbb{R}^n$, spanned by a set of m vectors, k of which are linearly independent. Then, $k \leq m$*

Definition 10 (Dimension)**.** Let $\mathbb{V}$ be a subspace of $\mathbb{R}^n$, and $\{\mathbf{b}_1, \cdots, \mathbf{b}_m\}$ be a basis for $\mathbb{V}$. Then, we define as dimension of $\mathbb{V}$, denoted as $\dim(\mathbb{V})$, the number of (linearly independent) vectors forming the basis. Hence, $\dim(\mathbb{V}) = m$.

2.3 Eigenvalues and eigenvectors

2.3.1 Fundamentals and derivation

Definition 11 (Eigenvalues, eigenvectors)**.** Let $A \in \mathbb{M}_{n,n}$. Then, a scalar $\lambda \in \mathbb{C}$ is defined as an eigenvalue of A, and a non-zero vector $\mathbf{q} \in \mathbb{R}^n$ as an eigenvector of A corresponding to λ (or a λ-eigenvector), if and only if the following condition is satisfied:

$$A\mathbf{q} = \lambda\mathbf{q} \tag{2.1}$$

This implies that $\mathbf{q}$ is a vector that, when multiplied by A, results in a scalar multiple of itself, with λ being the scalar.
To determine the eigenvalues λ and the corresponding eigenvectors of a matrix A, we seek non-zero solutions to the homogeneous system:

$$(\lambda I - A)\mathbf{q} = 0.$$

That has solutions if and only if the coefficient matrix $\lambda I - A$ is non-invertible, which occurs when:

$$\det(\lambda I - A) = 0.$$

(see Appendix A for definition of determinant)

Definition 12 (Characteristic polynomial)**.** Let $A \in \mathbb{M}_{n,n}$. Then, the characteristic polynomial of A, denoted by $c_A(\lambda)$, is defined as:

$$c_A(\lambda) \coloneqq \det(\lambda I - A) \tag{2.2}$$

This polynomial is of degree n in variable λ and its roots correspond to the eigenvalues of A. Hence, by the Fundamental Theorem of Algebra, $c_A(\lambda)$ has at most n distinct roots $\in \mathbb{C}$. It can be asserted that the characteristic polynomial of A is identical to that of A^T, symbolized as $c_{A^T}(\lambda)$. Consequently, the eigenvalues of A coincide with those of A^T.

Proof.

$$c_{A^T}(\lambda) = \det(\lambda I - A^T) = \det[(\lambda I - A)^T] = \det(\lambda I - A) = c_A(\lambda)$$

□

Definition 13 (Spectrum)**.** Let $A \in \mathbb{M}_{n,n}$ and $\lambda_1, \cdots, \lambda_n$ be the eigenvalues of A. Then, the set of all λ_i is called the spectrum of A, denoted as:

$$\sigma(A) = \{\lambda \in \mathbb{C} \mid c_A(\lambda) = 0\} \subset \mathbb{C}$$

Since c_A has real coefficients, when $\lambda \in \sigma(A)$, also the conjugate $\hat{\lambda} \in \sigma(A)$. The number of non-real eigenvalues is always even.

Definition 14 (Algebraic multiplicity)**.** Let λ be an eigenvalue of $A \in \mathbb{M}_{n,n}$. The algebraic multiplicity of λ, denoted as $m_a(\lambda)$, is the frequency of λ as a root of $c_A(\lambda)$.

It corresponds to the highest power of $(\lambda - z_i)$ in the factorized form of $c_A(\lambda)$, where $c_A(\lambda) = \det(\lambda I - A)$. This is expressed as:

$$c_A(\lambda) = (\lambda - z_i)^m g(\lambda) \qquad \text{with } g(\lambda) \neq 0 \text{ at } z_i$$

Definition 15 (Eigenspace)**.** Let $A \in \mathbb{M}_{n,n}$ and λ be an eigenvalue of A. Then, the eigenspace of A associated to λ, denoted as $E_\lambda(A)$, is the set:

$$E_\lambda(A) \coloneqq \{\mathbf{q} \in \mathbb{R}^n \mid A\mathbf{q} = \lambda\mathbf{q}\}$$

Since $A\mathbf{q} = \lambda\mathbf{q}$ is equivalent to $(\lambda I - A)\mathbf{q} = \mathbf{0}$:

$$E_\lambda(A) \coloneqq \{\mathbf{q} \mid (\lambda I - A)\mathbf{q} = \mathbf{0}\}$$

Because the solution set of this linear system forms the kernel of the matrix $(\lambda I - A)$:

$$E_\lambda(A) \coloneqq \ker(\lambda I - A)$$

It is a subspace of $\mathbb{C}^n$ and, if $\lambda \in \mathbb{R}^n$ it is also a subspace of $\mathbb{R}^n$.

Definition 16 (Geometric multiplicity)**.** Let λ be an eigenvalue of $A \in \mathbb{M}_{n,n}$. Then, the geometric multiplicity of λ, denoted as $m_g(\lambda)$, is the dimension of the eigenspace corresponding to λ, which is denoted as $\dim(\ker(A - \lambda I))$.

Hence, it is the maximum number of linearly independent eigenvectors that can be associated with the eigenvalue λ.

Proposition 2. *Let $A \in \mathbb{M}_{n,n}$. Then, for every eigenvalue $\lambda \in \sigma$:*

$$1 \leq m_g(\lambda) \leq m_a(\lambda) \leq n$$

2.3.2 Properties

Proposition 3. *Let $A \in \mathbb{M}_{n,n}$. Let λ_1, λ_2 be distinct eigenvalues of A, with $\mathbf{q}_1, \mathbf{q}_2$ corresponding eigenvectors. Then, $\mathbf{q_1}$ and $\mathbf{q_2}$ are linearly independent.*
Moreover, let $\sigma(A) = \lambda_1, \cdots, \lambda_i, \cdots, \lambda_k$ be distinct eigenvalues of A for $i = 1, \cdots, k$, with $k \leq n$, and $\mathbf{q}_{\lambda_{i_1}}, \cdots, \mathbf{q}_{\lambda_{i_{h_i}}}$ be linearly independent eigenvectors associated to λ_i where $h_i = m_g(\lambda_i) = dim(E_{\lambda_i}(A))$. Then, the vectors

$$\mathbf{q}_{\lambda_{1_1}}, \cdots, \mathbf{q}_{\lambda_{1_{h_1}}}, \quad \cdots, \quad \mathbf{q}_{\lambda_{i_1}}, \cdots, \mathbf{q}_{\lambda_{i_{h_i}}}, \quad \cdots, \quad \mathbf{q}_{\lambda_{k_1}}, \cdots, \mathbf{q}_{\lambda_{k_{h_k}}}$$

are linearly independent.

Proof (*by contradiction*). Assume that $\mathbf{q}_1, \cdots, \mathbf{q}_n$ are linearly dependent. Then, there exists a vector $\mathbf{q}_{p+1}$, with $p < n$ that can be expressed as a linear combination of the others in the set:

$$\mathbf{q}_{p+1} = \alpha_1\mathbf{q}_1 + \cdots + \alpha_p\mathbf{q}_p \tag{2.3}$$

where $\alpha_1, \cdots, \alpha_p$ are scalars with at least one $\alpha_i \neq 0$ for some $i = 1, \cdots, p$. Now we multiply both sides of the equation by A:

$$A\mathbf{q}_{p+1} = A(\alpha_1\mathbf{q}_1 + \cdots + \alpha_p\mathbf{q}_p)$$

Since $\mathbf{q}_i$ is an eigenvector of A corresponding to eigenvalue λ_i, we can express $A\mathbf{q}_i$ as $\lambda_i\mathbf{q}_i$ for $i = 1, \cdots, p$. Hence, we have:

$$\lambda_{p+1}\mathbf{q}_{p+1} = \alpha_1\lambda_1\mathbf{q}_1 + \cdots + \alpha_p\lambda_p\mathbf{q}_p \tag{2.4}$$

Now we multiply both sides of Equation 2.3 by λ_{p+1}:

$$\lambda_{p+1}\mathbf{q}_{p+1} = \lambda_{p+1}(\alpha_1\mathbf{q}_1 + \cdots + \alpha_p\mathbf{q}_p)$$

And we subtract it from Equation 2.4:

$$\lambda_{p+1}\mathbf{q}_{p+1} - \lambda_{p+1}\mathbf{q}_{p+1} = \alpha_1\lambda_1\mathbf{q}_1 + \cdots + \alpha_p\lambda_p\mathbf{q}_p - \lambda_{p+1}(\alpha_1\mathbf{q}_1 + \cdots + \alpha_p\mathbf{q}_p)$$

$$\mathbf{0} = \alpha_1(\lambda_1 - \lambda_{p+1})\mathbf{q}_1 + \cdots + \alpha_p(\lambda_p - \lambda_{p+1})\mathbf{q}_p \tag{2.5}$$

Since the set $\{\mathbf{q}_1 \cdots, \mathbf{q}_p\}$ is linearly independent, the coefficients $\alpha_i(\lambda_i - \lambda_{p+1})$ in Equation 2.5 must all be zero, but since $\lambda_i \neq \lambda_{p+1}$ this implies that $\alpha_i = 0\ \forall i = 1 \cdots, p$. This implies that $\mathbf{q}_{p+1} = \alpha_1\mathbf{q}_1 + \cdots + \alpha_p\mathbf{q}_p = \mathbf{0}$ which is a contradiction since eigenvectors are by definition non-zero vectors. □

The number of linearly independent vectors equals $\sum_{i=1}^{k} m_g(\lambda_i)$.

Proposition 4. *Let $A \in \mathbb{M}_{n,n}$. Then, we have $m_g(\lambda) = m_a(\lambda)\ \forall\lambda \in \sigma(A)$ if and only if $\mathbb{C}^n$ has a basis made by eigenvectors of A.*

It follows that we have n linearly independent eigenvectors.
If $\sigma(A) \subset \mathbb{R}$ then such a basis is made of real eigenvectors of A. In such a case, if expressed in the basis of eigenvectors, the matrix A becomes a diagonal matrix $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_i)$ where λ_i are the eigenvalues of A counted with their multiplicity.

Definition 17. Let $A \in \mathbb{M}_{n,n}$. Then, the spectral radius of A, denoted as $\rho(A)$, is the supremum of the magnitudes of the elements of the spectrum of A.

$$\rho(A) = \sup_{\lambda\in\sigma(A)} |\lambda|$$

2.3.3 Interpretation and applications

Eigenvalues and eigenvectors of matrices can be interpreted in a variety of ways, depending on what the matrix they are associated to represents and the context of application. Some of these interpretations are analysed below.

Linear transformations

Definition 18. Let $\mathbb{V}, \mathbb{W}$ be vector spaces with $\dim(\mathbb{V}) = n$ and $\dim(\mathbb{W}) = m$. Then, a function $T : \mathbb{V} \to \mathbb{W}$ is a linear transformation (or linear operator) if it satisfies the following properties:

- $T(\mathbf{v}+\mathbf{w}) = T(\mathbf{v}) + T(\mathbf{w}) \quad \forall\, \mathbf{v}, \mathbf{w} \in \mathbb{V}$
- $T(\alpha\mathbf{v}) = \alpha T(\mathbf{v}) \quad \forall\, \alpha \in \mathbb{R},\ \mathbf{v} \in \mathbb{V}$
- $T(0) = 0$
- $T(-\mathbf{v}) = -T(\mathbf{v}) \quad \forall \mathbf{v} \in \mathbb{V}$
- $T(\alpha_i\mathbf{v}_1 + \cdots \alpha_k\mathbf{v}_k) = \alpha_1 T(\mathbf{v}_1) + \cdots + \alpha_k T(\mathbf{v}_k) \quad \forall \alpha \in \mathbb{R},\ \mathbf{v}_1, \cdots, \mathbf{v}_k \in \mathbb{V}$

For each linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$, there exists a matrix $A \in \mathbb{M}_{n,m}$ such that:

$$T(\mathbf{v}) = A\mathbf{v} \quad \forall \mathbf{v} \in \mathbb{R}^n$$

Hence, applying a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ to a vector $\mathbf{v} \in \mathbb{R}^n$ via a matrix $A \in \mathbb{M}_{n,m}$, a new vector $A\mathbf{v} \in \mathbb{R}^m$ is produced, that is the image of $\mathbf{v}$ under T, expressed in the basis of the codomain $\mathbb{W}$.
When a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^m$ is applied to vectors, it rotates and/or scales it. The only case in which a vector is only scaled by a linear transformation, without being rotated, is the case in which the vector is an eigenvectors $\mathbf{q}$ of the matrix A associated with the linear transformation. Hence, when an eigenvector $\mathbf{q}$ of A is applied to the linear transformation T associated to A, it results in a scaled version of itself. The factor λ by which the eigenvector is scaled corresponds to the eigenvalue associated to it (a negative eigenvalue implies that the eigenvector is reversed in direction as a result of the transformation). This relationship can be expressed as:

$$T(\mathbf{q}) = \lambda\mathbf{q}$$

In other words, eigenvectors characterize the linear transformation by determining the directions along which vectors are transformed (stretched or compressed), while eigenvalues quantify the extent to which each eigenvector influences this transformation.

Principal Component Analysis (PCA)

In the context of statistics, PCA is a technique used to reduce the dimensionality of data by projecting it onto a lower-dimensional space, with the aim to retain as much variance as possible. This is achieved by projecting the data onto the principal components, which are the directions along which the data varies the most. These directions are determined by the eigenvectors of the covariance matrix associated to the data, while the eigenvalues quantify the variance of the data explained along the directions of the principal components. Therefore, the eigenvector associated with the largest eigenvalue is the principal component that captures the most variance in the data, while the eigenvectors associated with smaller eigenvalues capture portions of the remaining variance (proportional to the associated eigenvalues).

Markov chains

A Markov chain (or Markov process) is defined as a stochastic model that describes a sequence of events, each of which can assume a state with likelihood solely dependent on the state achieved in the preceding event. The model can be encoded in a stochastic matrix A whose entries $a_{i,j}$ signify the transition probabilities from state i to state j. Notably, each entry is non-negative and each row sums to 1, reflecting the total probability of transitioning from a specific state to any other state within the system. Because of the Perron-Frobenius Theorem (Perron, 1907), (Frobenius, 1912) the matrix A, being square and having non-negative entries, is shown to have a unique dominant real eigenvalue, which in the case of Markov matrices equals 1 by the Gerschgorin Theorem (Gerschgorin, 1931). The eigenvector corresponding to this eigenvalue represents the stationary distribution of the Markov chain, meaning the convergence of the system to a steady state. The entries of this eigenvector represent the unconditional probabilities of being in each state of the system (i.e. the probability of an event assuming a state regardless of the state of the previous event).

PageRank

The concept of the PageRank algorithm (Brin & Page, 1998), which ranks the importance of a web page by determining the number of incoming hyperlinks it receives from other pages and the importance of the pages linking to it, can be interpreted as an eigenvector problem. The web can be represented as a matrix A having as entries $a_{i,j}$ the existence of a hyperlink from page j to page i. This original matrix is usually normalized and adjusted by adding a damping factor d, to ensure that every page can be reached from every other page and that the probabilities converge to a steady state. The modified matrix G is defined as:

$$G = dP + \frac{(1-d)}{n}E$$

where P is the normalized adjacency matrix (in which the rows sum to 1), n is the total number of pages and E is a matrix of all ones. In order to determine the PageRank scores of the pages, it is necessary to find the stationary distribution of the Markov chain modelled by the matrix G. Hence, we need to find a vector $\mathbf{q}$ such that:

$$\mathbf{q} = G\mathbf{q}$$

Notably, $\mathbf{q}$ is the eigenvector of G corresponding to the eigenvalue 1. The entries of this vector give the PageRank of each page, normalized to sum to 1 (Wilf, 2002), (Moler, 2002), (Chandrashekhar et al., 2022).

2.4 Diagonalization

2.4.1 The role of eigenvalues and eigenvectors

Definition 19 (Diagonal matrix)**.** Let $\Lambda \in \mathbb{M}_{n,n}$. Then, it is diagonal if all entries outside its principal diagonal are zero. This is expressed as $\text{diag}(\lambda_1, \cdots, \lambda_n)$ where λ_i is a scalar.

Λ can be represented as:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

where $\lambda_1, \ldots, \lambda_n$ are scalar values.

Definition 20 (Diagonalizable matrix)**.** Let $A \in \mathbb{M}_{n,n}$. Then, it is diagonalizable if it exists a matrix $Q \in \mathbb{M}_{n,n}$, referred as diagonalizing matrix, that is invertible such that:

$$\Lambda = Q^{-1}AQ \text{ is diagonal} \tag{2.6}$$

(see Appendix A for definition of invertible matrices)

A matrix is considered to be diagonalizable if it is similar to a diagonal matrix (see Appendix A for definition of similar matrices).

Proposition 5. *Let $A \in \mathbb{M}_{n,n}$ have n distinct eigenvalues $\lambda_1, \cdots, \lambda_n$ and n corresponding eigenvectors $\mathbf{q}_1, \cdots, \mathbf{q}_n$. Let $Q = [\mathbf{q}_1, \cdots, \mathbf{q}_n]$ be an invertible matrix and $\Lambda = diag(\lambda_1, \cdots, \lambda_n)$. Then:*

$$A = Q\Lambda Q^{-1} \tag{2.7}$$

Proof. To derive matrix Λ, we start from Equation 2.6 and bring to the RHS matrix Q^{-1}:

$$AQ = Q\Lambda$$

Upon expanding the terms, we have:

$$A[\mathbf{q}_1, \mathbf{q}_1, \cdots, \mathbf{q}_n] = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

After performing matrix multiplication:

$$A[\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n] = [\lambda_1\mathbf{q}_1, \lambda_2\mathbf{q}_2, \ldots, \lambda_n\mathbf{q}_n]$$

This implies that for each i, where $1 \leq i \leq n$, the relationship:

$$A\mathbf{q}_i = \lambda_i\mathbf{q}_i$$

holds true, indicating that each $\mathbf{q}_i$ is an eigenvector of A associated with the eigenvalue λ_i □

We can choose the diagonalizing matrix Q so that the eigenvalues λ_i appear in any order we want along the main diagonal of Λ.
It is interesting to notice that the diagonal matrix Λ corresponds to matrix A expressed in the basis of its eigenvectors, and that matrix Q represents the change of basis matrix (which maps matrix A from the canonical basis into the basis of its eigenvectors). Therefore, $A = Q\Lambda Q^{-1}$ can be seen as the change of basis formula.

Proposition 6. *Let $A \in \mathbb{M}_{n,n}$. Then, it is diagonalizable if and only if every eigenvalue λ of A has $m_g(\lambda) = m_a(\lambda) = k \leq n$ linearly independent associated eigenvectors.*

Hence, a matrix $A \in \mathbb{M}_{n,n}$ is diagonalizable if and only the algebraic multiplicity of each eigenvalue λ equals $\dim[E_\lambda(A)]$. Note that in the case in which the algebraic multiplicity of an eigenvalue λ is less than $\dim[E_\lambda(A)]$, then A is not diagonalizable.
Therefore, we can state that a matrix $A \in \mathbb{M}_{n,n}$ is diagonalizable if and only if has n linearly independent eigenvectors $\{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$ (as a necessary and sufficient condition for Q to be invertible is the linear independence of its columns). This brings us to the following result:

Proposition 7. *Let $A \in \mathbb{M}_{n,n}$. Then it is diagonalizable (with real spectrum) if and only if R^n admits as a basis the set of its eigenvectors. That is, if and only if the spectrum of A is made of simple eigenvalues, which means:*

$$\sigma(A) \subset \mathbb{R} \text{ and } m_g(\lambda) = m_a(\lambda),\ \forall \lambda \in \sigma(A)$$

In the specific case in which $A \in \mathbb{M}_{n,n}$ has n distinct real eigenvalues, then A is diagonalizable with real eigenvectors. The fact that eigenvalues are real means that $\sigma(A) \subset \mathbb{R}$, while the fact that are distinct implies that $m_g(\lambda) = m_a(\lambda) = 1,\ \forall \lambda \in \sigma(A)$, since the algebraic multiplicity acts as an upper bound for the geometric multiplicity, hence the eigenvalues are simple. It is worth specify that the existence of n distinct real eigenvalues is a sufficient but not necessary condition for a matrix to be diagonalizable: there might be cases in which a matrix has $k \leq n$ distinct real eigenvalues, but the matrix will still be diagonalizable as long as the algebraic multiplicity of each eigenvalue equals its geometric multiplicity, as there will still be n linearly independent eigenvectors forming a basis for $\mathbb{R}^n$. If the eigenvalues are not real, then A is diagonalizable with complex eigenvectors.

2.4.2 Orthogonal diagonalization

Definition 21 (Orthogonal matrix)**.** Let $Q \in \mathbb{M}_{n,n}$. Then, it is orthogonal if and only if its column vectors form an orthonormal set.

Orthogonal matrices are invertible and $Q^{-1} = Q^T$. Hence, $QQ^T = I$.

Proposition 8. *Let $Q \in \mathbb{M}_{n,n}$. Then, (1) Q is invertible and $Q^{-1} = Q^T$ if and only if (2) the columns (and the rows) of Q are orthonormal.*

Proof $1 \Rightarrow 2$. Let $Q = [\mathbf{q}_1, \cdots \mathbf{q}_n]$. We know that $Q^{-1} = Q^T \Leftrightarrow QQ^T = I$. This implies $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = 0$ if $i \neq j$ and $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = 1$ if $i = j$. Hence, the columns of Q are orthonormal. □

Proof $2 \Rightarrow 1$. Let $Q = [\mathbf{q}_1, \cdots \mathbf{q}_n]$. We know that the columns of Q are orthonormal if and only if $Q^TQ = I$. This implies that $Q^T = Q^{-1}$, hence Q is invertible and $Q^{-1} = Q^T$. □

Definition 22 (Symmetric matrix)**.** Let $A \in \mathbb{M}_{n,n}$. Then, it is symmetric if and only if $A = A^T$.

Let $A \in \mathbb{M}_{n,n}$. Then, its eigenvalues are guaranteed to be real if A is symmetric. Note that a matrix may possess real eigenvalues without necessarily being symmetric.
Relevant properties of symmetric matrices are listed in the Real Spectral Theorem, also referred to as Principal Axes Theorem:

Theorem 2 (Real Spectral Theorem)**.** *Let $A \in \mathbb{M}_{n,n}$. Then, (1) A is symmetric if and only if (2) A has n real orthogonal eigenvectors.*

Proof $1 \Rightarrow 2$ (*by induction*). Let $A \in \mathbb{M}_{n,n}$ be real and symmetric.
If $A \in \mathbb{M}_{1,1}$, then A has one eigenvalue $\lambda_1 = a_{11}$ and one λ-eigenvector $\mathbf{q}_1 = [1]$.
From induction hypothesis, assume $A \in \mathbb{M}_{n-1,n-1}$ real and symmetric is orthogonally diagonalizable.
Now, let $A \in \mathbb{M}_{n,n}$ real and symmetric, let λ_1 be an eigenvalue of A and $\mathbf{w}$ be the corresponding eigenvector. Starting from this vector, we construct an orthonormal basis for $\mathbb{R}^n$: $\{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$ such that $\mathbf{q}_1 = \frac{1}{\|\mathbf{w}\|}\mathbf{w}$. Let $Q = [\mathbf{q}_1, \cdots, \mathbf{q}_n]$. Then:

$$AQ = [A\mathbf{q}_1, \cdots, A\mathbf{q}_n]$$

Because of Equation 2.1, we have:

$$AQ = [\lambda\mathbf{q}_1, A\mathbf{q}_2 \cdots, A\mathbf{q}_n]$$

Rewriting the product, we have:

$$AQ = Q \begin{bmatrix} \lambda & \mathbf{d} \\ \mathbf{0} & A' \end{bmatrix}$$

For some $d \in \mathbb{M}_{1,n-1}$ and $A' \in \mathbb{M}_{n-1,n-1}$.

Because Q is orthonormal by construction, we have:

$$Q^TAQ = \begin{bmatrix} \lambda & \mathbf{d} \\ \mathbf{0} & A' \end{bmatrix}$$

Taking the transpose of both sides, we have:

$$Q^TA^TQ = \begin{bmatrix} \lambda & \mathbf{0} \\ \mathbf{d} & A'^T \end{bmatrix}$$

Hence, because $A = A^T$ we have $Q^TAQ = Q^TA^TQ$, therefore $\mathbf{d} = \mathbf{0}$ and $A' = A'^T$. This means that A' is symmetric and by induction hypothesis, it has $n-1$ orthogonal eigenvectors. Hence, adding $\mathbf{q}_1$, A has n orthogonal eigenvectors. □

Proof $2 \Rightarrow 1$. Assuming that $Q = [\mathbf{q}_1, \cdots, \mathbf{q}_n]$ is an orthogonal matrix, we have that $Q^T = Q^{-1}$. Hence, we can express A as:

$$A = Q\Lambda Q^{-1} = Q\Lambda Q^T$$

To prove that A is symmetric, we need to show that $A^T = A$. We have:

$$A^T = (Q\Lambda Q^T)^T = (Q^{-1})^T \Lambda^T Q^T = Q\Lambda Q^T = A$$

Hence, $A^T = A$ and A is symmetric. □

Summarizing, from the Real Spectral Theorem we have that given a symmetric matrix $A \in \mathbb{M}_{n,n}$, it has exactly n eigenvalues $\lambda_1, \cdots, \lambda_n$, counting their multiplicities, moreover the corresponding eigenvectors are mutually orthogonal, hence they can be chosen to form an orthonormal basis $\{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ of $\mathbb{R}^n$ (as from any linearly independent set of eigenvectors we can obtain an orthonormal set using the Gram-Schmidt algorithm). Consequently, there exists an orthogonal matrix $Q = [\mathbf{b}_1, \cdots, \mathbf{b}_n] \in \mathbb{M}_{n,n}$ (which has the property: $Q^{-1} = Q^T$) and a diagonal matrix $\Lambda = \text{diag}\,(\lambda_1, \cdots, \lambda_n) \in \mathbb{M}_{n,n}$, such that $A = Q\Lambda Q^{-1} = Q\Lambda Q^T$.

Definition 23 (Orthogonally diagonalizable matrix)**.** Let $A \in \mathbb{M}_{n,n}$. Then, it is orthogonally diagonalizable if there exists an orthogonal matrix Q such that:

$$A = Q\Lambda Q^{-1} = Q\Lambda Q^T$$

This happens if and only if A is symmetric.

Proposition 9. *Let $A \in \mathbb{M}_{n,n}$ be a symmetric matrix. Then,*

$$A = Q\Lambda Q^T = \sum_{i=1}^{n} \lambda_i \mathbf{q}_i \mathbf{q}_i^T \tag{2.8}$$

Proof. From 2 we know that:

$$A = Q\Lambda Q^T$$

Expanding the terms, we have:

$$= \sum_{i=1}^{n} \left(\mathbf{q}_i \sum_{j=1}^{n} \lambda_{i,j} \mathbf{q}_j^T \right)$$

Expressing the summations under a slightly different notation, we can revrite:

$$= \sum_{i \in [1,n]} \left(\mathbf{q}_i \sum_{j \in [1,n]} \lambda_{i,j} \mathbf{q}_j^T \right)$$

We now differenciate the case in which $i = j$ from the case in which $i \neq j$:

$$= \sum_{i \in [1,n]} \left(\mathbf{q}_i \sum_{j \in \{i\}} \lambda_{i,j} \mathbf{q}_j^T \quad + \sum_{j \in [1,n] \setminus \{i\}} \lambda_{i,j} \mathbf{q}_j^T \right)$$

Since $\lambda_{i,j} = 0 \; \forall j \in [1,n] \setminus \{i\}$, the term $\sum_{j \in [1,n] \setminus \{i\}} \lambda_{i,j} \mathbf{q}_j^T$ equals zero. Hence:

$$= \sum_{i \in [1,n]} \left(\mathbf{q}_i \sum_{j \in \{i\}} \lambda_{i,j} \mathbf{q}_j^T \right)$$

As we are considering only cases in which $j = i$, we have that $\lambda_{i,j} = \lambda_i$, hence:

$$= \sum_{i \in [1,n]} \left(\mathbf{q}_i \lambda_i \mathbf{q}_i^T \right)$$

Rearrangin the terms, we have:

$$= \sum_{i \in [1,n]} \lambda_i \mathbf{q}_i \mathbf{q}_i^T$$

Expressing the summation in the initial notation:

$$= \sum_{i=1}^{n} \lambda_i \mathbf{q}_i \mathbf{q}_i^T$$

□

Chapter 3

Network Theory Framework

This chapter aims to provide a foundation in network theory and to understand the information encoded in the spectra of matrices when they represent graphs. Hence, after an overview on the basic notation and concepts on network theory, the work delves into the analysis of the spectral properties of some matrices associated to graphs, fundamental for understanding community structures within networks. The spectrum and the eigenvectors of these matrices, in fact, encode information on the underlying structure of graphs. Despite significant progress, the literature in this field is in ongoing discovery, with many facets yet to be fully understood. A comprehensive overview of the field is provided in (Cvetković et al., 1980), upon which this chapter is grounded, along with the frameworks developed in (Cvetković, 2009) and (Cvetković & Gutman, 2009).

3.1 Structure of the chapter

The structure of the chapter is as follows. An primer on the basic concepts on network theory is provided in Section 3.2, covering fundamental concepts in Subsection 3.2.1 and definitions and notation in Subsection 3.2.2. Section 3.3 is dedicated to the overview of standard methods for network generation. The models analysed are the Erdős-Rényi model in Subsection 3.3.1, the Configuration model in Subsection 3.3.2, the Barabási-Albert model (or Preferential Attachment model) in Subsection 3.3.3, and the Stochastic Block model in Subsection 3.3.4. Section 3.4 delves into the derivation of matrices associated with graphs, and for the more interesting ones it provides an overview of their spectral properties. The matrices analysed are the adjacency matrix in Subsection 3.4.1, the Seidel matrix in Subsection 3.4.2, the incidence matrix in Subsection 3.4.3, the degree matrix in Subsection 3.4.4, the Laplacian matrix in Subsection 3.4.5, the sum connectivity matrix in Subsection 3.4.6, the General Zagreb matrix in Subsection 3.4.7, the modularity matrix in Subsection 3.4.8, the transition matrix of a random walk in Subsection 3.4.9, the non-backtracking matrix in Subsection 3.4.10, and the Bethe-Hessian matrix in Subsection 3.4.11. Section 3.5 focuses on the global properties of

networks, focusing on community structure in Subsection 3.5.1, connectivity in Subsection 3.5.2, assortativity in Subsection 3.5.3 and energy in Subsection 3.5.4, alongside some metrics to measure them.

3.2 Primer on network theory

3.2.1 Fundamental elements

Networks can be represented through graphs, sets of entities and relations between them. A graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is characterized by two sets: the set $\mathcal{N}$ comprising the entities of interest, referred to as nodes (or vertex) and a set $\mathcal{E}$ of pairs of nodes $e_{u,v}$, referred to as edges (or links), which represent the relations interlinking nodes u, v. The set of nodes $\mathcal{N}$ is a finite, non-empty set whose cardinality $n = |\mathcal{N}|$ represents the order of the graph. The set of edges $\mathcal{E}$ is a subset of the set of all possible pairs of nodes, $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$. Its cardinality $m = |\mathcal{E}|$ represents the size of the graph.
Depending on the nature of its edges, networks can be directed or undirected, weighted or unweighted and can have self-loops. A graph is directed if each edge $e_{u,v} \in \mathcal{E}$ is an ordered pair, expressing a directional relation from node u to node v. As opposed, a graph is undirected if edges are unordered pairs, signifying symmetric relationships between nodes. A graph is weighted if each edge is associated with a numerical value (the weight) which quantify the intensity of the relation between its pairs of nodes. Instead, a graph is unweighted if each edge represent only the existence of the relation. A graph has self-loops if permits edges of the form $e_{u,v} \in \mathcal{E}$, where a node connects to itself. During this work, we will focus on undirected, unweighted graphs without self-loops, unless otherwise specified.

3.2.2 Notation

Definition 24. (adjacent nodes) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u, v \in \mathcal{N}$. Then u, v are said to be neighbours or adjacent if $e_{u,v} \in \mathcal{E}$. To say that two nodes are adjacent, we write $u \sim v$.

Definition 25. (Triangle) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u, v, w \in \mathcal{N}$. Then, a triangle is a set of three nodes $\{u, v, w\}$ such that $e_{u,v}, e_{v,w}, e_{w,u} \in \mathcal{E}$.

Definition 26. (Complement of a graph) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Then, the complement of the graph $\mathcal{G}' = (\mathcal{N}, \mathcal{E}')$ is a graph where $\mathcal{E}'$ is the set of edges that are not present in $\mathcal{E}$ (excluding self-loops).

Definition 27. (Degree of a node) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u \in \mathcal{N}$. Then, the degree δ_u of a node u is the number of edges $e_{u,v}$ incident to u.

For undirected graphs, the degree of a node is equivalent to the number of its neighbours.

Definition 28. (Degrees of a graph) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Then, the maximum degree of the graph $\delta_{\max}$ is the maximum degree of its nodes, while the minimum degree of the graph $\delta_{\min}$ is the minimum degree of its nodes.

Definition 29. (Path) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $e_{u,v} \in \mathcal{E}$ be an edge connecting nodes $u, v \in \mathcal{N}$. Then, we define as path from node u_1 to node v_k a sequence of edges $\{e_{u_1,v_1}, \cdots, e_{u_h,v_h}, \cdots, e_{u_k,v_k}\}$ such that $u_h = v_{h-1}$ for $h = 2, \cdots, k$, and all nodes $u_1, \ldots, v_k$ being distinct (except possibly for $u_1 = v_k$ if the path is closed).

The length k of the path is the number of its edges. The path with the smallest length connecting u to v is the shortest path between u and v, and its length is defined as distance distance between nodes u and v. The maximum length of the shortest path between any pair of nodes is the diameter of the graph. A graph is said to be connected if there exists a path between any pair of nodes belonging to it.

Definition 30. (Walk) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $e_{u,v} \in \mathcal{E}$ be an edge connecting nodes $u, v \in \mathcal{N}$. Then, we define as walk from node u_1 to node v_k a sequence of edges $\{e_{u_1,v_1}, \cdots, e_{u_h,v_h}, \cdots, e_{u_k,v_k}\}$ such that $u_h = v_{h-1}$ for $h = 2, \cdots, k$.

The definition of walk is less stringent with respect to the definition of path, as it allows for the presence of repeated nodes in the sequence of edges.

Definition 31. (Random walk) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $e_{u,v} \in \mathcal{E}$ be an edge connecting nodes $u, v \in \mathcal{N}$. Then, a random walk is a walk where each edge from the current node u_h (or equivalently v_{h-1}) to the next node v_h is chosen randomly with uniform probability from among all neighbors of u_h. Each step is independent of the previous steps.

A random walk is a Markov chain, as the probability of transitioning from one node to another depends exclusively on the current node and not on the sequence of previous nodes visited.

Definition 32. (Non-backtracking random walk) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $e_{u,v} \in \mathcal{E}$ be an edge connecting nodes $u, v \in \mathcal{N}$. Then, a non-backtracking random walk is a random walk where the next node v_h is chosen randomly with uniform probability from among all neighbors of u_h except for the previous node u_{h-1}.

Note that a non-backtracking random walk differs from a random walk and it is not a Markov chain as the probability of transitioning from one node to another depends not only on the current node, but also on the previous node visited.

3.3 Standard methods for network generation

In network theory, a wide range of methodologies have been developed for the generation of networks. These methods are designed to generate graphs that mimic different properties of real-world networks, and differs for targeted property, goal and complexity. The most common methods for network generation are based on random graph models and are designed to generate networks with specific (usually local) properties.

3.3.1 Erdős-Rényi Model

A first model is the Erdős-Rényi Model $\mathcal{G}'(n, m)$, in which the generated random graph is a function of the number of nodes n and the number of edges m. Under this model, a graph is chosen sampling with uniform probability from the set of all possible graphs having n nodes and m edges. A variation of this method is the Erdős-Rényi-Gilbert Model $\mathcal{G}'(n, p)$, in which the generated random graph is a function of the number of nodes n and the probability p is the parameter of one Bernoulli trial determining the existence of each potential edge. The model proceeds by connecting each pair of nodes with probability p, independently of the other pairs. In this case, the number of edges m is not fixed, but it is a random variable with expected value $p\binom{n}{2}$.

3.3.2 Configuration Model

The Configuration Model $\mathcal{G}'(n, \mathscr{D})$ follows a different approach, by which the generated random graph is a function of the number of nodes n and an input degree sequence $\mathscr{D} = \{\delta_1, \cdots, \delta_n\}$. The method proceeds by creating a stub for each edge (meaning, connecting the edge to one node only), and then connecting the stubs randomly. Under this methodology the output graph preserves information related to the number of nodes and the degree of each node, however it does not necessarily preserve information on the community structure of the input network.

3.3.3 Barabási-Albert Model

The Barabási-Albert Model or (Linear Preferential Attachment Model) $\mathcal{G}'(n, \gamma)$ is a model that generates random graphs by starting with a small number of nodes and adding the others iteratively, connecting each new node to γ existing nodes with a probability p proportional to their degree, and hence changes at each iteration. Under this model, the output graphs are characterized by a power-law degree distribution, meaning that the probability that a node has a degree δ is proportional to $\delta^{-\gamma}$.

3.3.4 Stochastic Block Model

A model focusing on community structure is the Stochastic Block Model $\mathcal{G}'(n, \mathcal{B}, P)$, which generates graphs as function of the number of nodes n and a set of blocks $\mathcal{B} = \{B_1, \cdots, B_k\}$, where each block B_i is a disjoint subset of the set of nodes, and a symmetric matrix $\mathbb{P} \in \mathbb{M}_{k,k}$ of edge probabilities. The model proceeds by connecting two nodes i, j with a probability $p_{i,j}$ that depends on the block to which the nodes belong. Notably, when the values on the main diagonal of $\mathbb{P}$ are higher than the values off the main diagonal, the output graph is likely to be partitioned in k distinct communities, as nodes belonging to the same block are more likely to be connected than nodes belonging to different blocks. However, while the output graph is characterized by a partition into communities, the model does not allow for a high degree of customization in the resulting graph structure.

These methods are standard for generating networks and are comparatively straightforward. A comprehensive examination of network generation methodologies, emphasizing algorithmic strategies, is documented in (Penschuck et al., 2020). We will expand further the topic of network generation in Chapter 4, which is dedicated to a methodology based on spectral properties of matrices associated to graphs.

3.4 Derivation and spectral properties of matrices associated to graphs

Network theory frequently employs linear algebra to represent networks and analyze network properties. Because matrices facilitate the characterization of pairwise relations they are chosen to represent relations within graphs; furthermore, the application of algebraic methods allow to better understand characteristics and dynamics within networks. In particular, the spectral analysis of matrices can reveal aspects of network structure.
One fundamental application of matrices in representing relationships within graphs is through the adjacency relation.

3.4.1 Adjacency matrix

The adjacency relation of nodes can be represented through the adjacency matrix A, which represents the structure of the graph, encoding the relations between nodes.

Definition 33. (Adjacency matrix) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an undirected graph where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and $u, v \in \mathcal{N}$. Then, the adjacency

matrix of $\mathcal{G}$ is $A \in \mathbb{M}_{n,n}$, whose entries are:

$$a_{u,v} = \begin{cases} 1 & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases} \quad \text{for } u, v \in \mathcal{N}$$

For undirected graphs, as the adjacency relation is bidirectional, the associated adjacency matrix is real and symmetric, hence it has real spectrum and, by the Real Spectral Theorem 2 it has orthogonal eigenvectors.

Proposition 10. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an undirected graph where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and $A \in \mathbb{M}_{n,n}$ the adjacency matrix of $\mathcal{G}$. Then, the relation $A\boldsymbol{q} = \lambda \boldsymbol{q}$ can be interpreted as:*

$$\lambda q_u = \sum_{v \sim u} q_v$$

where $\boldsymbol{q} = (q_1, \cdots, q_u, \cdots, q_n)^T$ for $u = 1, \cdots, n$ and the summation is performed over the neighbours of node u.

Hence, the eigenvectors and the spectrum of the adjacency matrix encode information about the structural properties of the graph, as each eigenvector component quantifies the connectivity influence of nodes.

Proposition 11. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an undirected graph where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes, $u, v \in \mathcal{N}$, and $A \in \mathbb{M}_{n,n}$ the adjacency matrix of $\mathcal{G}$. Then, by the Perron-Frobenius Theorem, A has a non-negative dominant eigenvalue $\ddot{\lambda}$ which equals the maximum degree of the graph δ_{max}.*

Proof.

$$\ddot{\lambda}\mathbf{q}_i = (A\mathbf{q})_i = \sum_{j=1}^{n} a_{i,j} q_j \leq \sum_{j=1}^{n} a_{i,j} q_j = \mathbf{q}_i \delta_i$$

□

3.4.2 Seidel Matrix

Another way to represent the adjacency relation in a graph is through the Seidel matrix, which captures it in a different way with respect to the adjacency matrix.

Definition 34. (Seidel matrix) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes, $u, v \in \mathcal{N}$, $A \in \mathbb{M}_{n,n}$ be its adjacency matrix, and $A^C \in$ the adjacency matrix of the complement of the graph associated to A (recall Definition 26). Then, the Seidel matrix of $\mathcal{G}$ is $S \in \mathbb{M}_{n,n}$, defined as:

$$S = A - A^C$$

Hence, the Seidel matrix has entries defined as:

$$s_{u,v} = \begin{cases} -1 & \text{if } u \neq v, e_{u,v} \in \mathcal{E} \\ 1 & \text{if } u \neq v, e_{u,v} \notin \mathcal{E} \\ 0 & \text{if } u = v \end{cases}$$

3.4.3 Incidence matrix

A matrix that captures the relationship between nodes and edges in a graph is the incidence matrix, which capture the incidence relation among nodes and edges.

Definition 35 (Incidence matrix)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes, m be the number of edges, and $u, v \in \mathcal{N}$. Then, the incidence matrix of $\mathcal{G}$ is $C \in \mathbb{M}_{n,m}$, whose entries are:

$$c_{u,e_{u,v}} = \begin{cases} 1 & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

3.4.4 Degree matrix

Moving forward, we define the degree matrix, which captures the degree of each node in the graph.

Definition 36 (Degree matrix)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let δ_u be the degree of node $u \in \mathcal{N}$. Then, the degree matrix of $\mathcal{G}$ is a diagonal matrix $D \in \mathbb{M}_{n,n}$, whose entries are:

$$d_{u,v} = \begin{cases} \delta_u & \text{if } u = v \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } u, v \in \mathcal{N}$$

Hence, $D = \text{diag}(\delta_1, \cdots, \delta_n)$.
The degree matrix is particularly useful because it encodes information about the connectivity of the graph, meaning the ability of the elements within the network to reach one another. For this reason, the degree matrix is often used to derive other fundamental matrices in network theory, such as the Laplacian matrix.

3.4.5 Laplacian matrix

The Laplacian matrix can be seen as the discrete counterpart of the Laplacian operator. Playing a similar role in graph theory, it quantifies how much a graph's value at a particular node (i.e., the number of connections) deviates from the values at its neighbouring nodes. Hence, it measures the change in the graph's structure.

Definition 37 (Laplacian matrix)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes, $A \in \mathbb{M}_{n,n}$ the adjacency matrix, and $D \in \mathbb{M}_{n,n}$ the degree matrix of $\mathcal{G}$. Then, the Laplacian matrix of $\mathcal{G}$ is $L \in \mathbb{M}_{n,n}$ defined as:

$$L = D - A$$

An alternative derivation of the Laplacian matrix is the following:

$$L = C^T C$$

Because L is real and symmetric, it has an orthogonal set of eigenvectors and real eigenvalues.

Proposition 12. *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and $L \in \mathbb{M}_{n,n}$ the Laplacian matrix of $\mathcal{G}$. Then, the smallest eigenvalue of L is $\dot{\lambda} = 0$ and its corresponding eigenvector is $\mathbf{q}_1 = (1, \cdots, 1)^T$.*

The algebraic multiplicity of 0 as an eigenvalue of the Laplacian matrix corresponds to the number of connected components in the associated graph.
The literature studying graph energy and graph irregularities also exploits matrices which are similar to the Laplacian: the signless Laplacian and the normalized Laplacian. The signless Laplacian matrix is defined as $L_{\text{signless}} = D + A$, while the normalized Laplacian matrix is defined as $L_{\text{norm}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.

3.4.6 Sum Connectivity matrix

Another matrix encoding encoding information on the connectivity of a graph is the Sum Connectivity matrix.

Definition 38. (Sum Connectivity matrix) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and $u, v \in \mathcal{N}$. Then, the Sum Connectivity matrix of $\mathcal{G}$ is $S_C \in \mathbb{M}_{n,n}$, whose entries are:

$$s_{C_{u,v}} = \begin{cases} \frac{1}{\sqrt{\delta_u \delta_v}} & \text{if } u \sim v \\ 0 & \text{otherwise} \end{cases}$$

For undirected graphs, the sum connectivity matrix is symmetric. The entries of its main diagonal are zero, while the off-diagonal entries are normalized by the product of the square root of the degrees of the nodes they connect.

3.4.7 General Zagreb Matrix

Moving forward, we define the General Zagreb matrix, firstly introduced in the field of Chemistry Physics. It encodes information on the degree sequence of a graph, generalizing its Zagreb indices (Horoldagva & Das, 2023).

Definition 39. (General Zagreb matrix) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes, $A \in \mathbb{M}_{n,n}$ its adjacency matrix and $D \in \mathbb{M}_{n,n}$ its degree matrix. Then, the General Zagreb matrix of $\mathcal{G}$ is $G_z \in \mathbb{M}_{n,n}$, defined as:

$$G_z = D^3 + A$$

3.4.8 Modularity matrix

The structure of a graph can be analysed through the modularity matrix, which encodes information on the number of edges within nodes of the same community with respect to a random graph generated under the configuration model (and hence with identical expected degree sequence). This matrix was first introduced in (Newman, 2006) for the study of community structure in networks.

Definition 40 (Modularity matrix)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edge, let n be the number of nodes, $u, v \in \mathcal{N}$, $A \in \mathbb{M}_{n,n}$ be its adjacency matrix. Then, the modularity matrix of $\mathcal{G}$ is $B \in \mathbb{M}_{n,n}$, whose entries are:

$$b_{u,v} = a_{u,v} - \frac{\delta_u \delta_v}{2m} \tag{3.1}$$

For undirected graphs, the modularity matrix is real and symmetric. An important property of this matrix is that it always has an eigenvector of ones associated to the zero eigenvalue, as the sum along its rows and columns is zero. Moreover, the eigenvector $\ddot{\mathbf{q}}$ associated with the largest eigenvalue $\ddot{\lambda}$ of the modularity matrix captures important information about the global structure of a graph. By dividing the nodes according to the signs of their corresponding elements in this eigenvector, we obtain groups that are more densely connected internally and sparsely connected with nodes in the other groups. The division of the network obtained following this method, is the one that maximizes the modularity (see Definition 48) of the network. In the case in which $\ddot{\mathbf{q}}$ is exactly the eigenvector of ones, then positive modularity cannot be achieved by any division of the network. The eigenvector $\ddot{\mathbf{q}}$ provides also information about the strength of the nodes' community affiliations: nodes corresponding to elements of large magnitude are strongly associated with their particular community, while nodes associated with elements of small magnitude (close to zero) have weak affiliation to any community.

The modularity matrix is one of the most famous matrices in network theory for its powerful capability to detect community structure in networks. More recently, also some other matrices with the same purpose have been proposed.

3.4.9 Transition matrix of a random walk

Recall Definition 31, which introduces the concept of a random walk. Based on this definition, we define the transition matrix associated with a random walk. Specifically, this matrix is a stochastic matrix whose entries represent the probability of transitioning from one node to another.

Definition 41. (Transition matrix of a random walk) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $e_{u,v} \in \mathcal{E}$ be an edge connecting nodes $u, v \in \mathcal{N}$. Then, the transition matrix P of a random walk on $\mathcal{G}$ is a stochastic matrix $P \in \mathbb{M}_{n,n}$ defined as:

$$P = D^{-1}A$$

By the characteristics of Markov chains, summarized in Paragraph 2.3.3, the stationary distribution of a random walk is described by the eigenvector associated with the largest eigenvalue $\ddot{\lambda}$ of the Transition matrix, which equals 1, and hence by solving the equation $P\mathbf{q} = \mathbf{q}$.

3.4.10 Non-backtracking matrix

Recall Definition 32 of non-backtracking random walks of a graph, and the fact that they are not Markov chains. It is possible to create a new graph $\hat{\mathcal{G}} = (\hat{\mathcal{N}}, \hat{\mathcal{E}})$ from the original one $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ such that a random walk on $\hat{\mathcal{G}}$ is equivalent to a non-backtracking random walk on $\mathcal{G}$ (Glover, 2021). The new graph $\hat{\mathcal{G}}$ is characterized by:

- $\hat{\mathcal{N}}$, the set obtained by taking each undirected edge $e_{u,v} \in \mathcal{E}$ and replacing it with two directed edges $e_{u,v}$ and $e_{v,u}$. Hence, the cardinality of this set will be $2m$ (where m is the cardinality of $\mathcal{E}$).
- $\hat{\mathcal{E}}$, the set defined as:

$$\{(u, v) \sim (k, l) : v = k, u \neq l\}$$

We define the adjacency matrix of graph $\hat{\mathcal{G}}$ as the non-backtracking matrix (or Hashimoto matrix) H (Hashimoto, 1989) of the original graph $\mathcal{G}$, as follows:

Definition 42 (Non-backtracking matrix)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let m be the number of edges and $u, v, k, l \in \mathcal{N}$. Then, the non-backtracking matrix of $\mathcal{G}$ is the matrix $H \in \mathbb{M}_{2m,2m}$ whose entries are defined as:

$$h_{(u,v),(k,l)} = \begin{cases} 1 & \text{if } v = k, u \neq l \\ 0 & \text{otherwise} \end{cases}$$

Note that, given a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, every random walk in its associated non-backtracking matrix H is equivalent to a non-backtracking random walk in its adjacency matrix A.
The non-backtracking matrix encodes information on the structure of links of a graph (Krzakalaa et al., 2013). Specifically, its complex spectrum can be divided by a circle of radius $\sqrt{\rho(H)}$ (where $\rho(H)$ is its spectral radius) (Saade et al., 2014b) in:

- a bulk of uninformative eigenvalues lying inside the circle

– informative (and real) eigenvalues lying outside the circle (the number of these eigenvalues, for graphs generated under the Stochastic Block Model, equals the number of communities in the graph)

All the eigenvalues λ of B that are different from ± 1 are roots of the polynomial:

$$\det\left[(\lambda^2 - 1)I - \lambda A + D\right]$$

Which corresponds to the Ihara-Bass formula for the graph zeta function.

3.4.11 Bethe-Hessian matrix

Definition 43. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let A be its adjacency matrix and D be its degree matrix. Then, the Bethe-Hessian matrix of $\mathcal{G}$ is $Z \in \mathbb{M}_{n,n}$ defined as:

$$Z(r) = (r^2 - 1)I - rA + D$$

where $r \in \mathbb{R}$ is a parameter and $I \in \mathbb{R}$ is the identity matrix.

The Bethe-Hessian is real and symmetric, hence it has real spectrum and orthogonal eigenvectors. Its determinant is given by the Ihara-Bass formula for the graph zeta function, and is hence related to the spectrum of the non-backtracking matrix:

$$\det Z(\lambda) = \det\left[(\lambda^2 - 1)I - \lambda A + D\right]$$

For large enough r, all the eigenvalues of Z are positive. It is possible to translate all the informative eigenvalues of the non-backtracking matrix into negative eigenvalues of the Bethe-Hessian matrix by choosing $r = \sqrt{\rho(H)}$ (Saade et al., 2014a).

3.5 Global properties of networks

Global measures of a graph are quantitative metrics providing information on the structure and the properties of a network. They include various properties, such as community structure, assortativity, connectivity, and energy of a graph.

3.5.1 Community structure

Definition 44 (Community)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Then, a community (or cluster) is a subset $\mathcal{C} \subseteq \mathcal{N}$ of nodes that are more densely connected to each other than to the rest of the graph.

A community is therefore characterized by a high density of edges within nodes belonging to it (high cohesion) and by a low density of edges between nodes belonging to it and the rest of the graph (high separation). Despite this intuitive conceptual notion of community the discipline lacks of a single, precise, and universally agreed-upon definition of community. This is largely due to the inherent complexity and variability of network structures, which exhibit diverse characteristics that necessitate different measures for their accurate identification.

Definition 45. (Community detection) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $\mathcal{C} \subseteq \mathcal{N}$ be a community. Then, the community detection (or clustering) is the process of partitioning $\mathcal{N}$ into subsets such that each subset, denoted as a community, comprises vertices that are more densely connected internally than with vertices outside the subset.

Because of the absence of an unequivocal definition of what constitutes a community, the literature developed a wide range of methodologies for community detection, each distinct in metrics, approach, and underlying principles. This variety allows for the identification of different types of communities, yet it also prevents the methodologies from being directly compared.
We proceed by introducing two different methods for community detection, which are the average clustering coefficient and the number of partitions under modularity maximization.

Average clustering coefficient

The concept of clustering coefficient (Watts & Strogatz, 1998) is a measure of the degree to which nodes in a graph tend to create strongly connected groups.

Definition 46. (Local clustering coefficient) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u \in \mathcal{N}$. Then, the local clustering coefficient c_u of a node u is the fraction of pairs of neighbours of u that are connected with each other.

$$c_u = \frac{2\tau_u}{\delta_u(\delta_u - 1)}$$

where τ_u is the number of triangles through node u and δ_u is the degree of node u.

These local clustering coefficient is 1 if every neighbour of node u is also connected to every other node within the neighbourhood, and 0 if no neighbour of node u is connected to any other node within the neighbourhood.

Definition 47. (average clustering coefficient) Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and $u, v \in \mathcal{N}$. Then, the clustering coefficient c of $\mathcal{G}$ is the average of the local clustering coefficients of all nodes in $\mathcal{N}$:

$$c = \frac{1}{n} \sum_{u \in \mathcal{N}} c_u$$

A high value of average clustering coefficient (close to 1) implies a tightly-knit network with few interconnected communities, while a small value (close to 0) implies a lack of cohesion and a more fragmented community structure.

Modularity maximization

Modularity is a measure of the strength by which a network is divided into communities.

Definition 48 (Modularity)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, $u, v \in \mathcal{N}$, and B be its modularity matrix. Then, the modularity Q of $\mathcal{G}$ is:

$$q = \frac{1}{2m} \sum_{u,v \in \mathcal{N}} \left(b_{u,v} \delta_{\mathcal{C}_u, \mathcal{C}_v} \right)$$

where m is the number of edges in $\mathcal{G}$, $\delta_{\mathcal{C}_u, \mathcal{C}_v}$ is the Kronecker delta function (meaning that $\delta_{\mathcal{C}_u, \mathcal{C}_v} = 1$ if $\mathcal{C}_u = \mathcal{C}_v$ and 0 otherwise), and $\mathcal{C}_u$ is the community of node u.

The modularity q is therefore a measure of the quality of a partition of a graph into communities. It assumes values in $[-1, 1]$, where a value close to 1 indicates a partitioning of the graph into cohesive communities with few links connecting them, 0 indicates a partitioning with fewer differences in connectivity within communities than between communities, while a negative value means that the graph have few edges within communities with respect to a randomly generated graph (under the configuration model). It is worth noting that for most networks the modularity is positive.

3.5.2 Connectivity

Connectivity refers to the degree to which the nodes in a graph are connected to one another through paths in the set of edges. It is an important property for analysing the robustness of networks.
To quantitatively measure connectivity, we employ the Beta index and the Sum connectivity index.

Beta index

Definition 49 (Beta index)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let n be the number of nodes and m be the number of edges. Then, the beta index $(\beta - \text{index})$ of $\mathcal{G}$ is defined as:

$$\beta = \frac{m}{n}$$

In a network with a fixed number of nodes, the higher the number of edges, the higher the number of possible paths in the network, hence the higher the $\beta - \text{index}$.

Sum connectivity index

Definition 50 (Sum connectivity index)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let δ_u be the degree of node u. Then, the sum connectivity index s_c of $\mathcal{G}$

is defined as:

$$s_c = \frac{\sum_{e_{u,v} \in \mathcal{E}} \frac{1}{\sqrt{\delta_u \delta_v}}}{\frac{n(n-1)}{2}}$$

where $\deg(i)$ and $\deg(j)$ are the degrees of nodes i and j, respectively.

The sum connectivity index accounts for the influence of the degree of individual nodes when participating in an edge, normalized by the maximum number of possible edges in the network.

3.5.3 Assortativity

Assortativity measures the tendency of nodes to connect with other nodes that are similar in some attribute, such as degree. Hence, it indicates the tendency for nodes to attach to others that are similar (assortative mixing) or different (disassortative mixing).
To quantitatively measure assortativity, the degree assortativity coefficient is commonly used.

Degree assortativity coefficient

Definition 51 (Degree Assortativity Coefficient)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an undirected graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u, v \in \mathcal{N}$, and δ_u be the degree of node u. Then, the degree assortativity coefficient r of $\mathcal{G}$ is defined as the Pearson correlation coefficient of degree between pairs of linked nodes. It is given by:

$$r = \frac{\sum_{u,v}(\delta_u \cdot \delta_u(\mathscr{E}_{uv} - p_u q_v))}{\sigma_p \sigma_q}$$

where, $\mathscr{E}_{uv}$ is the joint probability distribution of the degrees, p_i and q_j are the proportions of edges connected to nodes of degrees i and j, respectively, and σ_p and σ_q are the standard deviations of the distributions.

Hence, it measures the tendency of having a directed edge $e_{u,v}$ such that, $\delta_u = \delta_v$. A positive value of r indicates assortative mixing, while a negative value indicates disassortative mixing.

3.5.4 Energy

Definition 52 (Energy)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u \in \mathcal{N}$, and $A \in \mathbb{M}_{n,n}$ be its adjacency matrix and $\lambda_1, \cdots, \lambda_n$ its eigenvalues. Then, the energy e of $\mathcal{G}$ is defined as:

$$e = \sum_{u=1}^{n} |\lambda_u|$$

One specific measure related to the energy of a graph is the Forgotten Zagreb Index.

Forgotten Zagreb index

Definition 53 (Forgotten Zagreb Index)**.** Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph, where $\mathcal{N}$ is the set of nodes and $\mathcal{E}$ is the set of edges, let $u \in \mathcal{N}$, and δ_u be the degree of node u. Let $\deg(i)$ be the degree of node i. Then, the forgotten Zagreb Index f_z of $\mathcal{G}$ is defined as:

$$f_z = \sum_{u \in \mathcal{N}} (\delta_u)^3$$

Chapter 4

Spectral Based Methodology for Network Generation

In this chapter, we delve into a practical application of spectral properties of matrices explored in Chapter 2 within the domain of network theory. Drawing upon the concepts introduced in Chapter 3, we concentrate on the generation of synthetic networks. In particular, this chapter is dedicated to a mathematical analysis and subsequent elaboration of the methodology proposed by (Baldesi et al., 2018): the Spectral Graph Forge (SGF), designed for the generation of synthetic graphs preserving a determined rate of information on the community structure of an input network. Specifically, the algorithm is firstly understood, then replicated and applied to real world networks, and finally extended. Throughout the chapter, many network visualizations can be found, for which the Gephi software was employed.

4.1 Structure of the chapter

The structure of the chapter is outlined as follows. The Spectral Graph Forge method is discerned in Section 4.2. Within this section, the raw concept of the method is explained in Subsection 4.2.1, followed by a detailed mathematical formulation and coding implementation of the algorithm in Subsection 4.2.2. Subsequently, the application of the Spectral Graph Forge algorithm to two distinct examples featuring networks of distinct sizes, a small one and a large one, is presented in Subsection 4.2.3 and in Subsection 4.2.4 respectively. The results presented in these two subsections were computed through the Python function detailed in the section before, of which the comprehensive code is available in Appendix B. Throughout these subsections, for display purposes, all numerical values are rounded to the second decimal place. Some proposals of new transformations to use in the Spectral Graph Forge method are studied in Section 4.3. Within the section, the transformations are first introduced in Subsection 4.3.1, then a first broad graphical comparison of the synthetic networks

generated by the transformations is presented in Subsection 4.3.2 and finally their performance is evaluated in Subsection 4.3.3. The code used to implement the transformations and perform the analysis is available in Appendix C.

4.2 Spectral Graph Forge method

4.2.1 Foundational Concept

As anticipated in the introduction to this chapter, we are interested into generating networks with a target global property: community structure. This is particularly useful in contexts that require reliance on the high-level structure of original graphs, such as modelling diffusion, or sharing network data while anonymizing sensitive information of single nodes. To achieve this, the work of Baldesi et all. leverages the spectral properties of the modularity matrix associated to a graph, which, as seen in 3.4.8, is constructed in such a way that it encodes information about the community structure of the graph. The Spectral Graph Forge methodology is engineered to target the community structure of an input network, specifically its modularity, by leveraging the fact that this information is encoded in the eigenstructure of the network's modularity matrix (or an equivalent matrix tailored to a different measure). More precisely, the quantity of information retained by each eigenvector of this matrix is directly proportional to the magnitude of its corresponding eigenvalue. Thus, the creation of an output network involves generating its modularity matrix through a low-rank approximation of the modularity matrix of the input network. This process is controlled by the parameter α, which dictates the fraction of eigenvectors to be retained, which, in turn, determines the extent of community structure information preserved versus the level of anonymization achieved.

4.2.2 Mathematical formulation and coding implementation

Having delineated the fundamental concept of the Spectral Graph Forge method, we now delve into the mathematical formulation of the algorithm, detailing the specific steps it incorporates from a mathematical viewpoint. Additionally, we have implemented the algorithm via a Python function, adhering as closely as possible to the steps outlined in the section. Following the mathematical explanation of each step, the corresponding code segment is presented. The full code is available in Appendix B.

Input matrix computation

The Spectral Graph Forge algorithm starts by considering as input a matrix M which can be either the adjacency matrix A or a real-valued, symmetry-preserving transformation of A. The chosen input matrix will be referred as M.

An example of such transformation is the modularity matrix B, whose entries are derived from the

ones of A as in Equation 3.1:

$$b_{u,v} = a_{u,v} - \frac{\delta_u \delta_v}{2m}$$

When the modularity matrix is selected as input, we preserve both the information of this matrix and the node degrees, which are necessary to back-transform the approximated modularity matrix to an approximated adjacency matrix.

```
# Computation of M
A = nx.adjacency_matrix(G).toarray()
M = np.zeros((len(A), len(A)))
if transformation == "identity":
    M = A
if transformation == "modularity":
    degrees = [G.degree[node] for node in G.nodes()]
    m = sum(degrees) / 2
    B = A - np.outer(degrees, degrees) / (2 * m)
    M = B
```

Listing 4.1: Input matrix computation

Low rank α approximation

Because of the properties of M (reality and symmetry), we know from Equation 2.8 that it can be expressed as a sum of its eigenvectors $\mathbf{q}_1, \cdots, \mathbf{q}_n$, weighted by their associated eigenvalues $\lambda_1, \cdots, \lambda_n$:

$$M = Q\Lambda Q^T = \sum_{i=1}^{n} \lambda_i \mathbf{q}_i \mathbf{q}_i^T$$

We proceed by scaling the eigenvectors of M to have unit norm, and by sorting the eigenvalues and associated eigenvectors in the summation by the absolute value of the eigenvalues such that the condition $|\lambda_1| \geq \cdots \geq |\lambda_n|$ is satisfied. This way, the eigenvalues and eigenvectors are now in order of decreasing magnitude, and hence by the quantity of information they encode on the community structure.

At this point, we perform the low rank approximation on M by truncating the summation to the first $\lceil \alpha n \rceil$ terms:

$$\tilde{M} = \sum_{i=1}^{\lceil \alpha n \rceil} \lambda_i \mathbf{q}_i \mathbf{q}_i^T \qquad \text{with } \alpha \in [0, 1]$$

The parameter α determines the fraction of eigenvectors and eigenvalues of M that we want to retain, and hence consists in the retention rate of information on the community structure of the input network that we want to maintain. The parameter is set by the user, and it is a trade-off between the quantity of information on the community structure that we want to preserve and the level of anonymization of the input network that we want to achieve. The higher the value of α, the more information on the community structure is preserved, and the lower the value of α, the more anonymized the input network becomes.

```
    # Computation of eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eigh(M)
    eigenvectors = eigenvectors.T

    # sorting the eigenvectors, eigenvalues
    paired_sorted_list = sorted(zip(eigenvalues, eigenvectors), key=lambda x: abs(x[0]),
        reverse=True)
    eigenvalues_sorted, eigenvectors_sorted = zip(*paired_sorted_list)

    # Computation of M_tilde
    M_tilde = np.zeros((eigenvectors_sorted[0].shape[0], eigenvectors_sorted[0].shape[0]))
    for i in range(math.ceil(alpha * len(eigenvalues))):
        contribution = eigenvalues_sorted[i] * np.outer(eigenvectors_sorted[i],
            eigenvectors_sorted[i])
        M_tilde += contribution
```

Listing 4.2: Low rank α approximation

Back transformation

Moving forward, depending on the transformation we used to obtained M in the first step, we proceed by back-transform $\tilde{M}$, to obtain an approximated adjacency matrix $\tilde{A}$. For example, if we started from the modularity matrix B, we can obtain $\tilde{A}$ by reincorporating the information of the the node degrees as follows:

$$\tilde{a}_{u,v} = \tilde{m}_{u,v} + \frac{\delta_u \delta_v}{2m} \tag{4.1}$$

This matrix however is not guaranteed to be a valid adjacency matrix, as it might contain entries that differs from zero and one.

```
    # Computation of A_tilde
    A_tilde = np.zeros((len(M_tilde), len(M_tilde)))
    if transformation == "identity":
        A_tilde = M_tilde
    if transformation == "modularity":
        A_tilde = M_tilde + np.outer(degrees, degrees) / (2 * m)
```

Listing 4.3: Back transformation

Normalization

To obtain a valid adjacency matrix, we proceed by scaling the entries of $\tilde{A}$ to the interval $[0, 1]$ obtaining a new matrix $\ddot{A}$. This procedure furtherly masks the original adjacency matrix A, enhancing the anonymization of the input network. This step can be performed using different kind of scaling functions. Baldesi et al. propose the following three:

- Logistic $(\tilde{a}_{u,v}, k)$

$$\ddot{a}_{u,v} = \frac{1}{1 + e^{(0.5 - \tilde{a}_{u,v})k}} \qquad \text{with } k \in [2, 10]$$

- Truncate ($\tilde{a}_{u,v}$)

$$\ddot{a}_{u,v} = \begin{cases} 0 & \text{if } \tilde{a}_{u,v} \leq 0 \\ 1 & \text{if } \tilde{a}_{u,v} \geq 1 \\ \tilde{a}_{u,v} & \text{otherwise} \end{cases}$$

- Scale ($(\tilde{a}_{u,v})$)

$$\ddot{a}_{u,v} = \frac{\tilde{a}_{u,v} - \min_{s,t}(\tilde{a}_{s,t})}{\max_{s,t}(\tilde{a}_{s,t}) - \min_{s,t}(\tilde{a}_{s,t})}$$

Baldesi et al. identified the truncation method as the most effective. Conversely, they found the scaling method to introduce distortions (dependent on the variance between the minimum and maximum values). Furthermore, they reported that the logistic function, despite being optimized with a parameter $k = 6$, preserved only a minimal amount of the structure of the graph.

```
    A_dots = np.zeros((len(A_tilde), len(A_tilde[0])))

    # logistic
    if normalization_type == "logistic":
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                A_dots[i][j] = 1 / (1 + math.exp((0.5 - A_tilde[i][j])*k))

    # truncation
    if normalization_type == "truncate":
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                if A_tilde[i][j] < 0:
                    A_dots[i][j] = 0
                elif A_tilde[i][j] > 1:
                    A_dots[i][j] = 1
                else:
                    A_dots[i][j] = A_tilde[i][j]

    # scaling
    if normalization_type == "scale":
        A_tilde_flattened = A_tilde.flatten()
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                A_dots[i][j] = (A_tilde[i][j] - min(A_tilde_flattened)) / (
                    max(A_tilde_flattened) - min(A_tilde_flattened))
```

Listing 4.4: Normalization

Adjacency matrix generation

Since matrix $\ddot{A}$ has entries bounded in the interval $[0, 1]$, we can conceive each of them as a probability of existence of an edge between the corresponding nodes. Hence, we draw matrix A' by sampling

from a Bernoulli distribution that has as parameters the entries of $\ddot{A}$:

$$a'_{i,j} \sim \text{Bernoulli}(\ddot{a}_{i,j})$$

Then, to obtain a valid adjacency matrix, we set $a'_{i,j} = 0 \; \forall i = j$ to avoid self-loops, and we symmetrize the matrix by setting $a'_{i,j} = a'_{j,i} \forall j > i$.
The resulting matrix A' is the output of the Spectral Graph Forge algorithm, representing the adjacency matrix of a random graph that preserves a determined rate of information on the community structure of the input network.

```
    # Bernoulli sampling
    A_prime = np.zeros((len(A_dots), len(A_dots)))
    for i in range(len(A_tilde)):
        for j in range(len(A_tilde)):
            u = random.uniform(0, 1)
            if u < A_dots[i][j]:
                A_prime[i][j] = 1
                A_prime[j][i] = 1
            else:
                A_prime[i][j] = 0
                A_prime[j][i] = 0
            if i == j:
                A_prime[i][j] = 0
```

Listing 4.5: Adjacency matrix generation

4.2.3 Application to a small-sized network

We apply the Spectral Graph Forge algorithm to a small-sized network to further illustrate its steps and relative computations. The network under consideration is the well-known network of the Florentine Families, which is a small-sized network consisting of 15 nodes and 20 edges that maps the marriage ties between the most influential families of Florence in the 15th century. A representation of the network is provided in the figure below:

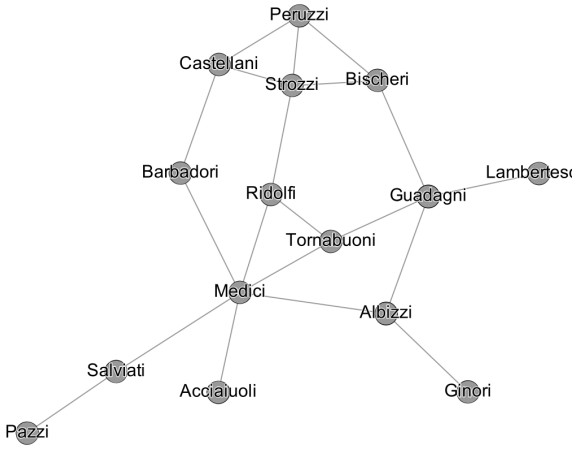

Figure 4.1: Florentine Families Graph

Input matrix

To begin, we compute the adjacency matrix A of the Florentine Families network. Given our focus on the modularity matrix, we also proceed to the calculation of the network's modularity matrix B, which will serve as the input matrix M for the Spectral Graph Forge algorithm. The matrices A and B are as follows:

$$A = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}$$

$$M = B = \begin{bmatrix}
-0.02 & 0.85 & -0.08 & -0.08 & -0.1 & -0.05 & -0.08 & -0.08 & -0.08 & -0.05 & -0.02 & -0.08 & -0.1 & -0.02 & -0.02 \\
0.85 & -0.9 & -0.45 & -0.45 & -0.6 & 0.7 & 0.55 & 0.55 & 0.55 & 0.7 & -0.15 & -0.45 & -0.6 & -0.15 & -0.15 \\
-0.08 & -0.45 & -0.22 & 0.78 & 0.7 & 0.85 & -0.22 & -0.22 & -0.22 & -0.15 & -0.08 & -0.22 & -0.3 & -0.08 & -0.08 \\
-0.08 & -0.45 & 0.78 & -0.22 & 0.7 & -0.15 & -0.22 & -0.22 & -0.22 & -0.15 & -0.08 & 0.78 & -0.3 & -0.08 & -0.08 \\
-0.1 & -0.6 & 0.7 & 0.7 & -0.4 & -0.2 & 0.7 & -0.3 & -0.3 & -0.2 & -0.1 & 0.7 & -0.4 & -0.1 & -0.1 \\
-0.05 & 0.7 & 0.85 & -0.15 & -0.2 & -0.1 & -0.15 & -0.15 & -0.15 & -0.1 & -0.05 & -0.15 & -0.2 & -0.05 & -0.05 \\
-0.08 & 0.55 & -0.22 & -0.22 & 0.7 & -0.15 & -0.22 & 0.78 & -0.22 & -0.15 & -0.08 & -0.22 & -0.3 & -0.08 & -0.08 \\
-0.08 & 0.55 & -0.22 & -0.22 & -0.3 & -0.15 & 0.78 & -0.22 & -0.22 & -0.15 & -0.08 & -0.22 & 0.7 & -0.08 & -0.08 \\
-0.08 & 0.55 & -0.22 & -0.22 & -0.3 & -0.15 & -0.22 & -0.22 & -0.22 & -0.15 & -0.08 & -0.22 & 0.7 & 0.92 & -0.08 \\
-0.05 & 0.7 & -0.15 & -0.15 & -0.2 & -0.1 & -0.15 & -0.15 & -0.15 & -0.1 & 0.95 & -0.15 & -0.2 & -0.05 & -0.05 \\
-0.02 & -0.15 & -0.08 & -0.08 & -0.1 & -0.05 & -0.08 & -0.08 & -0.08 & 0.95 & -0.02 & -0.08 & -0.1 & -0.02 & -0.02 \\
-0.08 & -0.45 & -0.22 & 0.78 & 0.7 & -0.15 & -0.22 & -0.22 & -0.22 & -0.15 & -0.08 & -0.22 & 0.7 & -0.08 & -0.08 \\
-0.1 & -0.6 & -0.3 & -0.3 & -0.4 & -0.2 & -0.3 & 0.7 & 0.7 & -0.2 & -0.1 & 0.7 & -0.4 & -0.1 & 0.9 \\
-0.02 & -0.15 & -0.08 & -0.08 & -0.1 & -0.05 & -0.08 & -0.08 & 0.92 & -0.05 & -0.02 & -0.08 & -0.1 & -0.02 & -0.02 \\
-0.02 & -0.15 & -0.08 & -0.08 & -0.1 & -0.05 & -0.08 & -0.08 & -0.08 & -0.05 & -0.02 & -0.08 & 0.9 & -0.02 & -0.02
\end{bmatrix}$$

Low rank α approximation

To compute the low-rank α approximation of matrix M, we start by determining its spectrum and eigenvectors. Below, we present the spectrum of M, with eigenvalues ordered by decreasing magnitude:

$$\sigma(M) = \{-2.8, 2.43, -2.07, -1.88, 1.71, -1.19, 1.11, 0.95, -0.89, -0.79, 0.6, -0.58, 0.26, -0.21, 0.0\}$$

The eigenvectors, normalized and ordered according to their associated eigenvalues as they are shown above, are:

$$q_1 = \begin{bmatrix} 0.2 \\ -0.64 \\ -0.08 \\ -0.05 \\ -0.15 \\ 0.2 \\ 0.11 \\ 0.26 \\ 0.35 \\ 0.21 \\ -0.1 \\ 0.14 \\ -0.42 \\ -0.15 \\ 0.12 \end{bmatrix}, q_2 = \begin{bmatrix} -0.16 \\ -0.39 \\ 0.36 \\ 0.46 \\ 0.44 \\ -0.02 \\ -0.08 \\ -0.23 \\ -0.24 \\ -0.2 \\ -0.08 \\ 0.34 \\ -0.07 \\ -0.1 \\ -0.03 \end{bmatrix}, q_3 = \begin{bmatrix} 0.12 \\ -0.27 \\ -0.46 \\ 0.3 \\ 0.27 \\ 0.34 \\ 0.01 \\ -0.07 \\ -0.08 \\ 0.16 \\ -0.08 \\ -0.47 \\ 0.36 \\ 0.03 \\ -0.18 \end{bmatrix}, q_4 = \begin{bmatrix} -0.09 \\ 0.18 \\ -0.22 \\ -0.12 \\ 0.55 \\ 0.04 \\ -0.56 \\ 0.38 \\ 0.11 \\ -0.11 \\ 0.07 \\ -0.05 \\ -0.28 \\ -0.05 \\ 0.16 \end{bmatrix}, q_5 = \begin{bmatrix} -0.16 \\ -0.29 \\ -0.25 \\ -0.04 \\ -0.1 \\ -0.31 \\ -0.17 \\ 0.08 \\ 0.27 \\ -0.24 \\ -0.14 \\ 0.26 \\ 0.57 \\ 0.16 \\ 0.34 \end{bmatrix}, q_6 = \begin{bmatrix} -0.13 \\ 0.15 \\ -0.33 \\ 0.51 \\ -0.27 \\ 0.14 \\ 0.14 \\ -0.08 \\ 0.16 \\ -0.45 \\ 0.37 \\ -0.03 \\ -0.23 \\ -0.14 \\ 0.18 \end{bmatrix}, q_7 = \begin{bmatrix} 0.03 \\ 0.09 \\ -0.13 \\ -0.05 \\ 0.19 \\ -0.13 \\ 0.57 \\ 0.5 \\ -0.27 \\ -0.29 \\ -0.31 \\ 0.04 \\ 0.05 \\ -0.29 \\ 0.0 \end{bmatrix}, q_8 = \begin{bmatrix} 0.1 \\ 0.11 \\ 0.26 \\ -0.01 \\ -0.03 \\ 0.35 \\ -0.03 \\ -0.04 \\ 0.33 \\ -0.49 \\ -0.53 \\ -0.18 \\ -0.07 \\ 0.33 \\ -0.09 \end{bmatrix},$$

$$q_9 = \begin{bmatrix} 0.03 \\ 0.03 \\ 0.21 \\ -0.12 \\ 0.23 \\ -0.16 \\ 0.09 \\ -0.19 \\ 0.58 \\ -0.07 \\ 0.13 \\ -0.19 \\ 0.2 \\ -0.59 \\ -0.17 \end{bmatrix}, q_{10} = \begin{bmatrix} 0.25 \\ -0.24 \\ -0.05 \\ -0.48 \\ 0.13 \\ 0.26 \\ 0.09 \\ -0.09 \\ -0.16 \\ -0.43 \\ 0.5 \\ 0.17 \\ 0.09 \\ 0.14 \\ -0.17 \end{bmatrix}, q_{11} = \begin{bmatrix} 0.03 \\ 0.01 \\ 0.25 \\ -0.09 \\ -0.25 \\ 0.46 \\ -0.26 \\ 0.06 \\ -0.29 \\ -0.04 \\ -0.05 \\ -0.08 \\ 0.26 \\ -0.47 \\ 0.45 \end{bmatrix}, q_{12} = \begin{bmatrix} 0.29 \\ -0.2 \\ 0.34 \\ 0.27 \\ -0.21 \\ -0.33 \\ -0.23 \\ 0.46 \\ -0.08 \\ -0.17 \\ 0.24 \\ -0.36 \\ 0.07 \\ 0.09 \\ -0.18 \end{bmatrix}, q_{13} = \begin{bmatrix} 0.76 \\ 0.2 \\ -0.23 \\ 0.15 \\ -0.04 \\ -0.15 \\ -0.23 \\ -0.21 \\ -0.05 \\ -0.05 \\ -0.21 \\ 0.32 \\ -0.01 \\ -0.19 \\ -0.06 \end{bmatrix}, q_{14} = \begin{bmatrix} -0.25 \\ 0.1 \\ -0.06 \\ 0.1 \\ -0.22 \\ 0.28 \\ -0.2 \\ 0.31 \\ 0.08 \\ 0.05 \\ -0.01 \\ 0.41 \\ 0.19 \\ -0.15 \\ -0.64 \end{bmatrix}, q_{15} = \begin{bmatrix} -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \\ -0.26 \end{bmatrix}$$

Subsequently, we can express the matrix M in the form $M = Q\Lambda Q^T$, where the matrices Q and Λ are defined as follows:

$$Q = \begin{bmatrix} 0.2 & -0.16 & 0.12 & -0.09 & -0.16 & -0.13 & 0.03 & 0.1 & 0.03 & 0.25 & 0.03 & 0.29 & 0.76 & -0.25 & -0.26 \\ -0.64 & -0.39 & -0.27 & 0.18 & -0.29 & 0.15 & 0.09 & 0.11 & 0.03 & -0.24 & 0.01 & -0.2 & 0.2 & 0.1 & -0.26 \\ -0.08 & 0.36 & -0.46 & -0.22 & -0.25 & -0.33 & -0.13 & 0.26 & 0.21 & -0.05 & 0.25 & 0.34 & -0.23 & -0.06 & -0.26 \\ -0.05 & 0.46 & 0.3 & -0.12 & -0.04 & 0.51 & -0.05 & -0.01 & -0.12 & -0.48 & -0.09 & 0.27 & 0.15 & 0.1 & -0.26 \\ -0.15 & 0.44 & 0.27 & 0.55 & -0.1 & -0.27 & 0.19 & -0.03 & 0.23 & 0.13 & -0.25 & -0.21 & -0.04 & -0.22 & -0.26 \\ 0.2 & -0.02 & 0.34 & 0.04 & -0.31 & 0.14 & -0.13 & 0.35 & -0.16 & 0.26 & 0.46 & -0.33 & -0.15 & 0.28 & -0.26 \\ 0.11 & -0.08 & 0.01 & -0.56 & -0.17 & 0.14 & 0.57 & -0.03 & 0.09 & 0.09 & -0.26 & -0.23 & -0.23 & -0.2 & -0.26 \\ 0.26 & -0.23 & -0.07 & 0.38 & 0.08 & -0.08 & 0.5 & -0.04 & -0.19 & -0.09 & 0.06 & 0.46 & -0.21 & 0.31 & -0.26 \\ 0.35 & -0.24 & -0.08 & 0.11 & 0.27 & 0.16 & -0.27 & 0.33 & 0.58 & -0.16 & -0.29 & -0.08 & -0.05 & 0.08 & -0.26 \\ 0.21 & -0.2 & 0.16 & -0.11 & -0.24 & -0.45 & -0.29 & -0.49 & -0.07 & -0.43 & -0.04 & -0.17 & -0.05 & 0.05 & -0.26 \\ -0.1 & -0.08 & -0.08 & 0.07 & -0.14 & 0.37 & -0.31 & -0.53 & 0.13 & 0.5 & -0.05 & 0.24 & -0.21 & -0.01 & -0.26 \\ 0.14 & 0.34 & -0.47 & -0.05 & 0.26 & -0.03 & 0.04 & -0.18 & -0.19 & 0.17 & -0.08 & -0.36 & 0.32 & 0.41 & -0.26 \\ -0.42 & -0.07 & 0.36 & -0.28 & 0.57 & -0.23 & 0.05 & -0.07 & 0.2 & 0.09 & 0.26 & 0.07 & -0.01 & 0.19 & -0.26 \\ -0.15 & -0.1 & 0.03 & -0.05 & 0.16 & -0.14 & -0.29 & 0.33 & -0.59 & 0.14 & -0.47 & 0.09 & -0.19 & -0.15 & -0.26 \\ 0.12 & -0.03 & -0.18 & 0.16 & 0.34 & 0.18 & 0.0 & -0.09 & -0.17 & -0.17 & 0.45 & -0.18 & -0.06 & -0.64 & -0.26 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix}
-2.8 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 2.43 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -2.07 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & -1.88 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 1.71 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.19 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.11 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.95 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.89 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.79 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.6 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.58 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.26 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.21 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0
\end{bmatrix}$$

$$Q^T = \begin{bmatrix}
0.2 & -0.64 & -0.08 & -0.05 & -0.15 & 0.2 & 0.11 & 0.26 & 0.35 & 0.21 & -0.1 & 0.14 & -0.42 & -0.15 & 0.12 \\
-0.16 & -0.39 & 0.36 & 0.46 & 0.44 & -0.02 & -0.08 & -0.23 & -0.24 & -0.2 & -0.08 & 0.34 & -0.07 & -0.1 & -0.03 \\
0.12 & -0.27 & -0.46 & 0.3 & 0.27 & 0.34 & 0.01 & -0.07 & -0.08 & 0.16 & -0.08 & -0.47 & 0.36 & 0.03 & -0.18 \\
-0.09 & 0.18 & -0.22 & -0.12 & 0.55 & 0.04 & -0.56 & 0.38 & 0.11 & -0.11 & 0.07 & -0.05 & -0.28 & -0.05 & 0.16 \\
-0.16 & -0.29 & -0.25 & -0.04 & -0.1 & -0.31 & -0.17 & 0.08 & 0.27 & -0.24 & -0.14 & 0.26 & 0.57 & 0.16 & 0.34 \\
-0.13 & 0.15 & -0.33 & 0.51 & -0.27 & 0.14 & 0.14 & -0.08 & 0.16 & -0.45 & 0.37 & -0.03 & -0.23 & -0.14 & 0.18 \\
0.03 & 0.09 & -0.13 & -0.05 & 0.19 & -0.13 & 0.57 & 0.5 & -0.27 & -0.29 & -0.31 & 0.04 & 0.05 & -0.29 & 0.0 \\
0.1 & 0.11 & 0.26 & -0.01 & -0.03 & 0.35 & -0.03 & -0.04 & 0.33 & -0.49 & -0.53 & -0.18 & -0.07 & 0.33 & -0.09 \\
0.03 & 0.03 & 0.21 & -0.12 & 0.23 & -0.16 & 0.09 & -0.19 & 0.58 & -0.07 & 0.13 & -0.19 & 0.2 & -0.59 & -0.17 \\
0.25 & -0.24 & -0.05 & -0.48 & 0.13 & 0.26 & 0.09 & -0.09 & -0.16 & -0.43 & 0.5 & 0.17 & 0.09 & 0.14 & -0.17 \\
0.03 & 0.01 & 0.25 & -0.09 & -0.25 & 0.46 & -0.26 & 0.06 & -0.29 & -0.04 & -0.05 & -0.08 & 0.26 & -0.47 & 0.45 \\
0.29 & -0.2 & 0.34 & 0.27 & -0.21 & -0.33 & -0.23 & 0.46 & -0.08 & -0.17 & 0.24 & -0.36 & 0.07 & 0.09 & -0.18 \\
0.76 & 0.2 & -0.23 & 0.15 & -0.04 & -0.15 & -0.23 & -0.21 & -0.05 & -0.05 & -0.21 & 0.32 & -0.01 & -0.19 & -0.06 \\
-0.25 & 0.1 & -0.06 & 0.1 & -0.22 & 0.28 & -0.2 & 0.31 & 0.08 & 0.05 & -0.01 & 0.41 & 0.19 & -0.15 & -0.64 \\
-0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26 & -0.26
\end{bmatrix}$$

We proceed by computing the low rank α approximation of the matrix M, by truncating the sum $\sum_{i=1}^{n} \lambda_i \mathbf{q}_i \mathbf{q}_i^T$ to the first $\lceil \alpha n \rceil$ terms, considering a parameter α of 0.75. The matrix obtained from this process, denoted as $\tilde{M}$, is:

$$\tilde{M} = \begin{bmatrix}
-0.16 & 0.81 & -0.03 & -0.11 & -0.08 & -0.03 & -0.02 & -0.05 & -0.07 & -0.04 & 0.02 & -0.16 & -0.11 & 0.02 & 0.02 \\
0.81 & -0.91 & -0.44 & -0.46 & -0.6 & 0.71 & 0.56 & 0.57 & 0.55 & 0.7 & -0.14 & -0.46 & -0.6 & -0.14 & -0.16 \\
-0.03 & -0.44 & -0.24 & 0.78 & 0.7 & 0.84 & -0.24 & -0.24 & -0.23 & -0.15 & -0.09 & -0.21 & -0.3 & -0.08 & -0.07 \\
-0.11 & -0.46 & 0.78 & -0.23 & 0.7 & -0.14 & -0.22 & -0.21 & -0.22 & -0.15 & -0.07 & 0.77 & -0.3 & -0.07 & -0.09 \\
-0.08 & -0.6 & 0.7 & 0.7 & -0.39 & -0.21 & 0.71 & -0.32 & -0.3 & -0.2 & -0.1 & 0.68 & -0.41 & -0.09 & -0.07 \\
-0.03 & 0.71 & 0.84 & -0.14 & -0.21 & -0.09 & -0.17 & -0.14 & -0.15 & -0.1 & -0.06 & -0.11 & -0.19 & -0.07 & -0.09 \\
-0.02 & 0.56 & -0.24 & -0.22 & 0.71 & -0.17 & -0.23 & 0.75 & -0.23 & -0.16 & -0.09 & -0.22 & -0.31 & -0.08 & -0.05 \\
-0.05 & 0.57 & -0.24 & -0.21 & -0.32 & -0.14 & 0.75 & -0.22 & -0.22 & -0.15 & -0.09 & -0.18 & 0.71 & -0.1 & -0.12 \\
-0.07 & 0.55 & -0.23 & -0.22 & -0.3 & -0.15 & -0.23 & -0.22 & -0.22 & -0.15 & -0.08 & -0.21 & 0.7 & 0.92 & -0.09 \\
-0.04 & 0.7 & -0.15 & -0.15 & -0.2 & -0.1 & -0.16 & -0.15 & -0.15 & -0.1 & 0.95 & -0.14 & -0.2 & -0.05 & -0.06 \\
0.02 & -0.14 & -0.09 & -0.07 & -0.1 & -0.06 & -0.09 & -0.09 & -0.08 & 0.95 & -0.04 & -0.06 & -0.1 & -0.03 & -0.03 \\
-0.16 & -0.46 & -0.21 & 0.77 & 0.68 & -0.11 & -0.22 & -0.18 & -0.21 & -0.14 & -0.06 & -0.22 & 0.72 & -0.07 & -0.13 \\
-0.11 & -0.6 & -0.3 & -0.3 & -0.41 & -0.19 & -0.31 & 0.71 & 0.7 & -0.2 & -0.1 & 0.72 & -0.39 & -0.11 & 0.87 \\
0.02 & -0.14 & -0.08 & -0.07 & -0.09 & -0.07 & -0.08 & -0.1 & 0.92 & -0.05 & -0.03 & -0.07 & -0.11 & -0.03 & -0.01 \\
0.02 & -0.16 & -0.07 & -0.09 & -0.07 & -0.09 & -0.05 & -0.12 & -0.09 & -0.06 & -0.03 & -0.13 & 0.87 & -0.01 & 0.06
\end{bmatrix}$$

Back transformation

Given that the modularity matrix M was selected for this process, it is necessary to perform a back transformation from the modularity matrix to the adjacency matrix, so that the results can be interpreted in the context of a network. The entries of the this quasi-adjacency matrix $\tilde{A}$, calculated using the formula $\tilde{a}_{u,v} = \tilde{m}_{u,v} + \frac{\delta_u \delta_v}{2m}$, are:

$$\tilde{A} = \begin{bmatrix}
-0.14 & 0.96 & 0.05 & -0.03 & 0.02 & 0.02 & 0.06 & 0.02 & 0.0 & 0.01 & 0.04 & -0.08 & -0.01 & 0.04 & 0.04 \\
0.96 & -0.01 & 0.01 & -0.01 & -0.0 & 1.01 & 1.01 & 1.02 & 1.0 & 1.0 & 0.01 & -0.01 & 0.0 & 0.01 & -0.01 \\
0.05 & 0.01 & -0.01 & 1.01 & 1.0 & 0.99 & -0.01 & -0.02 & -0.0 & -0.0 & -0.01 & 0.01 & -0.0 & -0.01 & 0.0 \\
-0.03 & -0.01 & 1.01 & -0.0 & 1.0 & 0.01 & 0.0 & 0.01 & 0.0 & 0.0 & 0.01 & 1.0 & 0.0 & 0.0 & -0.01 \\
0.02 & -0.0 & 1.0 & 1.0 & 0.01 & -0.01 & 1.01 & -0.02 & -0.0 & -0.0 & -0.0 & 0.98 & -0.01 & 0.01 & 0.03 \\
0.02 & 1.01 & 0.99 & 0.01 & -0.01 & 0.01 & -0.02 & 0.01 & 0.0 & 0.0 & -0.01 & 0.04 & 0.01 & -0.02 & -0.04 \\
0.06 & 1.01 & -0.01 & 0.0 & 1.01 & -0.02 & -0.01 & 0.97 & -0.01 & -0.01 & -0.01 & 0.0 & -0.01 & -0.0 & 0.02 \\
0.02 & 1.02 & -0.02 & 0.01 & -0.02 & 0.01 & 0.97 & 0.01 & 0.0 & 0.0 & -0.01 & 0.04 & 1.01 & -0.02 & -0.04 \\
0.0 & 1.0 & -0.0 & 0.0 & -0.0 & 0.0 & -0.01 & 0.0 & 0.0 & 0.0 & -0.0 & 0.01 & 1.0 & 1.0 & -0.01 \\
0.01 & 1.0 & -0.0 & 0.0 & -0.0 & 0.0 & -0.01 & 0.0 & 0.0 & -0.0 & 1.0 & 0.01 & 0.0 & -0.0 & -0.01 \\
0.04 & 0.01 & -0.01 & 0.01 & -0.0 & -0.01 & -0.01 & -0.01 & -0.0 & 1.0 & -0.01 & 0.02 & -0.0 & -0.01 & -0.0 \\
-0.08 & -0.01 & 0.01 & 1.0 & 0.98 & 0.04 & 0.0 & 0.04 & 0.01 & 0.01 & 0.02 & 0.01 & 1.02 & 0.0 & -0.05 \\
-0.01 & 0.0 & -0.0 & 0.0 & -0.01 & 0.01 & -0.01 & 1.01 & 1.0 & 0.0 & -0.0 & 1.02 & 0.01 & -0.01 & 0.97 \\
0.04 & 0.01 & -0.01 & 0.0 & 0.01 & -0.02 & -0.0 & -0.02 & 1.0 & -0.0 & -0.01 & 0.0 & -0.01 & -0.0 & 0.02 \\
0.04 & -0.01 & 0.0 & -0.01 & 0.03 & -0.04 & 0.02 & -0.04 & -0.01 & -0.01 & -0.0 & -0.05 & 0.97 & 0.02 & 0.09
\end{bmatrix}$$

Normalization

The normalization of matrix $\tilde{A}$ is carried out next, ensuring that its entries fall within the range $[0, 1]$. We choose to perform the truncation function. The matrix $\ddot{A}$ that we have obtained through this process is:

$$\ddot{A} = \begin{bmatrix}
0.0 & 0.96 & 0.05 & 0.0 & 0.02 & 0.02 & 0.06 & 0.02 & 0.0 & 0.01 & 0.04 & 0.0 & 0.0 & 0.04 & 0.04 \\
0.96 & 0.0 & 0.01 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.01 & 0.0 & 0.0 & 0.01 & 0.0 \\
0.05 & 0.01 & 0.0 & 1.0 & 1.0 & 0.99 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.01 & 0.0 & 0.01 & 0.0 & 0.0 & 0.01 & 1.0 & 0.0 & 0.0 & 0.0 \\
0.02 & 0.0 & 1.0 & 1.0 & 0.01 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.98 & 0.0 & 0.01 & 0.03 \\
0.02 & 1.0 & 0.99 & 0.01 & 0.0 & 0.01 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.04 & 0.01 & 0.0 & 0.0 \\
0.06 & 1.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.97 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.02 \\
0.02 & 1.0 & 0.0 & 0.01 & 0.0 & 0.01 & 0.97 & 0.01 & 0.0 & 0.0 & 0.0 & 0.04 & 1.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.01 & 1.0 & 1.0 & 0.0 \\
0.01 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.01 & 0.0 & 0.0 & 0.0 \\
0.04 & 0.01 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.02 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.01 & 1.0 & 0.98 & 0.04 & 0.0 & 0.04 & 0.01 & 0.01 & 0.02 & 0.01 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.01 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 1.0 & 0.01 & 0.0 & 0.97 \\
0.04 & 0.01 & 0.0 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.02 \\
0.04 & 0.0 & 0.0 & 0.0 & 0.03 & 0.0 & 0.02 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.97 & 0.02 & 0.09
\end{bmatrix}$$

Adjacency matrix generation

The adjacency matrix A' is then generated from the normalized matrix $\ddot{A}$, which is treated as a matrix of probabilities for Bernoulli sampling. As explained in 4.2.2, adjustments include setting $a'i, j = 0; \forall i = j$ to remove self-loops, and enforcing $a'i, j = a'_{j,i}; \forall j > i$ to reflect undirected connections. The adjacency matrix A', showcased below, emerges from a single iteration of the process. It's important to note that due to the random nature of the Bernoulli sampling involved, this matrix represents just one of the many potential outcomes that could result from the procedure:

$$A' = \begin{bmatrix}
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 0.0 \\
0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 1.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 1.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0
\end{bmatrix}$$

Finally, we can draw the graph corresponding to the adjacency matrix A', which is displayed in the figure below:

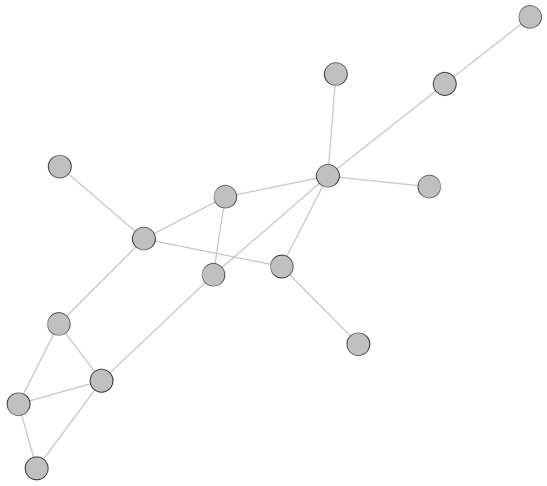

Figure 4.2: Generated Florentine Families Graph with $\alpha = 0.75$

Labels are not displayed on the graph as the process has anonymized it. However, it's worth noting that by looking at the graph we could potentially identify some family names. This is because we set the parameter α to 0.75, which means that the algorithm retains a significant amount of information regarding the community structure.

4.2.4 Application to a large-sized network

We now consider a larger network representing Facebook friendships connections among users, containing 3996 nodes and 88177 edges. The dataset, originally crafted by (McAuley & Leskovec., 2012), was sourced from the Stanford Large Network Dataset Collection (Leskovec & Krevl, 2014). Exhibiting a well-defined community structure with distinctly separate clusters, the network is illustrated in the figure below:

Figure 4.3: Facebook Graph

Following the same methodology outlined for the preceding example, we applied the Spectral Graph Forge method to this larger network, setting however an higher α parameter of 0.85. Due to the network's extensive size, it was impractical to visually present the matrices resulting from the various stages. The resulting graph, featuring 92359 nodes, is shown below. Notably, the principal clusters in the input network remain discernible in the generated one.

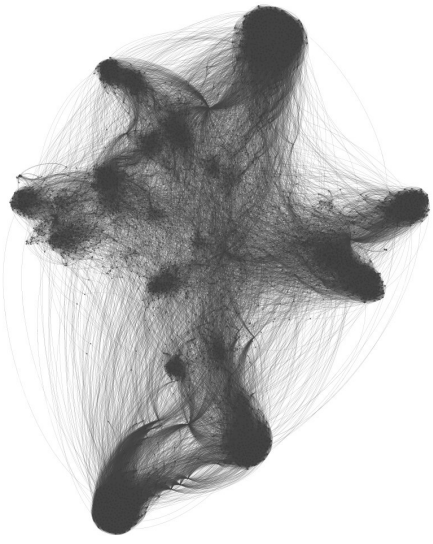

Figure 4.4: Generated Facebook Graph with $\alpha = 0.85$

4.3 Extension of the Spectral Graph Forge method

4.3.1 Introduction of new transformations

As seen in the previous section, the Spectral Graph Forge Method generated synthetic networks from an input network following steps:

1. Consider a graph G as input and compute its adjacency matrix A.
2. Compute a matrix M that is a real and symmetric transformation of the adjacency matrix A.
3. Perform a low rank α approximation on M, generating a matrix $\tilde{M}$.
4. Back transform $\tilde{M}$ to a quasi-adjacency matrix $\tilde{A}$.
5. Normalize and sample from $\tilde{A}$ to generate an effective adjacency matrix A'.

The transformation of the adjacency matrix that is mainly considered by (Baldesi et al., 2018) is the modularity matrix, however, any real and symmetric transformation of the adjacency matrix is in theory suitable for the method. In this section, we explore various matrices, employ them as transformations within the Spectral Graph Forge method and assess their influence on the resulting synthetic networks.

The considered matrices are several, starting from the adjacency matrix itself, obtained through identity transformation, and the modularity matrix outlined in Subsection 3.4.8. We also considered the Laplacian matrix and the signless Laplacian matrix, both discussed in Subsection 3.4.5. Moreover, we employed the General Zagreb matrix, defined in Subsection 3.4.7, the Transition matrix of a Random Walk, which is detailed in Subsection 3.4.9, and the Bethe-Hessian matrix, which is presented and analysed in Subsection 3.4.11. Furthermore, we have introduced a new matrix, which was developed and tuned empirically. This matrix, that we will denote as Factor-Layered-Community-Matrix (FLCM), is defined as $F = A - \eta\frac{\delta_u\delta_v}{2s} + \psi I$, where η and ψ are parameters, δ_u, δ_v are the degrees of nodes u, v, and $s = \sum_{i=1}^{n} k_i^2$ is a normalization factor. For the following analysis, we tuned both η and ψ at $\frac{1}{3}$

A summary on the matrices just discussed, their derivation and their back transformations to the adjacency matrix (required to reconstruct the quasi-adjacency matrix during the "Back Transformation" phase of the Spectral Graph Forge method) are detailed in the Table below:

Matrix	**Transformation**	**Back-transformation**
adjacency	A	A
modularity	$B = A - \frac{\delta_u \delta_v}{2m}$	$\tilde{A} = B + \frac{\delta_u \delta_v}{2m}$
Laplacian	$L = D - A$	$\tilde{A} = D - L$
signless Laplacian	$L_{\text{signless}} = D + A$	$\tilde{A} = L_{\text{signless}} - D$
general Zagreb	$G_Z = D^3 + A$	$\tilde{A} = G_Z - D^3$
sum connectivity	$S_C = \begin{cases} \frac{1}{\sqrt{\delta_u \delta_v}} & \text{if } i \sim j \\ 0 & \text{otherwise} \end{cases}$	$\tilde{A} = \begin{cases} S_{C_{u,v}} \sqrt{\delta_u \delta_v} & \text{if } a_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$
transition Random Walk	$P = D^{-1} A$	$\tilde{A} = DP$
Bethe Hessian	$Z(r) = (r^2 - 1)I - rA + D$	$\tilde{A} = \frac{1}{3}((r^2 - 1) * I - Z(r) + D)$
FLCM	$F = A - \eta \frac{\delta_u \delta_v}{2s} + \psi I$	$\tilde{A} = F + \eta \frac{\delta_u \delta_v}{2s} - \psi I$

Table 4.1: transformations

where A is the adjacency matrix, D is the degree matrix, δ_u is the degree of node u, m is the number of edges in the network, and $s = \sum_{i=1}^{n} k_i^2$.

4.3.2 Visual comparison of the transformations

To provide a first graphical visualization of the impact of the different transformations, we consider again the Facebook graph used in Subsection 4.2.4:

Figure 4.5: Facebook Graph

Below, are reported the synthetic versions of this graph generated by applying the different transformations, each with an alpha value of 0.85.

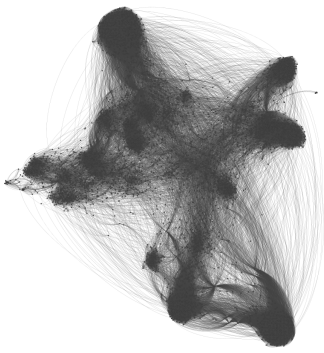

Figure 4.6: identity

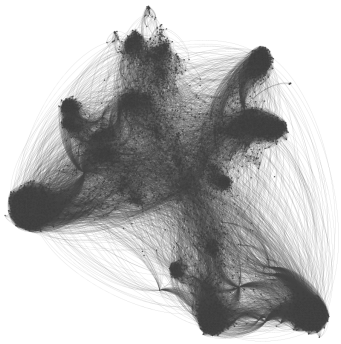

Figure 4.7: modularity

Figure 4.8: Laplacian

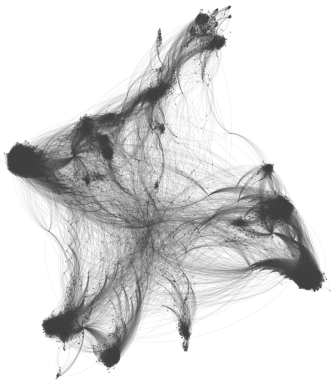

Figure 4.9: signless Laplacian

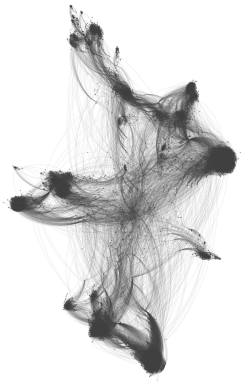

Figure 4.10: general Zagreb

Figure 4.11: sum connectivity

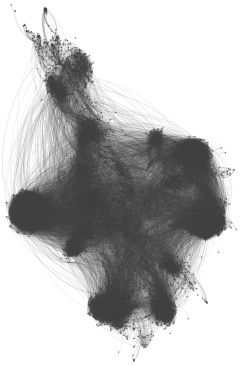

Figure 4.12: transition rw

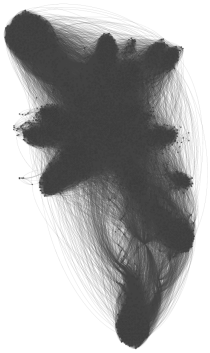

Figure 4.13: Bethe Hessian

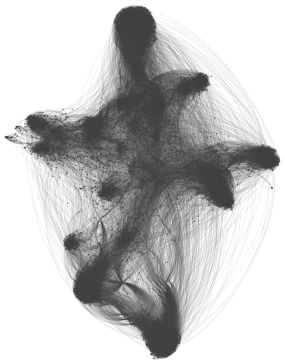

Figure 4.14: FLCM

We observe some differences among the graphs above in term of density. The graphs generated under the laplacian, the signless Laplacian, the general Zagreb and the sum connectivity transformations appear to have a sparser community structure with respect to the others, which seems to almost resemble the one of the input network. The graphs generated under the identity, modularity, and FLCM transformations appear to have a slightly denser community structure. Instead, the graphs generated under the transition rw, and the Bethe Hessian transformation present vey high density, with no complete separation among the still distinguishable communities.

4.3.3 Performance evaluation

To assess the performance of the various transformations when employed in the Spectral Graph Forge algorithm, we conducted an analysis based on 2000 simulations using graphs generated by the algorithms Lancichinetti-Fortunato-Radicchi benchmark (Lancichinetti et al., 2008), Stochastic Block Model (explained in Subsection 3.3.4), Windmill (Estrada, 2015), and Gaussian Random Partition (Brandes et al., 2003).

Since the primary goal of the Spectral Graph Forge method is to generate synthetic networks that

replicate the community structure of the input network, we started evaluating the performance of the different transformations by using two metrics: the ratio of the average clustering coefficient of the generated network to that of the input network (to which from now on we will refer to as "Avg Clustering Ratio"), and the ratio of the number of communities found by modularity maximization in the generated network to those in the input network (to which from now on we will refer to as "Modularity Communities Ratio"). A good transformation should yield a generated network with a similar community structure with respect to the input network, thus, the closer these ratios are to 1, the better the transformation.

However, we do not limit to evaluate the transformations in terms of community structure, but we analyse also their ability to replicate in the generated networks the connectivity, the assortativity, and the energy of the input graph. The connectivity is measured by the ratio of the beta index of the synthetic network to that of the input network (to which from now on we will refer to as "Beta Index Ratio") and the the ratio of the sum connectivity index of the synthetic network to that of the input network (to which from now on we will refer to as "Sum Connectivity Index Ratio"). The assortativity is measured by the ratio of the degree assortativity coefficient of the synthetic network to that of the input network (to which from now on we will refer to as "Degree Assortativity Coefficient Ratio"). The energy is measured by the ratio of the forgotten Zagreb index of the synthetic network to that of the input network (to which from now on we will refer to as "Forgotten Zagreb Index Ratio"). As mentioned before, the closer these ratios are to 1, the better the transformation.

The results of this analysis are presented below, reported for alpha values of $0.3, 0.6$ and 0.9. The boxplots are computed upon all the data collected from the analysed graphs as the measures are comparable across them.

Analysis of the average clustering ratio measure

Below are reported the boxplots showing the ratio of the average clustering coefficient of the generated network to that of the input network, achieved by the different transformations for various alpha values.

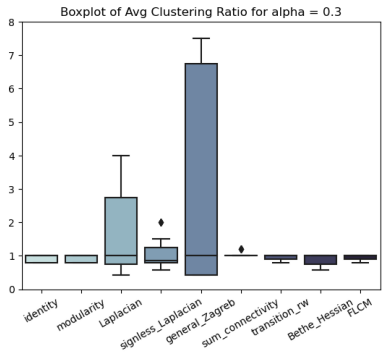

Figure 4.15: $\alpha = 0.3$

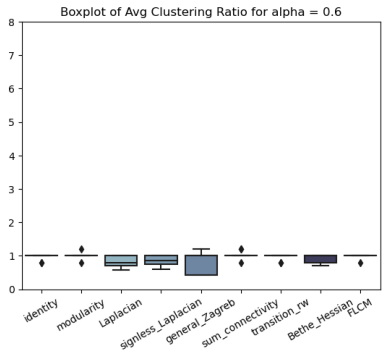

Figure 4.16: $\alpha = 0.6$

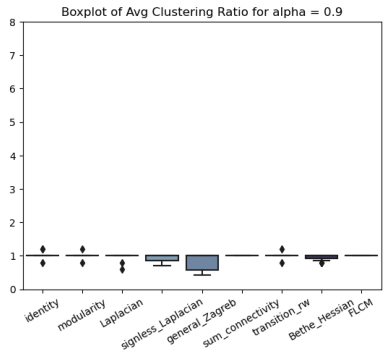

Figure 4.17: $\alpha = 0.9$

Observing the three plots, it is possible to notice that the majority of the transformations are able to produce synthetic graphs with a close average clustering to the one of the input networks, even when retaining a small quantity of eigenvectors. However, the Laplacian, the signless Laplacian and the general Zagreb transformations show significantly higher variability, especially for the alpha value of 0.3, suggesting that these transformations do not preserve the average clustering well at lower alpha values. Nevertheless, as the alpha value increases, also these transformations demonstrate significantly reduced variability.

Analysis of the modularity ratio measure

Below are reported the boxplots showing the ratio of the number of communities found by modularity maximization in the generated network to those in the input network, achieved by the different transformations for various alpha values.

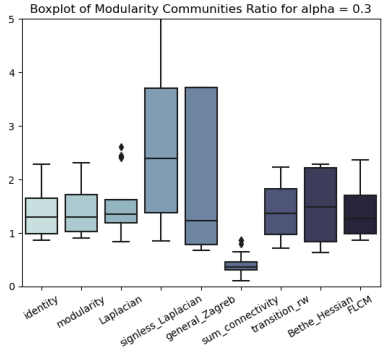

Figure 4.18: $\alpha = 0.3$

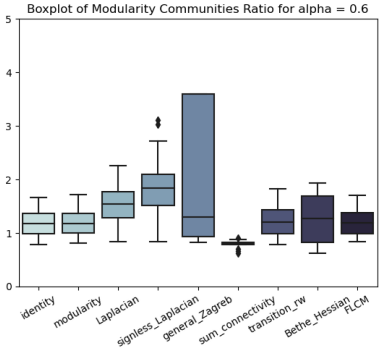

Figure 4.19: $\alpha = 0.6$

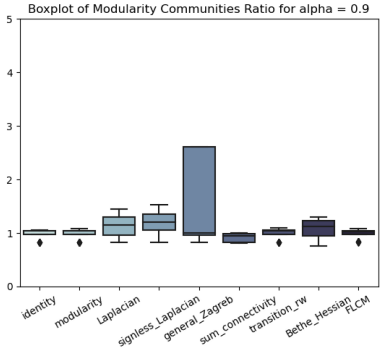

Figure 4.20: $\alpha = 0.9$

As mentioned, the boxplots illustrate the ratio of the number of communities identified by modularity maximization in the generated network to those in the input network. For the alpha value of 0.3 there is considerable variability, especially in the signless Laplacian and the general Zagreb transformations. At alpha of 0.6 the variability decreases slightly in all the transformations. When alpha reaches 0.9, the ratio become more consistent and closer to 1 across most transformations, excluding the general Zagreb, indicating that the most of the synthetic networks resemble the input network in terms of community structure. Among the transformations, the identity, the modularity, the sum connectivity, the transition rw and the FLCM consistently show lower variability and a ratio closer to 1, suggesting that these transformations are better at retaining the community structure of the input network with respect to the others.

Analysis of the beta index ratio measure

Below are reported the boxplots showing the ratio of the beta index of the synthetic network to that of the input network, achieved by the different transformations for various alpha values.

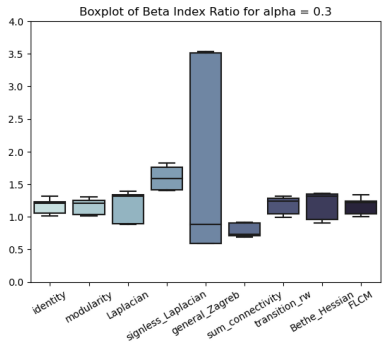

Figure 4.21: $\alpha = 0.3$

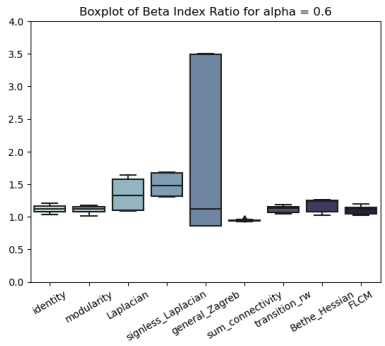

Figure 4.22: $\alpha = 0.6$

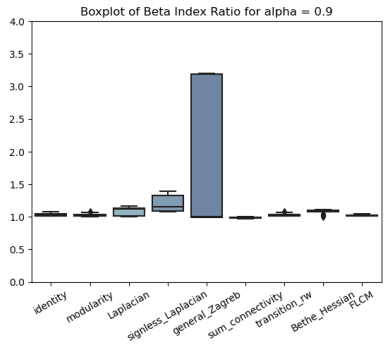

Figure 4.23: $\alpha = 0.9$

For all the alpha values, most transformations maintain a beta index ratio close to 1, indicating a similar level of connectivity between the synthetic and the input network. However, the general Zagreb transformation shows a significantly higher ratio, which remains noticeable also as the alpha value increases at 0.9. This suggests that while most transformations preserve the connectivity structure of the input network across different alpha values, the general Zagreb transformation consistently results in a synthetic network with higher beta index.

Analysis of the sum connectivity index ratio measure

Below are reported the boxplots showing the ratio of the sum connectivity index of the synthetic network to that of the input network, achieved by the different transformations for various alpha values.

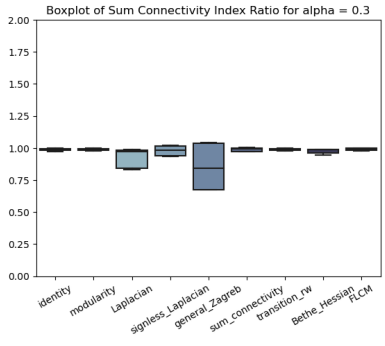

Figure 4.24: $\alpha = 0.3$

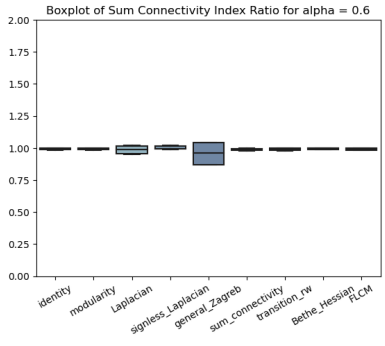

Figure 4.25: $\alpha = 0.6$

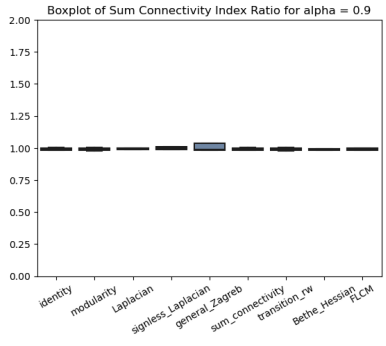

Figure 4.26: $\alpha = 0.9$

Across all the alpha coefficients the ratio remain close to 1 for most transformations, although the Laplacian and the general Zagreb transformations show slight deviations. This indicates that the sum connectivity index is mostrly preserved in the synthetic network independently of the transformation or the quantity of eigenvectors retained.

Analysis of the degree assortativity coefficient ratio measure

Below are reported the boxplots showing the ratio of the degree assortativity coefficient of the synthetic network to that of the input network, acieved by the different transformations for various alpha values.

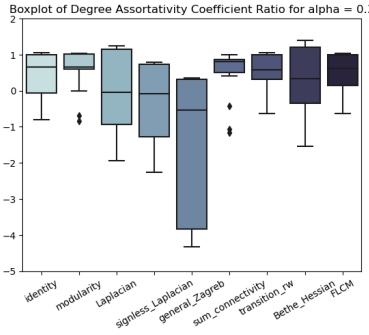

Figure 4.27: $\alpha = 0.3$

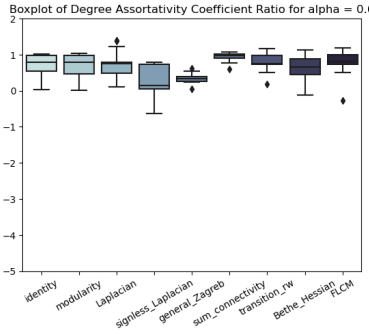

Figure 4.28: $\alpha = 0.6$

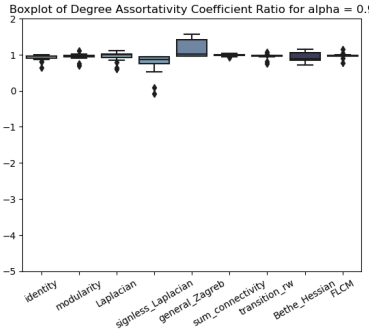

Figure 4.29: $\alpha = 0.9$

Observing the first graph, displaying results for alpha value of 0.3, we notice that the ratio show significant variability across transformations, with some like the general Zagreb displaying larger deviations. Moving to the second one, dysplaying results for alpha value of 0.6, we observe that most transformations exhibit a ratio closer to 1, though some variability remains. For the third graph, displaying results for alpha value of 0.9, the ratio for most transformations are around 1, indicating a high level of preservation of the degree assortativity coefficient. The general Zagreb transformation, while still showing some deviation, has also improved significantly. We can conclude that higher alpha values generally lead to better retention of the input network's degree assortativity characteristics, with most transformations performing well at this level.

Analysis of the forgotten Zagreb index ratio measure

Below are reported the boxplots showing the ratio of the forgotten Zagreb index of the synthetic network to that of the input network, achieved by the different transformations for various alpha values.

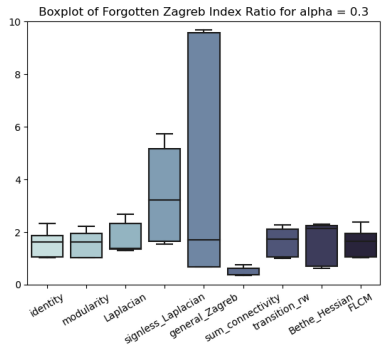

Figure 4.30: $\alpha = 0.3$

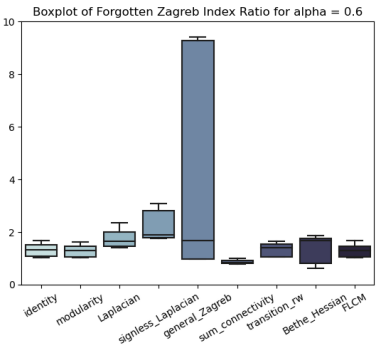

Figure 4.31: $\alpha = 0.6$

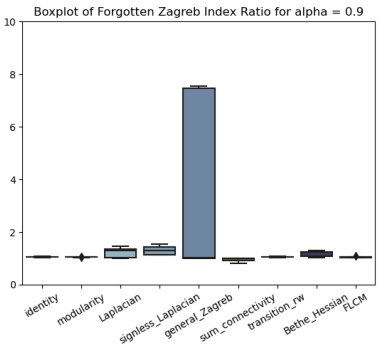

Figure 4.32: $\alpha = 0.9$

At alpha values of 0.3 and 0.6, there is considerable variability, particularly with the signless Laplacian and general Zagreb transformations. The ratio become more consistent for most transformations, at alpha value of 0.9. However, the general Zagreb transformation still exhibits significant deviations.

Summarizing our findings, we conclude that the identity, modularity, sum connectivity, transition rw, and FLCM transformations are effective in preserving community structure, as evidenced by the average clustering ratio and modularity ratio measures in the synthetic network. In terms of connectivity, measured through the beta index, the sum connectivity transformation demonstrates the best performance, with minimal differences observed across transformations when considering the sum connectivity index. Regarding degree assortativity and graph energy, reflected by the forgotten Zagreb index, the sum connectivity transformation again proves to be the most effective.

While assessing the performance of the transformations in terms of community structure, connectivity, assortativity and energy similarity between the generated graph and the input graph is fundamental, it is not sufficient on its own. In fact, if we only evaluated how similar the generated network is to the input one, we would always prefer transformations that create networks identical to the input one, which would not fulfil our objective, which is to generate a network that resembles the community structure of the input one while providing anonymity to the original nodes. Therefore, we want the synthetic network to be sufficiently distinct from the input one. Hence, we also evaluate the transformations based on the distance between the generated network and the input network. This distance is measured as the edit distance between their adjacency matrices, that is defined as the minimum cost of an edit path, which is a sequence of node and edge substitutions, deletions, and insertions, required to transform the first graph into a graph that is isomorphic to the second graph. The edit distance is scaled between 0 and 1, where 0 indicates identical graphs and 1 indicates maximum dissimilarity.
The figures below present the results of this analysis. Each figure corresponds to a different transformation, with the x-axis representing the alpha values and the y-axis representing the edit distance

between the generated network and the input network. The plotted line represents the average of the data collected from the analysed graphs.

Results for the identity transformation

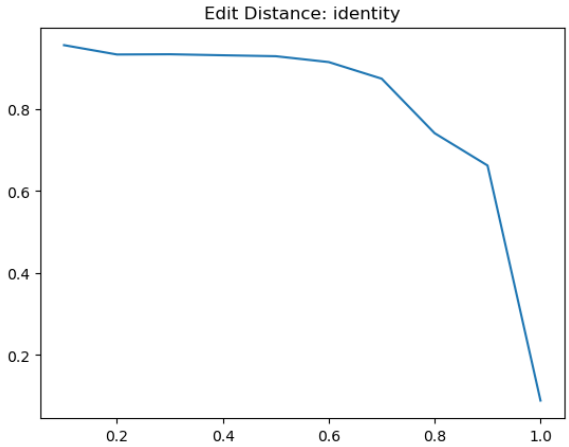

Figure 4.33: Edit distance for the identity transformation

Analyzing the above graph, it is evident that the edit distance between the generated network and the input network decreases as the alpha value increases. This occurs because, with higher alpha values, the generated network retains more information from the input network, making them more similar. However, this relationship is not linear. Up to an alpha value of approximately 0.7, the edit distance does not decrease significantly. Beyond this point, the decrease becomes more pronounced, with a substantial drop observed for alpha values greater than 0.9.

Results for the modularity transformation

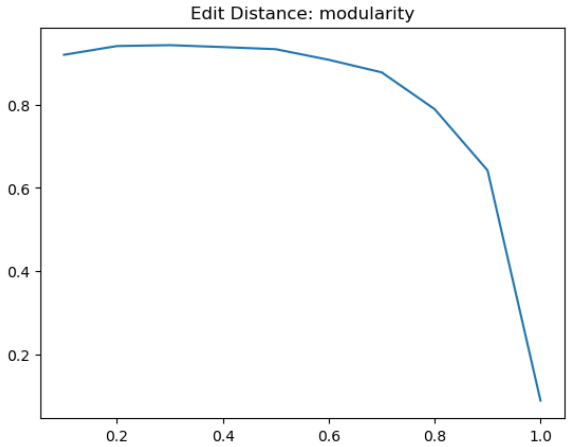

Figure 4.34: Edit distance for the modularity transformation

For what concernes the modularity transformation, there are not many differences compared to the previous analysis. A significant drop in the Edit distance occurs with alpha values larger than 0.7, with the most notable decrease observed at an alpha value of around 0.9. At an alpha value of 1, the Edit distance is at its lowest, indicating that the generated network closely resembles the input network.

Results for the Laplacian transformation

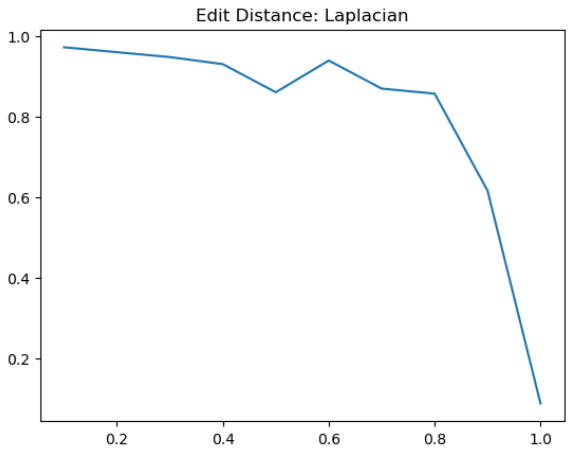

Figure 4.35: Edit distance for the Laplacian transformation

Observing the graph concering the Laplacian transformation we can identify a general trend of decreasing edit distance as the alpha value increases, despite some minor fluctuations between alpha values of 0.4 and 0.6. It is worth noting that the distance remain relatively high until the alpha value reaches 0.8.

Results for the signless Laplacian transformation

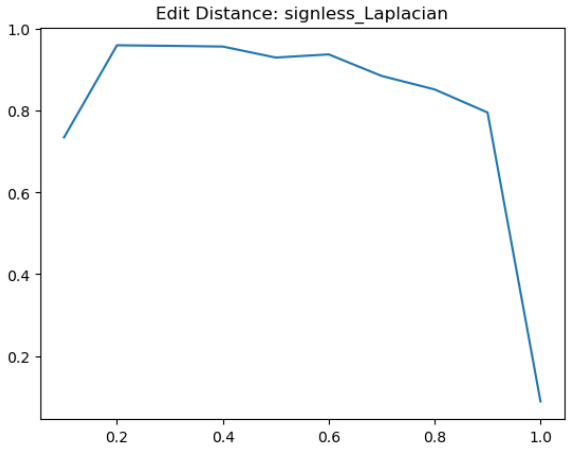

Figure 4.36: Edit distance for the signless Laplacian transformation

Analysing in the figure above the relation among the edit distance and the alpha values for the signless Laplacian transformation, we observe that the Edit distance increases slightly at the beginning until the alpha value is 0.2, then it remains relatively high and stable up to around 0.6. From this point onwards, the edit distance begins to decrease gradually, with a very pronounced drop observed after an alpha value of approximately 0.8.

Results for the general Zagreb transformation

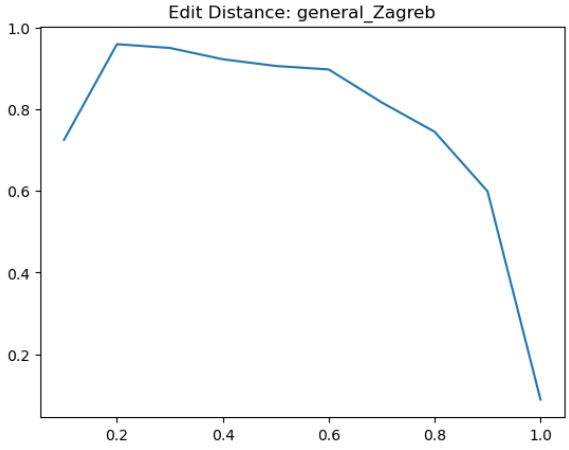

Figure 4.37: Edit distance for the general Zagreb transformation

For the General Zagreb transformation, we notice that initially the Edit Distance increases, then remains high and almost stable for alpha values ranging from 0.2 to around 0.6. From this point onwards, a gradual decrease is observed.

Results for the sum connectivity transformation

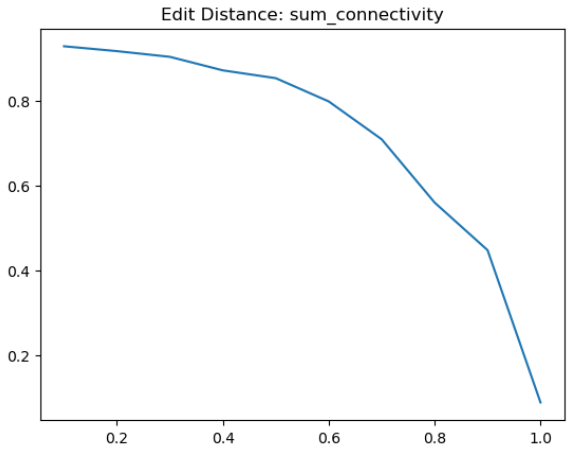

Figure 4.38: Edit distance for the sum connectivity transformation

By observing the above graph, we notice that the edit distance shows a consistent decreasing trend across the entire range of alpha values. Starting high at lower alpha values, it gradually declines as alpha increases, with more significant decreases occurring after an alpha value of approximately 0.5. Unlike other transformations, which exhibit a significant distance between the synthetic and input graphs until an high alpha value cause a sharp drop in the edit distance, the sum connectivity transformation appears to have a more continuous and decrease.

Results for the transition rw transformation

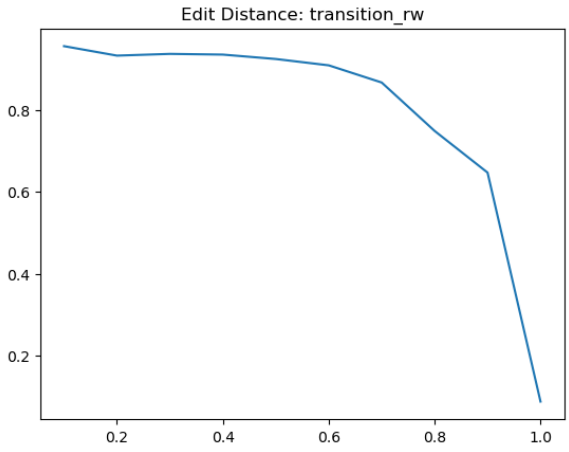

Figure 4.39: Edit distance for the transition rw transformation

The above graph shows a high and stable edit distance at lower alpha values, with a gradual decrease beginning around 0.3. The decline becomes more pronounced after an alpha value of 0.7, leading to a significant drop as alpha approaches 1.0.

Results for the Bethe Hessian transformation

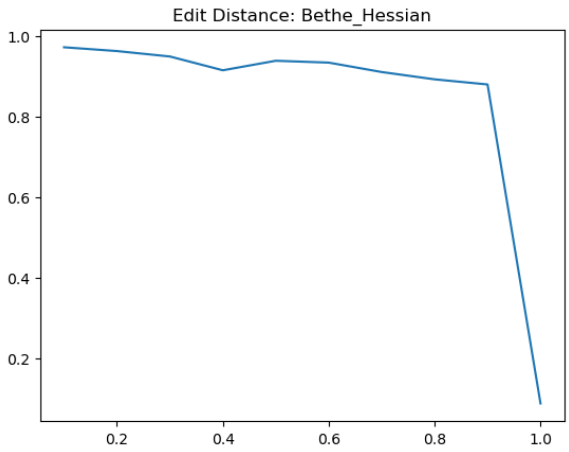

Figure 4.40: Edit distance for the Bethe Hessian transformation

Observing the graph related to the Bethe Hessian transformation, it is possible to notice a high edit distance, between the synthetic graph and the input graph, for alpha values up to 0.9, after which a sharp decrease is observed.

Results for the FLCM transformation

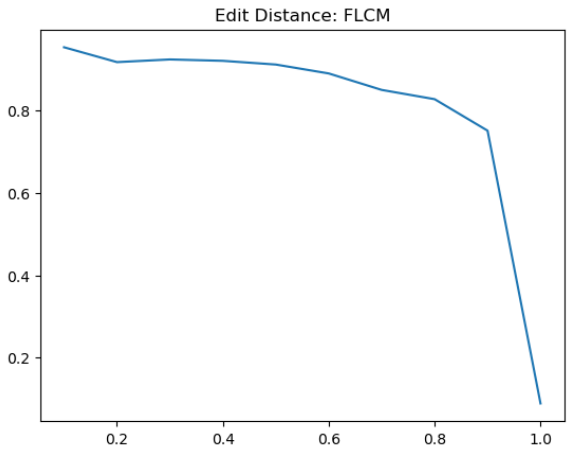

Figure 4.41: Edit distance for the FLCM transformation

Similarly to the previous graph of the Bethe Hessian, in the case of the FLCM transformation, the edit distance remains mostly high and stable for alpha values up to 0.9, after which point, a sharp decrease is observed.

From this analysis, it is possible to conclude that, in general, for all transformations, the generated network remains significantly different from the input network as long as the alpha value is less

than approximately 0.8.

Finally, we aim to further evaluate the transformations that performed best in terms of community structure, in order to identify the optimal transformation. To do so, we introduce a new metric that combines the similarity of the community structure and the distance between the generated network and the input network. To achieve this, we developed a metric c that considers the average clustering ratio, the modularity ratio, and the edit distance between the two networks. This measure is calculated as follows:

$$c = \sqrt{(1 - \text{modularity\_ratio})^2 + (1 - \text{average\_clustering\_ratio})^2 + \text{edit\_distance}^2}$$

This formula represents the Euclidean distance from the point $(1, 1, 1)$ in the 3-dimensional space defined by the three metrics. An optimal situation is achieved by minimizing this metric.
Below, we present the results of this analysis for the identity, modularity, sum connectivity, transition rw, and FLCM transformations, which were identified as the best transformations in terms of community structure. The x-axis represents the alpha values, while the y-axis represents the value of the metric c. The plotted line represents the average of the data collected from the analysed graphs. We expect the curve to be convex shaped, as while the community structure similarity increases with the alpha value, the distance between the generated and input networks decreases. We will therefore look for the alpha value that minimizes the metric c for each transformation.

Results for the identity transformation

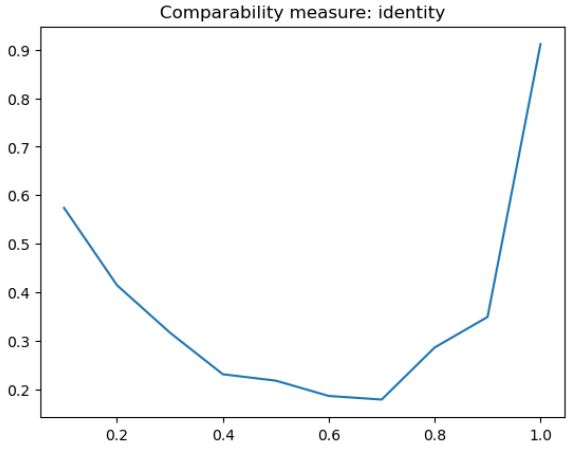

Figure 4.42: Comparability measure for the identity transformation

The graph above shows that the optimal alpha value for the identity transformation is 0.7. However, the transformation maintains low values of the metric c for all the alpha values in the range $[0.4, 0.7]$.

Results for the modularity transformation

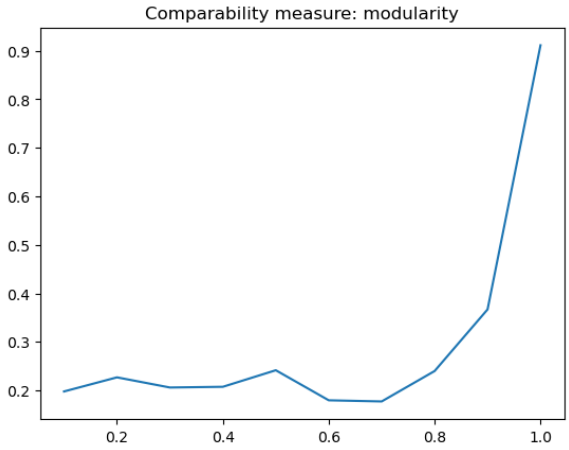

Figure 4.43: Comparability measure for the modularity transformation

The graph above shows that the curve for the modularity transformation is not perfectly convex. The transformatioon maintains low values of the metric c for all the alpha values smaller than 0.7, with the optimal alpha values being 0.2 and 0.7.

Results for the sum connectivity transformation

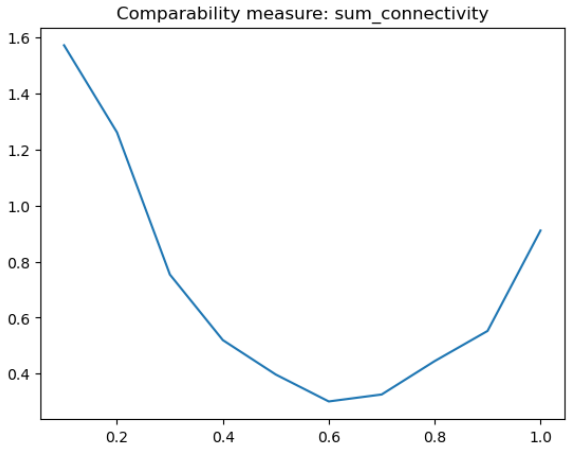

Figure 4.44: Comparability measure for the sum connectivity transformation

The graph above shows that the optimal alpha value for the sum connectivity transformation is 0.6.

Results for the transition rw transformation

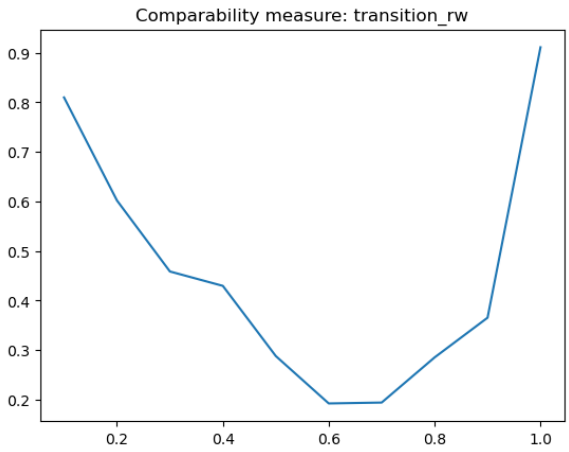

Figure 4.45: Comparability measure for the transition rw transformation

The graph above shows that the optimal alpha values for the transition rw transformation are 0.6 and 0.7.

Results for the FLCM transformation

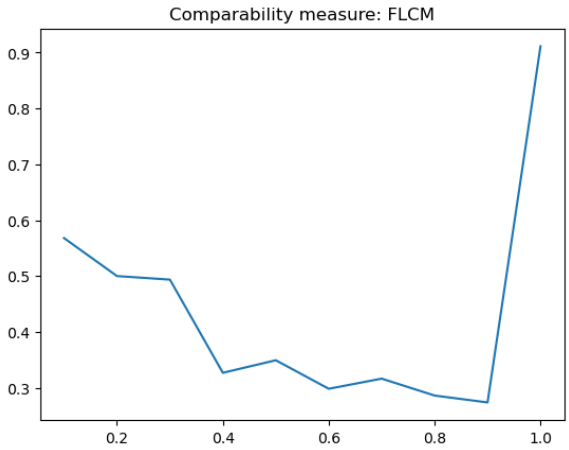

Figure 4.46: Comparability measure for the FLCM transformation

The graph above shows that the optimal alpha value for the FLCM transformation is 0.9. It is worth mentioning that this transformation maintain low values of the metric c for all the alpha values in the range $[0.4, 0.9]$.

Comparing the transformations through this metric, we can conclude that all of them are able to achieve a c value very close to 0 for at least one specific alpha value. We can also observe that the ones that can achieve a low c value for a wider range of alpha values are modularity and the FLCM transformations.

Chapter 5

Conclusions

This work began by laying the groundwork in Chapter 2 with an in-depth study of linear algebra focusing on the spectra of matrices. The derivation, properties, interpretation, and applications of eigenvectors and eigenvalues were studied, along with their role in diagonalization. Chapter 3, built on this foundation by analyzing the spectral properties of matrices associated with graphs, covering the basics of network theory, standard methods for network generation, and the global properties of graphs.

Building on this theoretical groundwork, Chapter 4 introduced the Spectral Graph Forge (SGF) method from a mathematical perspective, also providing a coding implementation. The method was first analysed using the modularity matrix as a transformation, which proved effective in preserving the community structure of the input network. The method was then extended by including new transformations, namely the identity, Laplacian, signless Laplacian, general Zagreb, sum connectivity, transition random walk, Bethe-Hessian, and FLCM transformations. Their performance was evaluated based on their ability to generate synthetic networks that closely resemble the input network in terms of community structure, connectivity, assortativity, and energy. For community structure, the identity, modularity, sum connectivity, and FLCM transformations appeared to be the bests, while for the metrics of connectivity, assortativity, and energy, the sum connectivity transformation appeared to be the most effective.

Bibliography

Axler, S. (1995). Linear algebra done right.

Baldesi, L., Markopoulou, A., & Butts, C. T. (2018). Spectral graph forge: Graph generation targeting modularity.

Brandes, U., Gaertler, M., & Wagner, D. (2003). Experiments on graph clustering algorithms.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Chandrashekhar, S. S., Srivastava, M., Jaganathan, B., & Shukla, P. (2022). Pagerank algorithm using eigenvector centrality - new approach. *CoRR*, *abs/2201.05469*.

Cvetković, D. M. (2009). Applications of graph spectra: An introduction to the literature.

Cvetković, D. M., Doob, M., & Sachs, H. (1980). Spectra of graphs: Theory and application.

Cvetković, D. M., & Gutman, I. (2009). Applications of graph spectra.

Estrada, E. (2015). When local and global clustering of networks diverge.

Frobenius, G. (1912). Uber matrizen aus nicht negativen elementen.

Gerschgorin, S. (1931). Über die abgrenzung der eigenwerte einer matrix. *Izv. Akad. Nauk. USSR Otd. Fiz.-Mat. Nauk*, *6*, 749–754.

Glover, C. (2021). The non-backtracking spectrum of a graph and non-bactracking pagerank.

Hashimoto, K. (1989). Zeta functions of finite graphs and representations of p-adic groups. *Advances in Studies in Pure Mathematics*, *15*, 211–280.

Horoldagva, B., & Das, K. C. (2023). On zagreb indices of graphs.

Krzakalaa, F., Moorec, C., Mosseld, E., Neemand, J., Slyd, A., Zdeborováe, L., & Zhang, P. (2013). Spectral redemption in clustering sparse networks. *PNAS*.

Lancichinetti, A., Fortunato, S., & Radicchi, F. (2008). Benchmark graphs for testing community detection algorithms. *Physical Review E*, *78*(4).

Leskovec, J., & Krevl, A. (2014). Snap datasets: Stanford large network dataset collection.

McAuley, J., & Leskovec., J. (2012). Learning to discover social circles in ego networks.

Moler, C. (2002). The world's largest matrix computation: Google's pagerank is an eigenvector of a matrix of order 2.7 billion. *Matlab News and Notes*.

Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*(23), 8577–8582.

Nicholson, W. K. (2019). Nicholson-openlawa-2019a.

Penschuck, M., Brandes, U., Hamann, M., Lamm, S., Meyer, U., Safro, I., Sanders, P., & Schulz, C. (2020). Recent advances in scalable network generation.

Perron, O. (1907). Zur theorie der matrizen.

Saade, A., Krzakala, F., & Zdeborova, L. (2014a). Spectral clustering of graphs with the bethe hessian.

Saade, A., Krzakala, F., & Zdeborova, L. (2014b). Spectral density of the non-backtracking operator.

Simon, C. P., & Blume, L. (1994). Mathematics for economists.

Solomon, J. (2015). Numerical algorithms.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of "small-world" networks.

Wilf, H. S. (2002). Searching the web with eigenvectors. *UMAP Journal*, *23*(2).

Appendix A

Further concepts on matrix algebra

Determinant of matrices

Definition 54 (Determinant of a matrix)**.** Let $A \in \mathbb{M}_{n,n}$. Then the determinant of A, denoted as $\det(A)$, is a real valued function mapping A to a scalar value.

For a matrix $A \in \mathbb{M}_{n,n}$, the determinant is given by:

$$\det(A) = \det \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

Generalizing the rule for a matrix $A \in \mathbb{M}_{n,n}$. Then, the determinant of A can be computed by recursive approach, assuming that the determinant of a sub matrix A_{ij} is known, where $A_{i,j} \in \mathbb{M}_{(n-1),(n-1)}$, represents the matrix obtained by excluding the i^{th} row and the j^{th} column from A. Hence, $\det(A)$ is defined as:

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(A_{ij})$$

Invertible matrices

Definition 55 (Invertible matrix)**.** Let $A \in \mathbb{M}_{n,n}$. Then, it is invertible when there exists a matrix $B \in \mathbb{M}_{n,n}$, referred to as the inverse of A such that:

$$AB = BA = I$$

where I is the identity matrix.

The inverse of a matrix A is unique and is denoted as A^{-1}

Proposition 13. *Let $A \in \mathbb{M}_{n,n}$. Then, it has an inverse if and only if $det(A) \neq 0$. In such case,* $\det(A^{-1}) = \frac{1}{\det(A)}$

Proposition 14. *Let $A \in \mathbb{M}_{n,n}$ be invertible. Then, also A^T is invertible, and its inverse is $(A^{-1})^T$.*

Proof.

$$A^T(A^{-1})^T = (A^{-1}A)^T = I^T = I$$
$$(A^{-1})^T A^T = (AA^{-1})^T = I^T = I$$

□

Proposition 15. *Let $A \in \mathbb{M}_{n,n}$. Then,* $\det(A^T) = \det(A)$

Proposition 16. *Let $A, B \in \mathbb{M}_{n,n}$ be invertible. Then, also AB is invertible, and its inverse $(AB)^{-1}$ equals $B^{-1}A^{-1}$.*

Proof.

$$AB(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AA^{-1} = I$$
$$(B^{-1}A^{-1})AB = B^{-1}(A^{-1}A)B = B^{-1}B = I$$

□

Similar matrices

Definition 56 (Similar matrices)**.** Let $A, B \in \mathbb{M}_{n,n}$. Then they are similar ($A \sim B$), if and only if there exists an invertible matrix P such that $B = P^{-1}AP$.

The properties of the similarity relation are:

- $A \sim A$
- If $A \sim B$, then $B \sim A$
- If $A \sim B$ and $B \sim C$, then $A \sim C$
- If $A \sim B$, then $A^{-1} \sim B^{-1}$
- If $A \sim B$, then $A^T \sim B^T$
- If $A \sim B$, then $A^k \sim B^k$ for some $k \geq 1$
- If $A \sim B$, then A and B have the same determinant, rank, trace, characteristic polynomial, and eigenvalues.

The similarity relation $\sim$ constitutes an equivalence relation on the set $\mathbb{M}_{n,n}$.

Appendix B

Coding implementation of the SGF

The Spectral Graph Forge algorithm as outlined in 4.2.2 was replicated through a Python implementation. The code, reported below, consists of a function that take as input:

- a graph G
- a string "transformation" (either "identity" or "modularity")
- a parameter α (bounded in the interval $[0, 1]$)
- a string "normalization_type" (either "truncate", "logistic" or "scale")
- a parameter k for the logistic function (bounded in the interval $[2, 10]$)

The function returns a graph W that is the output of the Spectral Graph Forge methodology.

```
def SGF(G, transformation = "modularity", alpha = 0.8, normalization_type = "truncate", k = 6
    ):

#-------------------------------------------
# INPUT MATRIX

# Computation of M
A = nx.adjacency_matrix(G).toarray()
M = np.zeros((len(A), len(A)))
if transformation == "identity":
    M = A
if transformation == "modularity":
    degrees = [G.degree[node] for node in G.nodes()]
    m = sum(degrees) / 2
    B = A - np.outer(degrees, degrees) / (2 * m)
    M = B

#-------------------------------------------
# LOW RANK ALPHA APPROXIMATION

# Computation of eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eigh(M)
```

```
    eigenvectors = eigenvectors.T

    # sorting the eigenvectors, eigenvalues
    paired_sorted_list = sorted(zip(eigenvalues, eigenvectors), key=lambda x: abs(x[0]),
        reverse=True)
    eigenvalues_sorted, eigenvectors_sorted = zip(*paired_sorted_list)

    # Computation of M_tilde
    M_tilde = np.zeros((eigenvectors_sorted[0].shape[0], eigenvectors_sorted[0].shape[0]))
    for i in range(math.ceil(alpha * len(eigenvalues))):
        contribution = eigenvalues_sorted[i] * np.outer(eigenvectors_sorted[i],
            eigenvectors_sorted[i])
        M_tilde += contribution

    #-------------------------------------------
    # BACK TRANSFORMATION

    # Computation of A_tilde
    A_tilde = np.zeros((len(M_tilde), len(M_tilde)))
    if transformation == "identity":
        A_tilde = M_tilde
    if transformation == "modularity":
        A_tilde = M_tilde + np.outer(degrees, degrees) / (2 * m)

    #-------------------------------------------
    # NORMALIZATION

    A_dots = np.zeros((len(A_tilde), len(A_tilde[0])))
    # logistic
    if normalization_type == "logistic":
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                A_dots[i][j] = 1 / (1 + math.exp((0.5 - A_tilde[i][j])*k))

    # truncation
    if normalization_type == "truncate":
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                if A_tilde[i][j] < 0:
                    A_dots[i][j] = 0
                elif A_tilde[i][j] > 1:
                    A_dots[i][j] = 1
                else:
                    A_dots[i][j] = A_tilde[i][j]

    # scaling
    if normalization_type == "scale":
        A_tilde_flattened = A_tilde.flatten()
        for i in range(len(A_tilde)):
            for j in range(len(A_tilde[i])):
                A_dots[i][j] = (A_tilde[i][j] - min(A_tilde_flattened)) / (
                    max(A_tilde_flattened) - min(A_tilde_flattened))

    #-------------------------------------------
    # ADJACENCY MATRIX GENERATION
```

```
74
75      # Bernoulli sampling
76      A_prime = np.zeros((len(A_dots), len(A_dots)))
77      for i in range(len(A_tilde)):
78          for j in range(len(A_tilde)):
79              u = random.uniform(0, 1)
80              if u < A_dots[i][j]:
81                  A_prime[i][j] = 1
82                  A_prime[j][i] = 1
83              else:
84                  A_prime[i][j] = 0
85                  A_prime[j][i] = 0
86              if i == j:
87                  A_prime[i][j] = 0
88
89      W = nx.from_numpy_array(A_prime)
90      return W
```

Listing B.1: Spectral Graph Forge implementation

Appendix C

Coding implementation of the Extended SGF

The Spectral Graph Forge algorithm extended with the new transformations elencated in Table 4.1 was implemented in Python. The function along with th ecode used for the simulations is reported below. The function takes as input:

- a graph G
- a string "transformation" ("identity", "modularity", "Laplacian", "signless_Laplacian", "general_Zagreb", "sum_connectivity", "transition_rw", "Bethe_Hessian", "FLCM")
- a parameter α (bounded in the interval $[0, 1]$)
- a string "normalization_type" (either "truncate", "logistic" or "scale")
- a parameter k for the logistic function (bounded in the interval $[2, 10]$)

The function returns a graph W that is the output of the Spectral Graph Forge methodology.

```
#-------------------------------------------
# SPECTRAL GRAPH FORGE

def SGF(G, transformation = "modularity", alpha = 0.8, normalization_type = "truncate", k = 6
    ):

    #-------------------------------------------
    # INPUT MATRIX

    # Setting up basic parameters
    A = nx.adjacency_matrix(G).toarray()
    n = len(A)
    degrees = [G.degree[node] for node in G.nodes()]
    M = np.zeros((n, n))

```

```
        # identity transformation
        if transformation == "identity":
            M = A

        # modularity transformation
        elif transformation == "modularity":
            m = sum(degrees) / 2
            B = A - np.outer(degrees, degrees) / (2 * m)
            M = B

        # laplacian transformation
        elif transformation == "Laplacian":
            D = np.diag(degrees)
            L = D - A
            M = L

        # signless laplacian transformation
        elif transformation == "signless_Laplacian":
            D = np.diag(degrees)
            L_signless = D + A
            M = L_signless

        # General Zagreb transformation
        elif transformation == "general_Zagreb":
            D = np.diag(degrees)
            GZ = pow(D,3) + A
            M = GZ

        # Sum connectivity transformation
        elif transformation == "sum_connectivity":
            SC = np.zeros((n, n))
            for i in range(n):
                for j in range(n):
                    if A[i][j] == 1:
                        SC[i][j] = A[i][j] * (1 / math.sqrt((degrees[i] * degrees[j])))
            M = SC

        # Transition Random Walk transformation
        elif transformation == "transition_rw":
            D = np.diag(degrees)
            P = np.linalg.inv(D) @ A
            M = P

        # Bethe-Hessian transformation
        elif transformation == "Bethe_Hessian":
            # Computing the non-backtracking matrix
            Gdirect = G.to_directed()
            S = np.zeros((len(Gdirect.edges),len(G.nodes)))
            T = np.zeros((len(G.nodes),len(Gdirect.edges)))
            for i,a in enumerate(Gdirect.edges):
                for j,b in enumerate(G.nodes):
                    if a [ 1 ] == b:
                        S[i,j]=1
                    if a [ 0 ] == b :
                        T[j,i] = 1
```

```
            tau = np.zeros((len(Gdirect.edges),len(Gdirect.edges)))
            for i,a in enumerate(Gdirect.edges):
                for j,b in enumerate(Gdirect.edges):
                    if a[0]==b[1] and a[1]==b[0]:
                        tau[i][j] = 1
            B = S@T - tau
            # Computing the Bethe-Hessian matrix
            D = np.diag(degrees)
            I = np.eye(n, n)
            rho = max(abs(np.linalg.eigvals(B)))
            # rho = np.mean(degrees) # alternative way to compute rho
            r = pow(rho, 1/2)
            # r = sum(d**2 for v, d in nx.degree(G)) / sum(d for v, d in nx.degree(G)) - 1 #
                alternative way to compute r
            H = (pow(r,2) - 1) * I - r * A + D
            M = H

        elif transformation == "FLCM":
            s = np.sum((np.sum(A, axis=1)) ** 2)
            D = np.diag(degrees)
            I = np.eye(n, n)
            T = A - (1/3)*(np.outer(degrees, degrees) / (2*s)) + (1/3)*I
            M = T

        else:
            raise ValueError("Invalid transformation")

        #-------------------------------------------
        # LOW RANK ALPHA APPROXIMATION

        # Computation of eigenvalues and eigenvectors
        eigenvalues, eigenvectors = np.linalg.eigh(M)
        eigenvectors = eigenvectors.T

        # sorting the eigenvectors, eigenvalues
        paired_sorted_list = sorted(zip(eigenvalues, eigenvectors), key=lambda x: abs(x[0]),
            reverse=True)
        eigenvalues_sorted, eigenvectors_sorted = zip(*paired_sorted_list)

        # Computation of M_tilde
        M_tilde = np.zeros((eigenvectors_sorted[0].shape[0], eigenvectors_sorted[0].shape[0]))
        for i in range(math.ceil(alpha * len(eigenvalues))):
            contribution = eigenvalues_sorted[i] * np.outer(eigenvectors_sorted[i],
                eigenvectors_sorted[i])
            M_tilde += contribution

        #-------------------------------------------
        # BACK TRANSFORMATION

        # Computation of A_tilde
        A_tilde = np.zeros((n, n))
        if transformation == "identity":
            A_tilde = M_tilde
        elif transformation == "modularity":
            A_tilde = M_tilde + np.outer(degrees, degrees) / (2 * m)
```

```
122         elif transformation == "Laplacian":
123             A_tilde = D - M_tilde
124         elif transformation == "signless_Laplacian":
125             A_tilde = M_tilde - D
126             A_tilde = D_sqrt @ (np.eye(A.shape[0]) - M_tilde) @ D_sqrt
127         elif transformation == "general_Zagreb":
128             A_tilde = M_tilde - pow(D,3)
129         elif transformation == "sum_connectivity":
130             for i in range(n):
131                 for j in range(n):
132                     if A[i][j] == 1:
133                         A_tilde[i][j] = M_tilde[i][j] * math.sqrt((degrees[i] * degrees[j]))
134         elif transformation == "transition_rw":
135             A_tilde = D @ M_tilde
136         elif transformation == "Bethe_Hessian":
137             A_tilde = ((r**2 - 1) * I - M_tilde + D) * (1 / r)
138         elif transformation == "FLCM":
139             A_tilde = M_tilde + (1/3)*(np.outer(degrees, degrees) / (2*s)) - (1/3)*I
140 
141         #-----------------------------------------
142         # NORMALIZATION
143 
144         A_dots = np.zeros((n, n))
145         # logistic
146         if normalization_type == "logistic":
147             for i in range(n):
148                 for j in range(n):
149                     A_dots[i][j] = 1 / (1 + math.exp((0.5 - A_tilde[i][j])*k))
150 
151         # truncation
152         elif normalization_type == "truncate":
153             for i in range(n):
154                 for j in range(n):
155                     if A_tilde[i][j] < 0:
156                         A_dots[i][j] = 0
157                     elif A_tilde[i][j] > 1:
158                         A_dots[i][j] = 1
159                     else:
160                         A_dots[i][j] = A_tilde[i][j]
161 
162         # scaling
163         elif normalization_type == "scale":
164             A_tilde_flattened = A_tilde.flatten()
165             for i in range(n):
166                 for j in range(n):
167                     A_dots[i][j] = (A_tilde[i][j] - min(A_tilde_flattened)) / (
                            max(A_tilde_flattened) - min(A_tilde_flattened))
168 
169         #-----------------------------------------
170         # ADJACENCY MATRIX GENERATION
171 
172         # Bernoulli sampling
173         A_prime = np.zeros((n, n))
174         for i in range(n):
175             for j in range(n):
```

```
176                 u = random.uniform(0, 1)
177                 if u < A_dots[i][j]:
178                     A_prime[i][j] = 1
179                     A_prime[j][i] = 1
180                 else:
181                     A_prime[i][j] = 0
182                     A_prime[j][i] = 0
183                 if i == j:
184                     A_prime[i][j] = 0
185
186         W = nx.from_numpy_array(A_prime)
187         return W
```

Listing C.1: Extended Spectral Graph Forge