
Stockholms universitet

# Coarse graining and out-of-sample approximation for the spectral theory of complex networks

Mikael Rizvanovic

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2023:5
Matematisk statistik
Juni 2023

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2023:5**
http://www.math.su.se

# Coarse graining and out-of-sample approximation for the spectral theory of complex networks

Mikael Rizvanovic*

June 2023

**Abstract**

Spectral graph theory have many applications in machine learning and beyond. The graph has been shown to be a very powerful mathematical object and much can be said about it from its spectrum (eigenvectors and eigenvalues) alone. Nevertheless, this relies on us being able to compute the spectrum which is notoriously expensive and often unfeasible for even moderately large data sets. In this thesis we will look at ways to bring down this computational cost while hopefully preserving most of the relevant information in the graph. We will examine two methods to accomplish this: 1) Coarse graining, which reduces the overall size of the graph and thus also the computational cost, and 2) Out-of-sample extension, where we extend an already known eigenspace to new data points.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: Mikael.Rizvanovic@gmail.com. Supervisor: Chun-Biu Li.

## Acknowledgements

I would like to thank my supervisor Chun-Biu Li for his support and guidance throughout this project. I would also like to thank Tobias Wängberg and Joanna Tyrcha for the many discussions we had regarding the work in this thesis.

# Contents

# 1 Introduction

Graphs are mathematical structures that model relationships between objects. They were first formalized by Leonhard Euler in 1736 for a proof of the Seven Bridges of Königsberg problem. This problem asked for a path through the city that crossed each of its (at the time) seven bridges exactly once. By representing the problem as a graph Euler could ignore everything but the most relevant topological features of the city and provided a simple and elegant proof that no such path exits.

Today, graphs are used to solve a wide range of problems in mathematics and adjacent fields. For example, to model protein structure, friendship networks, data structures, energetic states of subatomic particles and much more. Many applications rely on a special matrix representation of graphs, called the *graph Laplacian*.

Much of what we will discuss in this thesis is related to spectral graph theory, which is an area that uses eigenvectors and eigenvalues to study the graph. It is perhaps not obvious why we would use spectral theory for this. After all, many of the traditional graph problems do not mention anything about the spectrum. Despite this, the graph's spectrum have been shown to reveal a lot of information about graphs, much of which is of particular importance in machine learning and computer science. Some of the applications in machine learning are dimensional reduction, clustering, recommendation systems, computer vision etc. Cvetković and Simić [5] published an extensive survey of spectral graph theory applications in computer science.

One of the major problems of graphs theory, and spectral graph theory in particular, is the high computational cost of storing the graph, computing the spectrum and performing operations on them in general. To represent all relationships in a graph with $n$ nodes (objects) we are required to store a $n \times n$ matrix, if we want to compute the spectrum then that is done in $O(n^3)$ time. There are various methods one can use that reduces the memory cost, these will not be a focus in this thesis. Instead, we will study ways of reducing the time complexity for solving the eigenproblem.

The outline of this thesis is as follows. In section 2 we provide some background theory, with a focus on spectral graph theory. Then in section 3 we will present two methods that reduces the overall time complexity of solving the eigenproblem. In section 4 we will evaluate how much information of the original spectrum is retained when using these methods. Finally, in section 5 we will finish with a discussion.

# 2 Background

## 2.1 Eigenvalues and eigenvectors

It is assumed that the reader already have a basic understanding of eigenvalues and eigenvectors but for completeness we will give a short recap. All theory in this subsection comes from [11], unless otherwise specified.

Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a matrix. A scalar $\lambda \in \mathbb{R}$ is an *eigenvalue* of $\boldsymbol{A}$ corresponding to the non-zero *eigenvector* $v \in \mathbb{R}^{n \times 1}$ if it satisfies the *eigenproblem*

$$\boldsymbol{A}v = \lambda v. \tag{1}$$

Every solution to the eigenproblem $(\lambda_k, v_k)$ is called an *eigenpair* and the set of all eigenpairs of $\boldsymbol{A}$ is called the *spectrum* of $\boldsymbol{A}$.

We can think of the matrix as a linear transformation of a vector from $\mathbb{R}^n$ to $\mathbb{R}^n$. The eigenvectors are special because this transformation will only multiply them by some constant, other vectors will also point in a different direction. Each matrix has its own set of eigenpairs.

Solving equation (1) gives the right eigenvector, the left eigenvector solves $u^T\boldsymbol{A} = \lambda u^T$. We note that the left and right eigenvectors are not always the same, but their eigenvalues are[1]. Every $n \times n$ matrix have $n$ eigenpairs but not necessarily $n$ unique eigenvalues or $n$ orthogonal eigenvectors, with an important exception for symmetric matrices.

**Theorem 1** *If $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{A} = \boldsymbol{A}^T$ then there exists $n$ orthogonal eigenvectors. Furthermore, if we gather the eigenvectors in a matrix $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ and their corresponding eigenvalues on the diagonal of $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times n}$ then*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T = \sum_{k=1}^{n} \lambda_k v_k v_k^T.$$

This is known as the *eigendecomposition* of $\boldsymbol{A}$. It means that all information in $\boldsymbol{A}$ is contained in $\boldsymbol{A}$'s spectrum. This also means that the left and right eigenvectors for symmetric matrices are the same. Naturally, since real symmetric matrices behave so nicely, they are of special interest to us.

### 2.1.1 Complexity of the eigenproblem

Practically, the eigenproblem is solved with numerical methods. There are two broad types of methods, *direct* and *iterative*. Different eigenproblem algorithms can offer some advantages or disadvantages over others but the leading time complexity term is always $O(n^3)$ [9]. It is not the purpose of this thesis to study these algorithms but we will give a very brief overview based mostly on chapter

[1] Assuming $\boldsymbol{A}$ is a square matrix.

8 & 10 of [9].

With direct methods the full eigendecomposition is computed. These methods store the full matrix in memory and therefore do not fully profit from matrix sparsity[2] but they do compute the exact eigenpairs. Several direct methods have been suggested, some of which are more time efficient then others but they all require an initial tridiagonalization of the matrix which have time complexity $O(n^3)$.

Iterative methods start with a few initial vectors and approximate eigenpairs from low dimensional subspaces. Each iteration improves the precision of this approximation and the algorithm stops when improvements between iterations are below some threshold. After the initial eigenpairs have been found the algorithm moves on to find more. It is thus possible to stop after a subset of $k < n$ eigenpairs has been found. Iterative methods do not require us to store the full matrix in memory and are thus better suited for large eigenrpoblems. Because of this, they are also able to fully profit from matrix sparsity. There are several indirect methods but their leading time complexity term are $O(kn^2)$ for dense matrices and $O(k^2n)$ for sparse. We note that the time complexity for iterative methods are still $O(n^3)$ when the whole spectrum is computed (even if the matrix is sparse). Further improvements can be made if we only want to compute extreme eigenvalues, i.e., the largest and smallest.

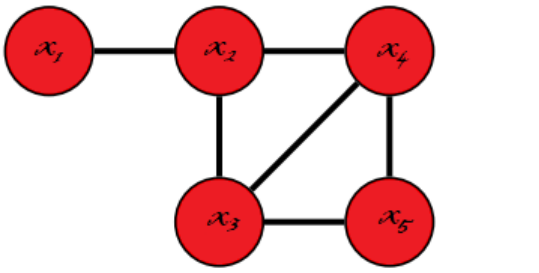
## 2.2 Spectral graph theory

Spectral graph theory studies the properties of graphs via the eigenpairs of their associated graph matrices, that is, through the adjacency matrix, graph Laplacian and its variants.

Graphs represent similarities between objects. Each object is represented as a node and similar nodes are pairwise connected to each other with edges. These edges can be either directed or undirected, directed edges go from one node to the other but not back. Edges can also be weighted, representing that some nodes are more similar than other. A very simple graph is illustrated in Figure 1a). The adjacency matrix can be used to represent the graph, Figure 1b), and the Laplacian can be used instead of the adjacency matrix in a way that is more mathematically convenient. In this thesis we will study undirected & weighted graphs. The unweighted graph is just a special case of the weighted one so the results here will be valid for that graph as well. The directed graph, however, does not have a symmetric adjacency matrix and some results we present will therefore not be valid for directed graphs (although others are).

Graphs has many applications in mathematics and related fields. Because of that, this topic has been very well studied over the years from many different

[2] A matrix is said to be sparse if many of its elements are 0.




**a)** An unweighted & undirected graph

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

**b)** Adjacency matrix of a)

Figure 1: a) Example of a simple graph. b) The adjacency matrix for the graph in a).

perspectives and we cannot discuss all of them here. Instead, we will just give a broad overview of some of the most important results, a much more in depth treatment can be found in [4] and [16].

### 2.2.1 Basic notations

We will begin by introducing basic graph notations that will be used throughout this thesis.

Let $G = (\Omega, E)$ be an undirected graph where $\Omega = \{x_1, ..., x_n\}$ is the set of nodes and $E$ is the set of edges between nodes. We will assume that the graph is weighted, that means each edge carries a non-negative weight $w_{ij} \geq 0$. Since $G$ is undirected we have $w_{ij} = w_{ji}$ and if $w_{ij} = 0$ then the nodes $x_i$ and $x_j$ are not connected by an edge. The weighted adjacency matrix of the graph is $\boldsymbol{A} = (w_{ij})_{i,j=1,...,n}$. That is, it represents all pairwise weights. We note that $\boldsymbol{A}$ is symmetric. The degree of a node $x_i$ is defined as

$$d(x_i) = d_i = \sum_{j=1}^{n} w_{ij}.$$

The degree matrix, $\boldsymbol{D}$, is the diagonal matrix with the degrees $d_1, ..., d_n$ on its diagonal. The *volume* of a graph is the sum of all degrees, i.e.

$$\text{vol}(G) = \sum_{i=1}^{n} d_i.$$

We say that the graph is connected if there is a path of edges connecting any two nodes in the graph. Unless otherwise specified, we will always assume that the graph is connected.

### 2.2.2 Graph Laplacians

In this section we will define different graph Laplacians and point out some of their basic properties. We will distinguish between different variants of graph Laplacians but note that there is no unique convention in the literature[3]. The three graph Laplacians we will study are

$$\begin{aligned} \boldsymbol{L} &= \boldsymbol{D} - \boldsymbol{A}, && \text{(unnormalized Laplacian)} \\ \boldsymbol{\mathcal{L}} &= \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}, && \text{(normalized Laplacian)} \\ \boldsymbol{L}_{\text{rw}} &= \boldsymbol{D}^{-1}\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}. && \text{(Random walk Laplacian)} \end{aligned}$$

Because $\boldsymbol{A}$ is symmetric, so is $\boldsymbol{L}$. Both $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{L}_{\text{rw}}$ are normalized but $\boldsymbol{\mathcal{L}}$ is normalized in a way that makes sure it is symmetric. As we saw in Theorem 1, these have several properties that make them more appealing to study from a spectral point of view. The random walk Laplacian is not symmetric but it is closely related to a random walk on a graph and thus has a nice and intuitive interpretation, we will discuss this in greater detail in section 2.2.3.

These three Laplacians are all related in the following way.

$$\begin{aligned} \boldsymbol{L} &= \boldsymbol{D}^{1/2}\boldsymbol{\mathcal{L}}\boldsymbol{D}^{1/2} = \boldsymbol{D}\boldsymbol{L}_{\text{rw}} \\ \boldsymbol{\mathcal{L}} &= \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{D}^{1/2}\boldsymbol{L}_{\text{rw}}\boldsymbol{D}^{-1/2} \\ \boldsymbol{L}_{\text{rw}} &= \boldsymbol{D}^{-1}\boldsymbol{L} = \boldsymbol{D}^{-1/2}\boldsymbol{\mathcal{L}}\boldsymbol{D}^{1/2}. \end{aligned}$$

Because of this relationship they also share many properties. We will summarize the main ones in the following proposition.

**Proposition 1** *The different graph Laplacians satisfy the following properties*

1. $\boldsymbol{L}$ *and* $\boldsymbol{\mathcal{L}}$ *are symmetric.*
2. $(\lambda_k, v_k)$ *is an eigenpair of* $\boldsymbol{L}_{rw}$ *if and only if* $(\lambda_k, v_k\boldsymbol{D}^{-1/2})$ *is an eigenpair of* $\boldsymbol{\mathcal{L}}$*.*
3. $(\lambda_k, v_k)$ *is an eigenpair of* $\boldsymbol{L}_{rw}$ *if and only if* $\lambda_k$ *and* $v_k$ *solve the generalized eigenproblem* $v_k\boldsymbol{L} = \lambda_k\boldsymbol{D}v_k$*.*
4. *The smallest eigenvalue of* $\boldsymbol{L}$ *is 0, the corresponding eigenvector is the constant one vector* $\mathbb{1}$*.*
5. *The smallest eigenvalue of* $\boldsymbol{L}_{rw}$ *is 0, the corresponding right eigenvector is* $\mathbb{1}$*. The smallest eigenpair of* $\boldsymbol{\mathcal{L}}$ *is* $(0, \boldsymbol{D}^{1/2}\mathbb{1})$*.*
6. $\boldsymbol{L}, \boldsymbol{\mathcal{L}}$ *and* $\boldsymbol{L}_{rw}$ *are positive semi-definite and have* $n$ *non-negative, real valued eigenvectors* $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq 2$*.*

[3] Often the generic term "graph Laplacian" is used to describe whatever type of Laplacian the author studies.

Property 3 follows directly from $\boldsymbol{L} = \boldsymbol{D}\boldsymbol{L}_{\text{rw}}$. A proof for property 6 can be found in [4, section 1]. We will indirectly prove the remaining properties in the next section to build intuition for future results. We emphasize that property 2 shows there is a direct relationship between the spectrum of $\boldsymbol{L}_{\text{rw}}$ and $\boldsymbol{\mathcal{L}}$. So, by studying one of them we gain insight into the other as well.

Which Laplacian to use will depend on the situation. Usually one of the two normalized Laplacians ($\boldsymbol{L}_{\text{rw}}$ or $\boldsymbol{\mathcal{L}}$) are preferred as their spectrum is consistent the spectrum in stochastic processes and spectral geometry [4]. In some situations, such as splitting the graph into subgraphs, the two types of Laplacians will optimize different objectives (based on subgraph volume or cardinality), see [18] for more on this. We will focus on the normalized Laplacians in this thesis.

We are also going to allow the nodes to have self-loops (or self-edges), i.e. $w_{ii} \geq 0$. Self-loops do not change to unnormalized Laplacian, $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, as the new weight on the diagonal of $\boldsymbol{A}$ gets canceled out by the extra term in $\boldsymbol{D}$. However, the spectrum of the two normalized Laplacians changes in line with property 2 & 3 of Proposition 1.

The second smallest eigenvalue, $\lambda_2$, is of particular importance. A graph is connected if and only if $\lambda_2 > 0$ [16]. Furthermore, the multiplicity of eigenvalue 0 corresponds to the number of connected components, each with eigenvector $\mathbb{1}$ (or $\boldsymbol{D}^{1/2}\mathbb{1}$ for the normalized Laplacian). Cheeger's inequality generalizes this further and says that $\lambda_2$ is large if and only if the graph is well-connected. One can define graph connectedness in several different ways but simplified we just say that a graph is poorly connected if one can cut off many nodes by only removing a few edges.

### 2.2.3 Random walks on graphs

We will now look at the random walk on a graph, connect it to the random walk Laplacian and examine some of its properties. We begin with some definitions. Let $G = (\Omega, E, \boldsymbol{A})$ be a weighted undirected graph. A random walk on $G$ is a process that begins at some node and at each time step transitions to another with some probability. If the graph is unweighted then this probability will be uniform among all nodes with edges to the starting node. When the graph is weighted then the transition probability will be proportional to the weight of the corresponding edge [4], i.e.

$$p(x_j|x_i) = \frac{w_{ij}}{d(x_i)}.$$

Here we use the notation $p(x_j|x_i)$ to denote the probability of going to node $x_j$ given that we are in node $x_i$. We note that the weights are symmetric but the transition probabilities are generally not, i.e. $p(x_j|x_i) \neq p(x_i|x_j)$. This is because the degree can vary between nodes. We can write this in matrix form

as

$$\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}.$$

We call $\boldsymbol{P}$ the *transition matrix*. This matrix describes the random walk on the entire graph.

We mentioned earlier that the transition matrix $\boldsymbol{P}$ and random walk Laplacian $\boldsymbol{L}_{\text{rw}}$ are related. To see this, let $(\alpha_k, \boldsymbol{\phi}_k)$ be an eigenpair of $\boldsymbol{P}$ then,

$$\boldsymbol{\phi}_k \boldsymbol{L}_{\text{rw}} = \boldsymbol{\phi}_k \left(\boldsymbol{I} - \boldsymbol{P}\right) = \boldsymbol{\phi}_k - \alpha_k \boldsymbol{\phi}_k = (1 - \alpha_k)\boldsymbol{\phi}_k.$$

So $\boldsymbol{L}_{\text{rw}}$ have the same left eigenvectors as $\boldsymbol{P}$ but with corresponding eigenvalues $\lambda_k = 1 - \alpha_k$, one can show the same thing for right eigenvectors. This shows that we can prove properties for the two normalized Laplacians (as $\boldsymbol{L}_{\text{rw}}$ and $\boldsymbol{\mathcal{L}}$ are directly related) by studying the transition matrix and vice versa.

In the remainder of this section we will indirectly prove most of the properties in Proposition 1 by proving their equivalence for $\boldsymbol{P}$. Throughout the rest of this thesis, we will adopt the notation $\alpha$ to denote eigenvalues of $\boldsymbol{P}$ and $\lambda = 1 - \alpha$ for eigenvalues of the Laplacians.

By taking powers of $\boldsymbol{P}$ we get the transition probability between nodes in a given number of steps, e.g. the element $\boldsymbol{P}^t_{ij}$ describes the probability of going from node $x_i$ to $x_j$ in exactly $t$ steps. As the random walker progresses through time, he will spend a greater proportion of time in some nodes over others. After enough time have passed, the distribution will reach an equilibrium and continuing the random walk will no longer change the proportion of time spent in each node. That is,

$$\boldsymbol{\phi}_0 \boldsymbol{P} = \boldsymbol{\phi}_0 \tag{2}$$

where $\boldsymbol{\phi}_0$ is the distribution of time spent in each node. This is known as the *stationary distribution*.

If the graph is connected and non-bipartite[4] then the stationary distribution is unique and given by [4]

$$\boldsymbol{\phi}_0(x) = \frac{d(x)}{\text{Vol}(G)} \tag{3}$$

for every node $x$. See [4, section 1.5] for a proof. In this thesis we always assume that a unique stationary distribution exists. We will however note that, since we allow self-loops, the graph is automatically non-bipartite if $p(x_i|x_i) > 0$ for at least one node $x_i$. So, practically, it would be enough to just make sure that the graph is connected.

Now let us analyze the spectrum of $\boldsymbol{P}$. Our first observation is that $\boldsymbol{P}$ is

[4] A graph is said the be bipartite if we can assign each node into one of two classes in such a way that the edges from one node will always go only to nodes of the opposite class.

not symmetric, meaning that the left and right eigenvectors differ. But we also notice that $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$ is very similar to

$$\boldsymbol{S} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$$

which is symmetric. Specifically, we can write

$$\boldsymbol{P} = \boldsymbol{D}^{-1/2}\left(\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}\right)\boldsymbol{D}^{1/2} = \boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{1/2}. \tag{4}$$

Now, since $\boldsymbol{S}$ is symmetric we know that it has a complete set of orthonormal eigenvectors and real eigenvalues (see Theorem 1). Suppose $(\alpha_k, v_k), k = 1, ..., n$, is an eigenpair of $\boldsymbol{S}$. If we gather the eigenvectors in $\boldsymbol{V}$ and eigenvalues on the diagonal of $\boldsymbol{\Lambda}$ then we can express $\boldsymbol{S}$ with the eigendecomposition as

$$\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\boldsymbol{T}} = \sum_{k=1}^{n} \lambda_k v_k v_k^T.$$

If we insert this into (4) we see that

$$\boldsymbol{P} = \boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{1/2} = \boldsymbol{D}^{-1/2}\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{D}^{1/2}$$

Now let

$$\boldsymbol{\Phi} = \boldsymbol{D}^{-1/2}\boldsymbol{V} \qquad \text{and} \qquad \boldsymbol{\Psi} = \boldsymbol{D}^{1/2}\boldsymbol{V}$$

then clearly

$$\boldsymbol{P} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Psi}^T.$$

This means that the left and right eigenvectors of $\boldsymbol{P}$ are $\boldsymbol{\phi}_k = \boldsymbol{D}^{-1/2}v_k$ and $\boldsymbol{\psi}_k = \boldsymbol{D}^{1/2}v_k$ respectively. We note that this implies $\boldsymbol{\phi} = \boldsymbol{D}^{-1}\boldsymbol{\psi}$. Since $v_k$ are orthogonal it is easy to confirm that $\langle \boldsymbol{\phi}_i, \boldsymbol{\psi}_j \rangle = \delta_{ij}$ where $\delta_{ij}$ is the Kronecker delta. We say that the columns of $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ form a bi-orthogonal system, i.e.

$$\boldsymbol{P}^t = \left(\boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Psi}^T\right)^t = \boldsymbol{\Phi}\boldsymbol{\Lambda}^{\boldsymbol{t}}\boldsymbol{\Psi}^T = \sum_{k=1}^{n} \alpha_k^t \boldsymbol{\psi}_k^T \boldsymbol{\phi}_k. \tag{5}$$

This will be useful later in the thesis.

We have already seen one eigenpair of $\boldsymbol{P}$ in equation (2). That is, the first left eigenvector of $\boldsymbol{P}$ (corresponding to eigenvalue 1) is the stationary distribution of $\boldsymbol{P}$. Since $\boldsymbol{P}$ is row stochastic, it is easy to confirm that $\mathbb{1}$ is the corresponding right eigenvector. The remaining eigenvectors cannot be derived so easily but we can sometimes put bounds on their eigenvalues. The Perron-Frobenius theorem states that if $G$ is a connected and weighted graph then $\alpha_1 \geq -\alpha_n$ with equality if and only if $G$ is bipartite [16, theorem 4.5.1]. Since the first eigenvalue is 1 this means that $|\alpha_k| \leq 1$.

It is usually of interest to be able to say something about the number of steps

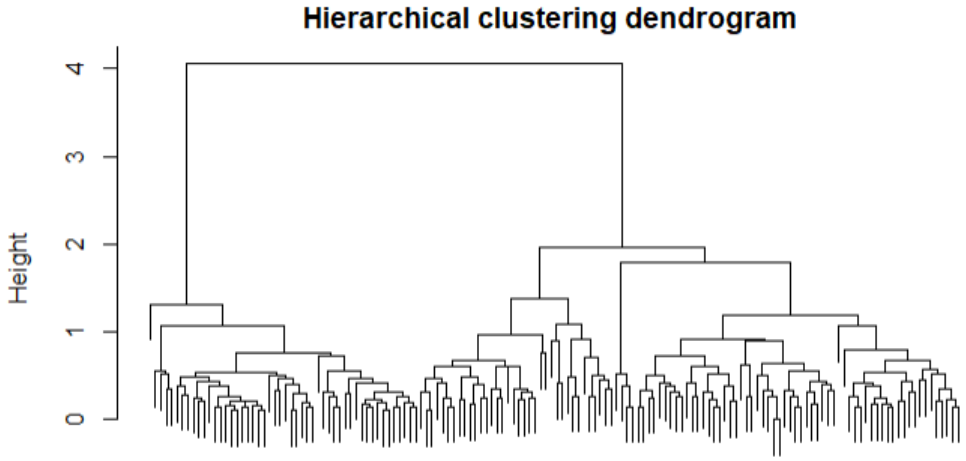



Figure 2: Example of a dendrogram produced by the hierarchical clustering algorithm. Figure is produced by running the agglomerative algorithm on the IRIS[6] dataset with Euclidean distance and average linkage. The height represents the Euclidean distance between nodes/groups where they have been merged.

it takes for the walk to be close to its stationary distribution. It is possible to show that the time is bounded by an expression which depends on the second eigenvalue $\alpha_2$, see [4, section 1.5]. This is a special case of Cheeger's inequality. Informally, we can explain this with the bi-orthogonal system in (5). Each left/right eigenvector pair forms a direction in the eigenspace with their corresponding eigenvalue representing the prominence of that direction. With $\alpha_2 = 1 - \lambda_2$ corresponding to the eigenvector pair with the most prominent direction (largest). If we now think of the random walk on this eigenspace instead of on the graph then the random walk will reach its stationary distribution on the whole space when it has been reached in each direction individually. So, a bound for the convergence time should depend on the direction in which this takes the longest. This is the direction corresponding to eigenvalue $\alpha_2$ because traversing this direction requires the largest number of steps.

## 2.3 Hierarchical clustering

We are now going to introduce a clustering method. Clustering aims at grouping together a set of objects (data points) in such a way that objects in the same group (cluster) are more similar to each other (in some way) than to objects in other groups. There exist many different clustering methods, see [10, chapter 14] for a discussion one some of the most popular methods. Here, we will just

[6]This is a standard dataset which was introduced by R.A. Fisher in 1936 and today comes preinstalled in most statistical software packages, such as R. The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

introduce one method called hierarchical clustering that we will use later in the thesis to group together similar nodes.

As the name suggests, hierarchical clustering produces hierarchical representations of data in which the clusters at each level of the hierarchy are created by merging clusters from the level below. This means that at the lowest level each "cluster" only contains a single observation while at the highest level all data points belong to the same cluster. Merging is done by minimizing some dissimilarity measure between (disjoint) groups of observations. The dissimilarity between groups can be calculated in different ways but they are all based on the individual observations in the groups [10, Section 14.3.12]. See Figure (2) for an illustration of what hierarchical clustering can look like.

As we can see from Figure (2), hierarchical clustering can be represented as a dendrogram. The height in the dendrogram where two branches fuse together indicates how different the two groups are. Groups that fuse at the bottom of the tree are quite similar to each other while groups fused close to the top tend to be quite different [10].

One of the advantages of hierarchical clustering is that we do not have to specify the number of clusters beforehand. Instead, the hierarchical clustering algorithm will produce a dendrogram. We can then choose the number of clusters by cutting it at some height (some dissimilarity value). However, this comes at a computational cost. Stopping after a set clusters have been formed are cheaper.

The dendrogram can be constructed in two ways. 1) agglomerative (bottom-up) and 2) divisive (top-down). The bottom-up approach starts from the leafs (single observations) and fuse clusters together pairwise until we end up with one big cluster. The top-down approach does the opposite, it starts with one big cluster and splits it until each cluster only contain one observation [10]. The most common type of hierarchical clustering is the bottom-up approach [10], which we will focus on here.

We can describe the agglomerative HC algorithm in the following way

**Algorithm 1** Agglomerative hierarchical clustering

1. Define some (dis)similarity measure.
2. Calculate the pairwise (dis)similarity between all clusters.
3. Fuse the two most similar clusters .
4. Repeat step 2-3 until only 1 cluster remains.

The time complexity of this algorithm depends on what linkage is used. For average linkage it is $O(\log(n)n^2)$ where $n$ the number of data points.

Dissimilarity between clusters can be calculated in many different ways, this is known as *linkage*. The wikipedia article on HC lists about 20 different linkage methods but many more have been suggested throughout the years. However, the 3 most common are *complete-, single-* and *average* linkage. All these methods start by computing all pairwise dissimilarities between observations in two clusters. Complete and single linkage uses the maximal and minimal dissimilarity respectively while average linkage uses the average dissimilarity [10]. The resulting dendrogram from HC will depend on both the dissimilarity measure used and linkage type. Different linkage methods can result in radically different dendrograms.

One underlying assumption in HC is that data possesses some hierarchical structure. This is not always the case. For example, suppose that our data consists of university students split evenly between two universities and three faculties. We can imagine a scenario where the best division into two clusters will split the students by university while the best division into three clusters will split them by faculty. In this case, data does not possess a hierarchical structure. Meaning that the best division into two clusters does not result from merging two of the three clusters one level below. Therefore, one should consider whether this assumption is fulfilled before using HC.

# 3 Methods

In this section we will present two methods that aims at reducing the computational cost of solving the eigenproblem while hopefully still preserving most of the information in the graph. The first method is a coarse graining procedure, where we reduce the graphs size by merging nodes before solving the eigenproblem. The second approach extends an existing spectrum to encompass new out-of-sample points, without solving the whole eigenproblem again. This could be something we do for new points or simply because the data set is too large for us to solve the full eigenproblem outright.

## 3.1 Coarse graining

The purpose of coarse graining (CG) is to combine nodes that are similar to each other while preserving some properties of the original graph [8]. By doing this, we hope to reduce the total number of nodes in the graph and thus also reduce the computational cost of solving the eigenproblem. However, we still want to preserve as much of the original spectrum as possible, and by extension as much of the information in the original graph as possible, without computing the eigendecomposition of the original graph.

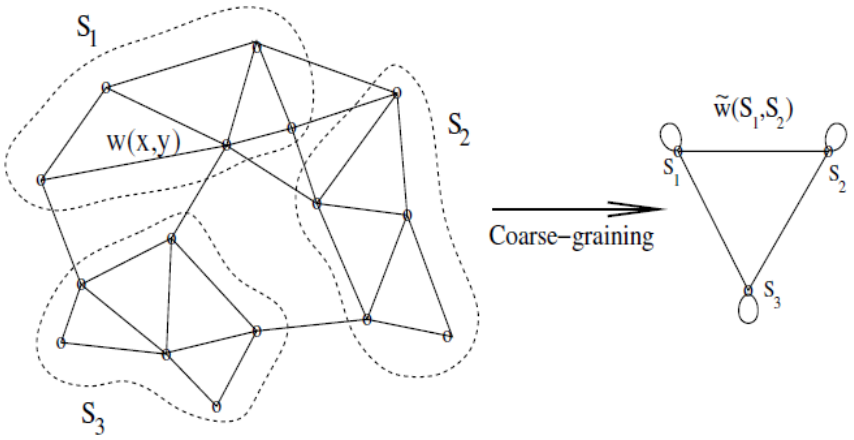



Figure 3: Illustration of CG. Each circle is a node and lines between them are edges, each edge have their own weight. All nodes inside the three dotted areas are combined into three groups of nodes.

In section 2.2.3 we discussed the relationship between the random walk on a graph and its spectrum, we also discussed how the spectrum of different graph Laplacians are related to each other. The thinking now is that by coarse graining the graph in a way that preserves large-scale behavior of the random walk we will also preserve most of the characteristics of the spectrum.

This section is split into 3 subsections. In the first we will look at how the weight matrix changes when nodes are grouped together from the perspective of a random walk. We will then look at how the spectrum changes when we group nodes like this and finally suggest a way to coarse grain the random walk that hopefully captures this change.

Throughout this section, let $\boldsymbol{A}$ be the adjacency matrix, $\boldsymbol{D}$ be the diagonal matrix with entries $d_{ii} = \sum_j \boldsymbol{A}_{ij}$ and $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$ be the transition matrix. We will also use $\boldsymbol{\phi}, \boldsymbol{\psi}$ and $\alpha$ to denote eigenvectors (left and right) and eigenvalues of $\boldsymbol{P}$ respectively. Furthermore, we will use $\boldsymbol{\phi}_k(x)$ to denote the element of the $k$:th (left) eigenvector corresponding to node $x$.

### 3.1.1 Constructing a coarse-grained graph

Our aim here is to work out how the weight matrix changes when we group nodes together. These will make up our new, coarse grained, weight matrix. An illustration of this is shown in Figure 3.

We start by considering an arbitrary partition of the set of nodes $\Omega$ into $k$ groups $\{S_i\}_{1\leq i\leq k}$. Let $p(x|y)$ denote the transition probability of going to node

$x$ given that we are in node $y$ and let $p(x)$ denote the stationary distribution of node $x$. Standard calculations give us

$$\begin{aligned} p(S_i|S_j) &= \frac{p(S_i, S_j)}{p(S_j)} = \sum_{x \in S_j} \frac{p(S_i, x)}{p(S_j)} \\ &= \frac{1}{p(S_j)} \sum_{x \in S_j} p(S_i|x)p(x) \\ &= \frac{1}{p(S_j)} \sum_{x \in S_j} \sum_{y \in S_i} p(y|x)p(x). \end{aligned} \tag{6}$$

Where $p(S_j) = \sum_{x \in S_j} p(x)$ is the stationary distribution for group $S_j$. Recall from section 2.2.3 that

$$p(x|y) = \frac{w(x,y)}{d(x)}, \qquad p(x) = \frac{d(x)}{\text{Vol}(G)} \qquad \text{and} \qquad d(x) = \sum_y w(x,y).$$

Using this in equation (6) gives us the following,

$$p(S_i|S_j) = \frac{1}{d(S_j)} \sum_{x \in S_i} \sum_{y \in S_j} w(y,x) = \frac{1}{d(S_j)} \sum_{y \in S_j} w(y, S_i). \tag{7}$$

Meaning that the transition probability between groups $S_j$ and $S_i$ is just the sum of all edges between them divided by the degree of group $S_j$. We note that this can be computed relatively efficiently by storing the updated edge weights after each node/group merger and using the right most equation in (7).

This means that the updated weights are given by

$$\tilde{w}(S_i, S_j) \propto \sum_{x \in S_j} \sum_{y \in S_i} p(y|x)p(x).$$

We have proportionality here because the transition matrix is invariant to any scalar multiplication of the weight matrix. The reader can trivially confirm this by plugging in $c \cdot w(x,y)$ into $p(x|y)$, where $c$ is some constant.

In order to preserve the total weight of the graph we choose to define the updated weights as

$$\tilde{w}(S_i, S_j) = \text{Vol}(G) \sum_{x \in S_i} \sum_{y \in S_j} p(x|y)p(x) = \sum_{x \in S_i} \sum_{y \in S_j} w(x,y).$$

However, we note that preserving the total weights like this is not strictly necessary. This is because multiplying the weight matrix by a scalar will only scale the eigenvalues, the eigenvectors will still point in the same direction. Thus, if we know the scaling constant we can still retrieve the original spectrum[7].

[7] To see this just compare the characteristic polynomials between $c\boldsymbol{A}$ and $\boldsymbol{A}$, where $c$ is some constant.

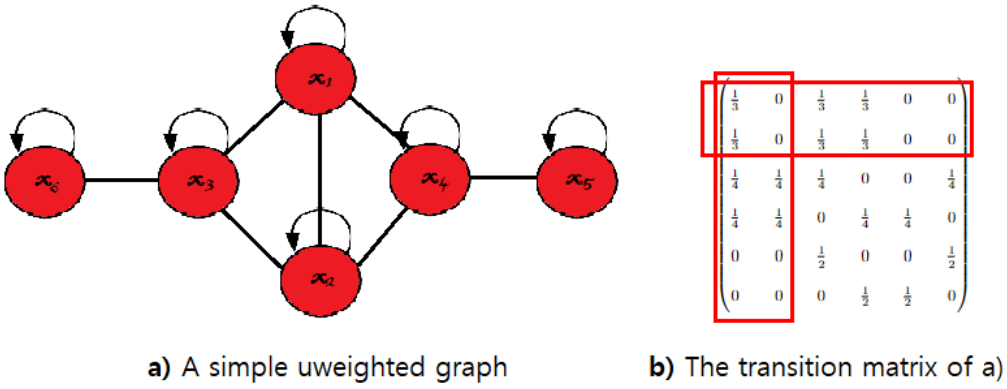



Figure 4: Example of a situation where two nodes ($x_1$ and $x_2$) have the same neighbors and weights. a) A unweighted graph with self loops. b) Corresponding transition matrix, highlighting that column/row 1 and 2 are identical.

We chose this definition to make the eigenvalues of the CG graph more easily comparable with the original graph and different CG depths (i.e., combining different number of nodes).

### 3.1.2 The coarse grained spectrum

We will begin by considering the simple case when two nodes (say node $x_1$ and $x_2$) in an undirected graph have the same neighbors and weights. This example is borrowed from Gfeller and Rios [8]. Figure 4a) displays a simple example when this is the case.

Since node $x_1$ and $x_2$ have the same neighbors and weights, row/column 1 and 2 of the transition matrix will be identical, see Figure 4b). This means that the two nodes are indistinguishable from a random walk point of view. Because of this the first and second element of any right (and left) eigenvector will also be the same, i.e. $\boldsymbol{\psi}_k(x_1) = \boldsymbol{\psi}_k(x_2)$. The obvious coarse graining step now would be to merge these two nodes. This would result in a new graph where the first row/column of its adjacency matrix, $\tilde{\boldsymbol{A}}$, carries the sum of the edges of $\boldsymbol{A}$, as we discussed in the previous section.

At this point, we could get the coarse grained transition matrix, $\tilde{\boldsymbol{P}}$, by normalizing in the same way we got $\boldsymbol{P}$. But we could also write it as a product of three matrices [8],

$$\tilde{\boldsymbol{P}} = \boldsymbol{RPK}. \tag{8}$$

Where $\boldsymbol{R}$ and $\boldsymbol{K}$ are two projection like operators, specifically

$$\boldsymbol{R} = \begin{pmatrix} \frac{p_1}{p_1+p_2} & \frac{p_2}{p_1+p_2} & 0\dots 0 \\ 0 & 0 & \\ \vdots & \vdots & \boldsymbol{I}_{n-2} \\ 0 & 0 & \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{K} = \begin{pmatrix} 1 & 0\dots 0 \\ 1 & 0\dots 0 \\ 0 & \\ \vdots & \boldsymbol{I}_{n-2} \\ 0 & \end{pmatrix}.$$

To simplify the expression we used $p_i = p(x_i)$ for the stationary probability of node $x_i$.

By expressing $\tilde{\boldsymbol{P}}$ like this we can easily show that the vector $\boldsymbol{R}\boldsymbol{\psi}_k$ is a right eigenvector of $\tilde{\boldsymbol{P}}$ with eigenvalue $\alpha_k$. To see this, we first notice that

$$\boldsymbol{K}\boldsymbol{R}\boldsymbol{\psi}_k = \begin{pmatrix} \frac{p_1}{p_1+p_2} & \frac{p_2}{p_1+p_2} & 0\dots 0 \\ \frac{p_1}{p_1+p_2} & \frac{p_2}{p_1+p_2} & 0\dots 0 \\ 0 & 0 & \\ \vdots & \vdots & \boldsymbol{I}_{n-2} \\ 0 & 0 & \end{pmatrix} \boldsymbol{\psi}_k = \boldsymbol{\psi}_k$$

because $\boldsymbol{\psi}_k(x_1) = \boldsymbol{\psi}_k(x_2)$. And thus, the eigenproblem becomes

$$\tilde{\boldsymbol{P}}\boldsymbol{R}\boldsymbol{\psi}_k = \boldsymbol{R}\boldsymbol{P}\boldsymbol{K}\boldsymbol{R}\boldsymbol{\psi}_k = \boldsymbol{R}\boldsymbol{P}\boldsymbol{\psi}_k = \alpha_k \boldsymbol{R}\boldsymbol{\psi}_k.$$

For an undirected graph, we can use a similar argument to show that $\boldsymbol{\phi}\boldsymbol{K}$ is a left eigenvector of $\tilde{\boldsymbol{P}}$ (where $\boldsymbol{\phi}_k$ is a left eigenvector of $\boldsymbol{P}$). This is true for the first eigenvector, $\boldsymbol{\phi}_0$ (corresponding to the stationary distribution), even if the graph is directed. This shows that combining nodes by summing up all their weights will preserve the overall stationary distribution of the random walk, this is also apparent from equation (2).

In this example we looked at the case when two nodes are combined but the procedure can be expanded to group several nodes at once and/or combining nodes into several individual groups by making appropriate changes to $\boldsymbol{R}$ and $\boldsymbol{K}$ [8]. Of course, in practice two nodes will seldom be exactly identical so the coarse graining will be approximate. How good this approximation is will depend on how similar the nodes are.

To summarize, we have shown that grouping similar nodes together will preserve the eigenvalue $\alpha_k$, average the components of the right eigenvector and (for undirected graphs) sum up the components of the left eigenvector. Some of the information contained in the original graph will be lost but this seems to preserve most of the information we are after.

As a final remark we note that the coarse grained adjacency matrix $\tilde{\boldsymbol{A}}$ is still an adjacency matrix and behaves as we have come to expect. This means that applications which are defined on standard graphs will also work on coarse-grained graphs.

### 3.1.3 Coarse graining the random walk

We will now suggest a way to coarse grain the graph based on node similarity from a random walk point of view. Our idea is to use hierarchical clustering with squared diffusion distance as a dissimilarity measure and average linkage.

To motivate our choice of dissimilarity measure we need to think about what we want to achieve with it. We want to define a metric between nodes such that they are close if their transition probabilities are similar. There are several ways one could do this, such as using the Kullback-Leibler divergence or a simple $L^1$ norm but we choose squared diffusion distance because it has a nice spectral interpretation, as we shall see soon.

The squared diffusion distance between two nodes is the weighted $L^2$ sum [6]

$$D_t^2(x_i, x_j) = \sum_{y \in \Omega} \frac{1}{d(y)} \left( \boldsymbol{P}^t(y|x_i) - \boldsymbol{P}^t(y|x_j) \right)^2 . \tag{9}$$

Where $\boldsymbol{P}^t(y|x_i)$ denotes the probability of going from node $x_i$ to $y$ in exactly $t$ steps. It is worth noting that the transition probabilities from one node always sum up to 1 but the degree can differ between nodes. It is usually the case that nodes in denser regions have more edges than those in sparser regions. So, weighting the terms with the target nodes degree tends to penalize discrepancies in sparser regions more. This makes sense from a random a random walk point of view. Because there are less paths in sparse regions, small changes can have an outsized effect on the overall random walk. In dense regions there will be many alternative paths so changes will have a relatively small impact.

Most nodes that are similar in this sense will be close to each other and thus have a strong edge between them. So, for this to work we will need to include self-loops in our graph. If we do not include self-loops then (9) could punish nodes that are close and have one dominant edge between each other.

Our main motivation behind using squared diffusion distance is that it can be expressed as a weighted Euclidean distance in diffusion space (space spanned by left or right eigenvectors), specifically equation (9) can be expressed as

$$D_t^2(x_i, x_j) = \sum_{k=2}^{n} \alpha_k^{2t} \left( \boldsymbol{\psi}_k(x_i) - \boldsymbol{\psi}_k(x_j) \right)^2 .$$

This follows from the spectral decomposition of $\boldsymbol{P}$, see appendix A for a proof. So, by using equation (9) as a dissimilarity measure in hierarchical clustering we are minimizing a weighted sum of differences between components of right eigenvectors[8] at each merger in the dendrogram. The sum is weighted by the corresponding eigenvalue which means that priority is given to preserving the

[8] These are related to the left eigenvectors, see section 2.2.3

leading eigenpairs, which we know corresponds to large-scale behavior of the random walk. However, we note that the combined sum of several smaller eigenpairs can "overturn" decisions made by the dominant eigenpairs. So this dissimilarity measure will look at preserving the whole spectrum as well as possible, not just the leading eigenpairs.

We can further control the priority of large-scale behavior by setting $t$. As we discussed in section 2.2.3, increasing $t$ corresponds to propagating the local influence of each node with its neighbors. This could be of interest in some situations although it comes at an additional computational cost in computing $\boldsymbol{P}^t$. Out of time considerations, we will restrict ourselves to only study the case when $t = 1$.

Recall that one assumption in HC is that data possesses some hierarchical structure. This seems to fit our intuition for the random walk. If two nodes belong to the same cluster for the best $k$ partitioning of the nodes, then we would also expect them to be in the same cluster when there are $m < k$ clusters. If this is not the case, then the dynamics of the random walk must be different between the two situations.

Note that by using average linkage the node to group dissimilarity is the average distance from a given node to all nodes in the group, and similar for group to group dissimilarity. This means that we always compare distances against the original graph, thus ensuring that they are comparable and that the hierarchical structure assumption holds.

The biggest drawback of this method is that we do not have a way of knowing when to stop coarse graining, other than to reduce the graph until the eigenproblem is solvable.

## 3.2 Out-of-sample extension

In this section we will study a method to extend the embedding from a graphs spectrum to new points without solving the eigenproblem again. The work here originates from a series of articles written by Williams, Seeger & Shawe-Taylor in the early 2000's. They studied the eigenfunctions of a kernel and suggested a way (among other things) to extend eigenvector embeddings to new points. Bengio *et al.* later used this to provide a method for out-of-sample generalization for a wide family of spectral dimensional reduction & clustering methods [3].

Informally, we can interpret their suggested out-of-sample (OOS) extension method in the following way: Instead of saying that we gathered $m$ new data points we assume that we had $N = n + m$ data points from the beginning and projected it down to a $n$-dimensional space such that the spectrum of $\boldsymbol{A}$ represent the best $n$-rank approximation of $\boldsymbol{M}$. Where $\boldsymbol{M} \in \mathbb{R}^{N \times N}$ is the matrix containing all $N$ data points. So, under this interpretation what we want to

do when "adding back" the $m$ points is to approximate their coordinates in the reduced space. That is, find an approximation for their projection from the eigenspace of $\boldsymbol{M}$ to the eigenspace of $\boldsymbol{A}$.

To accomplish this, we reinterpret data as being output from a kernel. A kernel is a type of function that lets us compute the dot product between two vectors in some (possible high dimensional) feature space without explicit knowledge about that space [17]. That such a space exists was proven by Aronszajn in [1], under the assumption that the kernel is positive semi-definite (PSD)[9] and symmetric. Here, we will not actually compute anything new with the kernel. Our data is just reinterpreted to be output from some kernel, that is $K(x_i, x_j) = \boldsymbol{A_{ij}}$. Note that $\boldsymbol{A}$ is symmetric and only has non-negative elements, so we know that there exists an appropriate kernel. And because the kernel is a function, we also reinterpret the eigenvector problem as an *eigenfunction* problem.

The eigenfunctions of a linear operator $D$, defined on some function space, is any non-zero function $f$ in that space such that $f$ acted upon by $D$ will only multiply $f$ by some scaling factor, $\lambda'$, called an eigenvalue, i.e.,

$$(Df)(x) = \lambda' f(x).$$

For all practical purposes here, we can just interpret eigenfunctions as eigenvectors. However, we note that, because the underlying space could be infinite dimensional, there could be an infinite number of eigenfunctions. Our linear operator will be the kernel $K$ and

$$(Kf)(x) \stackrel{\text{def}}{=} \int K(x, y) f(y) p(y) dy$$

where $p(y)$ is a density function.

Using this one can formulate the following proposition.

**Proposition 2 (Out-of-sample extension [3])** *Let $K(x, y)$ be a kernel function giving rise to the symmetric kernel matrix $\boldsymbol{K}$ with entries $\boldsymbol{K}_{ij} = K(x_i, x_j) \geq 0$ upon a dataset $D = \{x_1, ..., x_n\}$. Let $(v_k, \lambda_k)$ be the eigenvector/value pair that solves $\boldsymbol{K}v_k = \lambda_k v_k$ and let $(f_k, \lambda'_k)$ be the eigenfunction/value pair that solves $(Kf_k)(x) = \lambda' f_k(x)$ for any $x$. Then*

$$f_k(x) = \frac{\sqrt{n}}{\lambda_k} \sum_{i=1}^{n} v_k(x_i) K(x, x_i). \tag{10}$$

*Proof* We first note that the exact eigenfunction problem is

$$\int K(x_i, x) f_k(x_i) p(x_i) dx_i = \lambda' f_k(x). \tag{11}$$

[9] An $n \times n$ matrix, $\boldsymbol{M}$, is said to be PSD if $x^T \boldsymbol{M} x \geq 0$ for all $x \in \mathbb{R}^n$ [11].

But because we do not know $p$ we approximate the problem using the sample distribution

$$\frac{1}{n}\sum_{i=1}^{n} K(x, x_i) f_k(x_i) = \lambda'_k f_k(x). \tag{12}$$

This is not directly solvable either. However, if $x$ was one of the original data points, for which the spectrum has already been calculated, we know that

$$\sum_{i=1}^{n} K(x_j, x_i) v_k(x_i) = \lambda_k v_k(x_j).$$

Comparing this with equation (12) when $x = x_j \in D$ directly gives $\lambda'_k = \frac{1}{n}\lambda_k$ and $f_k(x_j) = \sqrt{n} v_k(x_j)$. If we enforce this similarity between eigenfunctions and eigenvectors then we will be able to solve equation (12) by inserting $f_k(x_j) = \sqrt{n} v_k(x_j)$.

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n} K(x, x_i) f_k(x_i) = \frac{\sqrt{n}}{n}\sum_{i=1}^{n} K(x, x_i) v_k(x_i) = \lambda'_k f_k(x) \\ \iff f_k(x) = \frac{\sqrt{n}}{\lambda_k}\sum_{i=1}^{n} v_k(x_i) K(x, x_i).\end{aligned}$$

Q.E.D.

Equation (10) is known as the Nyström formula and has been used to speed up various numerical methods since at least 1977 [2]. It can (and has) been used to approximate eigenvectors and speed up kernel machines [20]. The computational cost of this is linear in $n$ for each $f_k(x)$ we compute. It has also been noted that equation (10) is proportional to the kernel PCA projection formula [19]. As for the theoretical justification. It is possible to show that (12) converges to (11) when $n \to \infty$, even under the limited assumptions made here [2]. It is also possible to put bounds on the convergence error [15]. This means that, as $n \to \infty$, we can expect each eigenvector to converge to an eigenfunction for the kernel $K$ (up to a normalization factor).

An alternative way of deriving equation (10), which I think is more informative, is by minimizing

$$\sum_{i=1}^{n}\left(K(x, x_i) - \sum_{t=1}^{n'} \lambda'_t f_t(x) f_t(x_i)\right)^2 \tag{13}$$

w.r.t. $f_k(x)$ when $f_k(x_i) = \sqrt{n} v_k(x_i)$ and $n' \leq n$. The proof is a little bit tedious but trivial since this is a convex optimization problem[10]. So, we will skip the proof and focus on the intuition.

[10] Just set the derivative w.r.t. $f_k(x)$ to zero and solve for $f_k(x)$.

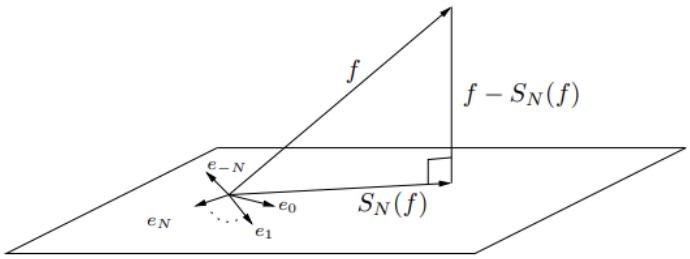



Figure 5: Orthogonal projection of a function $f$ down to the space spanned by first $N$ eigenfunctions. Where $S_N(f) = \sum_{t=1}^{n'} \lambda'_t f_t(x) f_t(x_i)$. Figure taken from [17].

A standard result in Fourier analysis is that the best approximation of $K(x, y)$ (in the sense of minimizing expected square error) while using only $n'$ terms is the eigenfunction-decomposition using the $n'$ first eigenfunction-pairs [17], i.e.

$$K(x, y) \approx \sum_{t=1}^{n'} \lambda'_t f_t(x) f_t(x_i).$$

So with the optimization problem in (13) we are saying that the eigenfunctions we have already computed is the best approximation of $K(x, y)$ and try to find the remaining, unknown terms. The geometrical interpretation of this is that $f_k(x)$ is the orthogonal projection of the full function $f$ in the plane spanned by the first $n'$ eigenfunctions. This is illustrated in Figure 5 (which is taken from [17]).

The time complexity of equation (10) is linear in training data, i.e., $O(n)$. But that is only for one eigenfunction. If we want to embed a new point $x$ into the eigenspace of $\boldsymbol{A}$ we need to compute equation (10) for all $n$ eigenfunctions, which makes the cost $O(n^2)$.

# 4 Results

In this section we will test the coarse graining and OOS extension methods on two data sets, described below. We will use the R programming language for these tests. The eigenvalues/vectors are always computed using the built in "eigen" function, which computes the eigenpairs down to computer precision.

The tests here will be carried out on the normalized Laplacian, $\boldsymbol{\mathcal{L}}$. Mostly because the OOS-extension method requires this and we do not want to mix between subsections. Since $\boldsymbol{\mathcal{L}}$ is symmetric we do not have to distinguish between left and right eigenvectors.

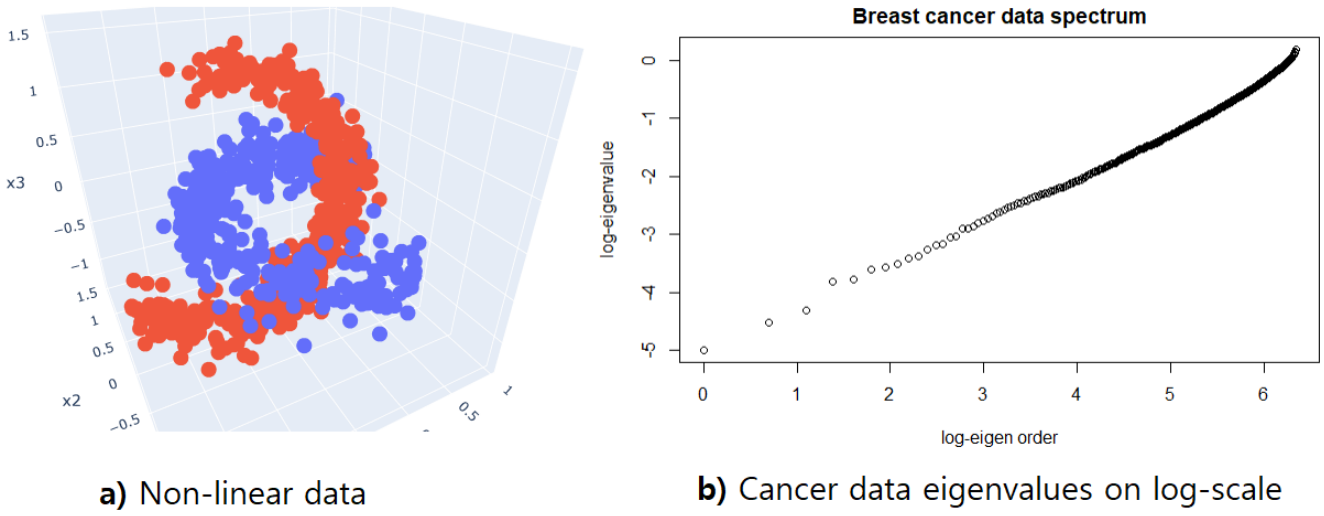



Figure 6: a) Non-linear (NL) data. Two arches in a 3-dimensional space (coloring is just for orientation). b) log-transformed eigenvalues of the breast cancer data.

## 4.1 Preliminaries

### 4.1.1 Data

Throughout this section we will use two data sets. A non-linear (NL) data set and a breast cancer data set from the UCI ML repository [7].

The NL data is displayed in Figure 6a). This data consist of 2 arches, each containing 400 points, in a 3-dimensional space. This data set was chosen for its simplicity and friendly properties. Specifically, each of the two arches are essentially 1-dimensional structures and the data is dense, as we can see in the figure. Because of this we can expect there to be a lot of redundancies in the spectrum of this data set. Unfortunately, the data is a little bit "too dense". This results in some extremely small eigenvalues and eigenvector components which might causes numerical instabilities. This is not an insurmountable problem (perhaps not even a problem at all) but it is easier to just add some noise and ignore it. To this end, we will add 3 more dimensions to the data, each generated from a Gaussian distribution with mean 0 and standard deviation 1. This will make the second smallest eigenvalue about 15 times larger, the largest eigenvalues are roughly the same.

The breast cancer data's eigenvalues on the log-scale are displayed in Figure 6b). As we can see, they look linear. This indicates that the spectrum of the graph Laplacian follows a power law [14] with slope parameter less than 2. The

power law, $f(x) \propto \lambda^{-k}$, have some interesting characteristics. It only has a well defined mean over $x \in [1, \infty)$ if $k > 2$ and finite variance if $k > 3$. Not having a finite variance means that the eigenvalue sum diverges as $n \to \infty$. If we think of the eigenvalues as capturing information in one direction of space then this would translate into "an infinite amount of information not being included in any finite sample size". Graphs like these arise naturally in many large real-world graphs/networks, like the world-wide-web [14] (although most often with slope $2 < k < 3$ , thus having a well defined mean but not variance).

### 4.1.2 Graph construction

From the raw data we construct the adjacency matrix in the following way. Data is first standardized so that each variable have mean 0 and unit variance. We then compute the pairwise Euclidean distance between points and find their $k$-nearest neighbors (kNN). These neighbors will be the edges from the node and each edge will have weight

$$w_{ij} = \frac{1}{1 + ||x_i - x_j||^2}.$$

Where $||x_i - x_j||^2$ is the Euclidean distance between node $x_i$ and $x_j$. One can calculate the weights in different ways. The perhaps most popular way is by using Gaussian weights, i.e. $w_{ij} = \exp\left(\frac{-||x_i - x_j||^2}{2\sigma^2}\right)$, but Gaussian weights require us to set the scale parameter $\sigma$ [18]. We chose to use the above defined weights mainly to avoid having to set this parameter.

Note that some nodes can have more than $k$ edges. This is because $x_i$ can be among $x_j$'s $k$NN but not the other way around and we will still connect them with an undirected edge. Also note that we have not excluded self-loops (which is usually the convention). Obviously, the Euclidean distance from a point to itself is 0, so all self-loops have weight 1, i.e., $w_{ii} = 1$.

This is just one out of several possible ways to construct a graph from data, see [18, section 2.2] for a concise explanation of some of the most popular graph construction methods. Different construction methods will give graphs with slightly different sets of edges and thus different adjacency matrices. This will naturally result in somewhat different spectrum's but this is not of any particular importance in this thesis. We are just interested of approximating a graph's spectrum, no matter what the adjacency matrix looks like.

### 4.1.3 Distance validation

We can think of both coarse graining and OOS-extension as dimensional reduction techniques. We are interested in knowing how well the reduced graph approximate the full graph. One way to gauge this is to look at rank distances between the two graphs.

Rank distances converts distances to rank orderings where the closest point have rank 1 and the point furthest away rank $n$. These methods are used as we do not care about absolute differences in distances, only relative differences. That is, if two nodes are a given distance away from each other in the full graph then the distance should be comparable in the reduced graph, relative to all other nodes.

We are going to use two rank correlation methods for this, Spearman's rho and Kendall's tau. Intuitively, they both measure the degree of similarity between two rankings (although in a slightly different way) and can therefore be used to assess the significance between them.

We first note that there are $\binom{n}{2} = \frac{n(n-1)}{2}$ ways of ordering $n$ objects. So, in order for a rank method to be comparable between data sets, we should normalize by this coefficient [12].

Let $r_i$ and $s_i$ be the rankings of the $i$:th member of the two vectors $x$ and $y$ respectively and let $R(x)$ denote the rank ordering of $x$. Spearman's rho is defined similarly as Pearson correlation coefficient but between rank variables. Simple but long and tedious calculations give [12]

$$\rho = \frac{\operatorname{cov}\big(R(x), R(y)\big)}{\sigma_{R(x)}\sigma_{R(y)}} = \cdots = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)}.$$

Where $d_i$ is the rank difference between $r_i$ and $s_i$.

Kendall's tau count the number *concordant* and *discordant* pairs. Any pair of observation rankings $(r_i, s_i)$ and $(r_j, s_j)$ are said to be concordant if their order agrees. That is, if both $r_i > r_j$ and $s_i > s_j$ holds **or** if both $r_i < r_j$ and $s_i < s_j$. If this is not the case, then the pair is discordant. One can calculate this as [12]

$$\begin{aligned}\tau &= \frac{2}{n(n-1)}\,(\#\text{ concordant pairs} - \#\text{ discordant pairs}) \\ &= \ldots = \frac{2}{n(n-1)}\sum_{i<j}\operatorname{sgn}(r_i - r_j)\operatorname{sgn}(s_i - s_j).\end{aligned}$$

where # reads as "number of".

Both $\rho$ and $\tau$ takes values between $-1$ and 1. With 1 indicating perfect rank correlation, 0 no correlation and $-1$ complete rank distortion. These two measures are in fact both special cases of a more general correlation coefficient [12].

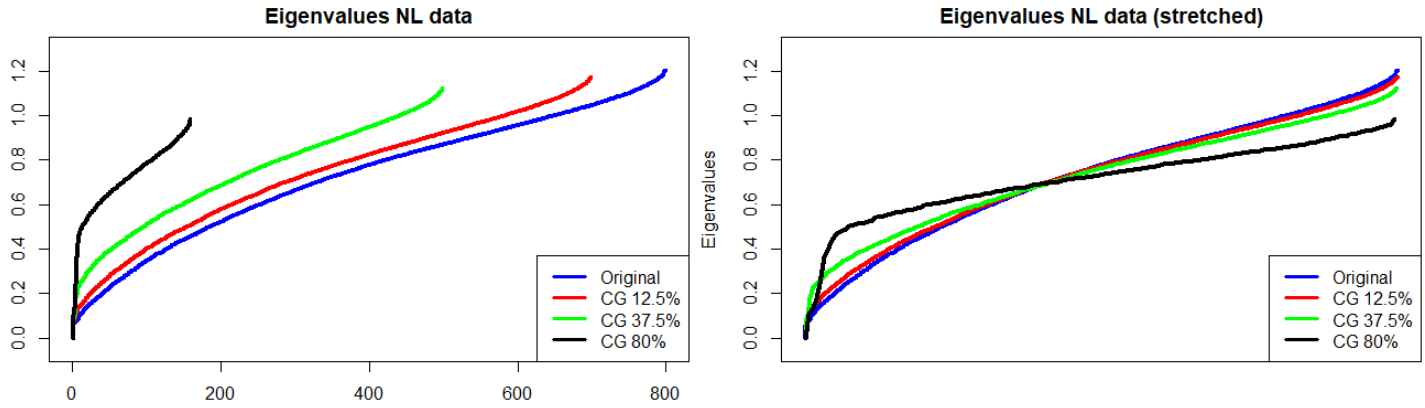



Figure 7: (left) Eigenvalue ordered by size for the NL data for the original graph (blue line) and a few coarse graining graphs. (right) Same figure but with stretched lines for relative difference.

## 4.2 Coarse graining

We begin by looking at how well the set of eigenvalues is preserved for a few CG depths. The eigenvalues for the non-linear (NL) data is displayed in Figure 7 (left). The blue line is the original graph while the red, green and black lines are eigenvalues for CG graphs of different depths. The graph behind the red line have had 12.5% of its nodes combined. For the green and black line that figure is 37.5% and 80% respectively. In Figure 7 (right) we stretch the CG lines so that they have equal length as the blue line. This is done to illustrate which part of the eigenvalue set that gets reduced. We can see that the more nodes we combine the bigger difference in eigenvalus. It is worth noting that when only a relative few nodes are merged we still get fairly good results with the biggest difference being for the first few eigenvalues.

Figure 8 displays the CG eigenvalues for the cancer data. The slope parameter changes somewhat but they still seem to follow the power law. Since data is scale-free, decreasing the number of data points by CG will only reduce the information captured in the eigenvalues (roughly) linearly.

We might also be interested in knowing how individual eigenvalues change as we CG the graph. And in particular how the first non-trivial eigenvalue $\lambda_2$ changes. Recall that, by Cheeger's inequality, $\lambda_2$ tells us how connected the graph is and gives bounds on the stationary distribution convergence time. So, if $\lambda_2$ changes too much we have good reason to believe that the random walk, and therefore also the spectrum, have changed significantly.

Figure 9a) and 10a) displays $\lambda_2$ against CG depth (how many nodes have been

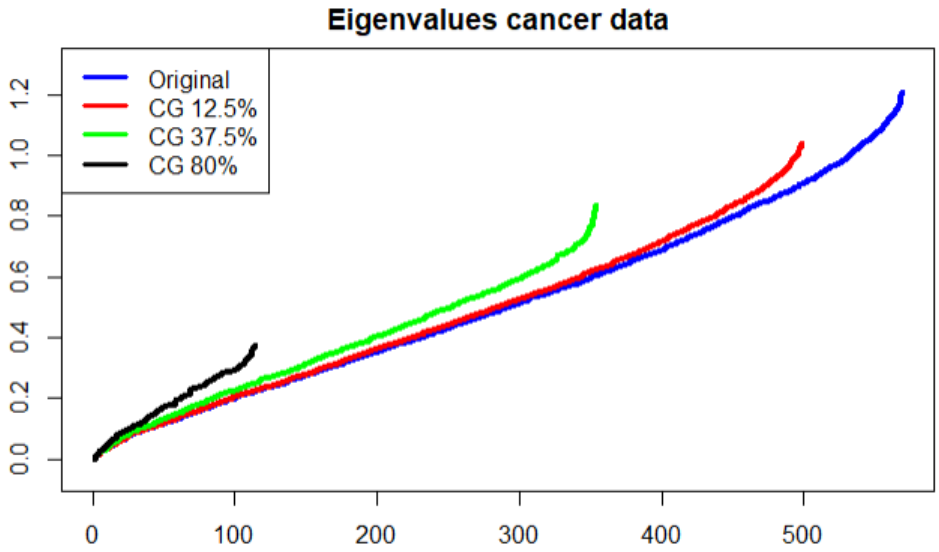



Figure 8: Eigenvalues ordered by size for the cancer data set for the original graph (blue line) and a few coarse grained graphs.

merged), the depth is down to 80% of the data (640/800 for the NL data and 455/569 for the cancer data). We can see from both figures that $\lambda_2$ values tend to group up and then "jump" to form a gap. This is also true for most other eigenvalues, although less pronounced, see Figure 9b) and 10b). This could mean that, as we combine nodes, the random walk will change (relatively) little up to some critical point where it undergoes a larger change. Unfortunately, we do not have time to study this further. But we will say that just before the first such gap would be a natural candidate for a place to stop CG. The first few eigenvalues can be computed efficiently using the Arnoldi method [9] but this is still likely too slow practically. So, it is probably not usable in practice.

As a final measure we will test how well distances are preserved on the CG graph. If the CG graph is well preserved then pairwise distances should be comparable to the full graph. With comparable we mean similar in a relative terms. If two nodes are close on the original graph then the distance might be longer or shorter on the CG graph in absolute terms but should be comparable relative to all other nodes. We will test this using Spearman's rho and Kendall's tau on the biharmonic distance. The biharmonic distance is given by [13]

$$C_{ij} = \mathrm{Vol}(G) \sum_{k=2}^{n} \frac{1}{\lambda_k^2} \left( \frac{v_k(x_i)}{\sqrt{d_i}} - \frac{v_k(x_j)}{\sqrt{d_j}} \right)^2 .$$

Where $d_i$ is the degree of node $i$. There is no direct geometric interpretation for this distance but it is similar to the commute time distance (CTD). In CTD the

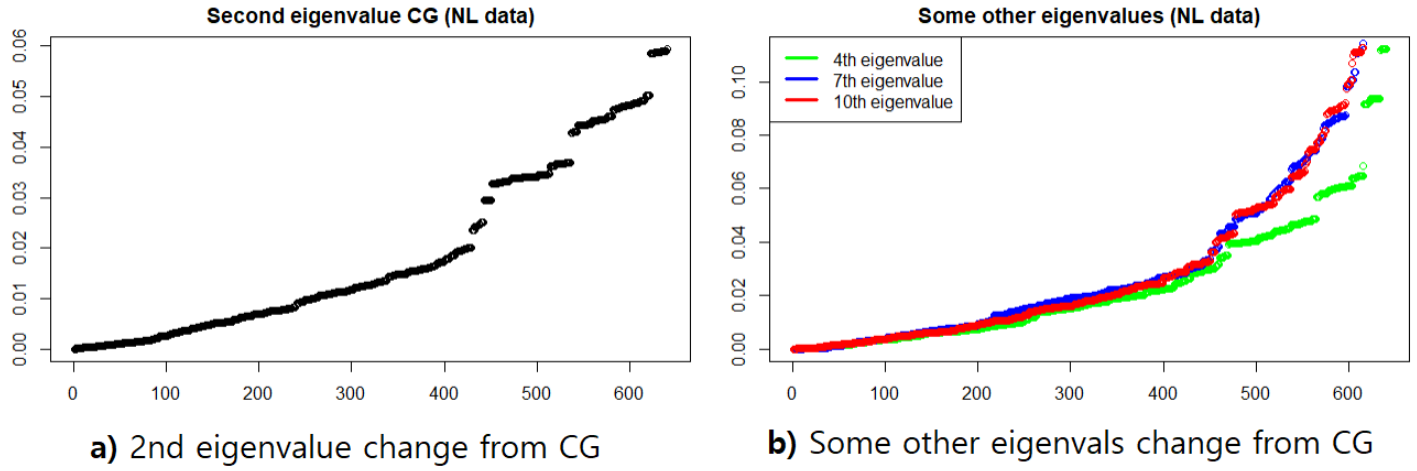



Figure 9: Eigenvalue gap for the non-linear data. a) Second eigenvalue. b) 4th (green), 7th (blue) and 10th (red) eigenvalue. x-axis represent the number of nodes merged.

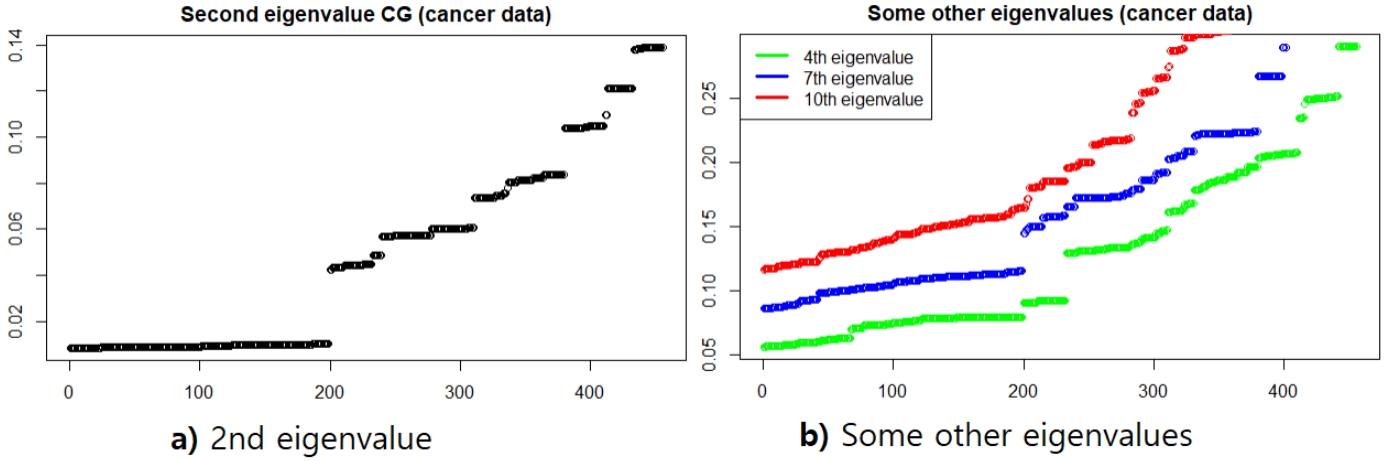



Figure 10: Eigenvalue gap for the cancer data. a) Second eigenvalue. b) 4th (green), 7th (blue) and 10th (red) eigenvalue. x-axis represent the number of nodes merged.

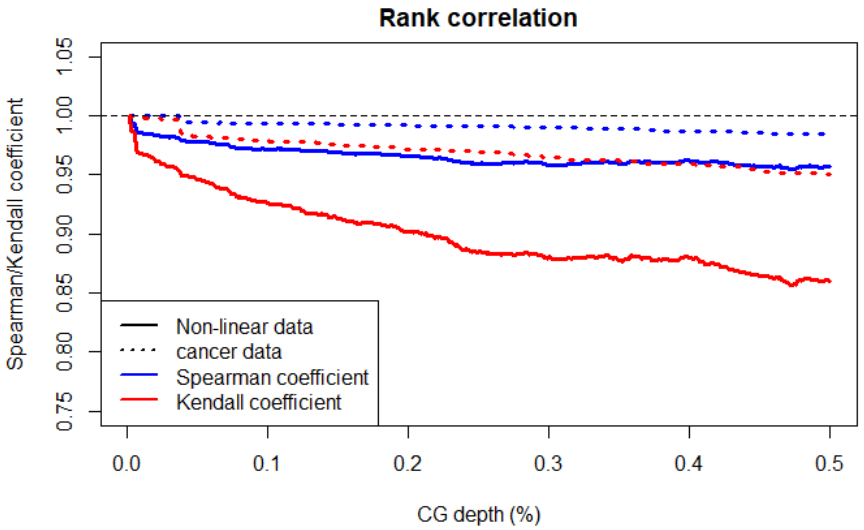



Figure 11: Spearman's rho (blue) and Kendall's tau (red) for the NL-data (solid lines) and cancer data (dotted lines). Y-axis start at 0.75.

eigenvalue in the denominator is not squared, i.e., $1/\lambda_k$ weightings, otherwise they are the same. CTD measures the expected time it takes a random walk to go from $x_i$ to $x_j$ **and** back. The different weighting in the biharmonic distance have been shown to increase stability when estimating larger distances [13].

One can only look at rank correlation when comparing distance vectors of equal length. Since the CG graph have less nodes there is obviously going to be less pairwise distances. Therefore, when the distance on the CG graph goes to a group (a node that have been merged) we will compare it to the average distance of all nodes in that group on the full graph. The result is displayed in Figure 11. The solid line is for the NL-data and dotted lines are for cancer data, blue indicates Spearman's rho and red Kendall's tau.

From Figure 11 we can see that rank distances are preserved fairly well for both data sets but noticeably better for the cancer data. This is perhaps a little bit surprising as the cancer data follows the power law[11] and the NL data was purposefully chosen to have a lot of redundancies. So, one would perhaps think it should be the other way around. However, as we saw in Figure 8, the eigenvalues of a power law are proportional to each other and this still holdes for the CG power law. So there should only be a (roughly) linear change in graph distances. Thus, good rank preservation.

[11] Thus an "infinite amount of information" is not included in any finite sample.

We also observed from Figure 11 that the Spearman's rho indicates a better rank correlation than Kendall's tau. This means that CG is better at capturing linear rank changes. That is, when rank distances increase/decrease on the full graph, they tend to do so on the CG graph as well.

## 4.3 Out-of-sample extension

We will now look into the OOS-extension method. Before we can start we need to consider what the kernel looks like. Recall the formula for the extension is (from equation (10))

$$f_k(x) = \frac{\sqrt{n}}{\lambda_k} \sum_{i=1}^{n} v_k(x_i) K(x, x_i).$$

Now, as previously mentioned, we will not actually compute anything with the kernel. We just say that "data comes from some kernel". This means that each kernel term will just be a term from the Laplacian, i.e., $K(x_i, x_j) = \boldsymbol{\mathcal{L}}_{ij}$. However, we still need to compute this Laplacian. We construct the adjacency matrix according to the procedure in section 4.1.2. Naturally, if we include the test points when we construct the adjacency matrix we will get a different result then if we do not. So, we will make the choice not to include them in the original graph and instead add these nodes as if we "found more data". That means, connecting these "new points" with their kNN but not changing any other edge in the adjacency matrix. So, we do not recompute the entire graph. Both test points and removed data is chosen randomly. Which points are chosen/removed will effect the final result so we will redo the test several times (10 times in each situation) and compute standard error (SE) bars.

To test how well the spectral embedding works for "new points" we will use the same rank distance procedure as we used in the CG section (with biharmonic distance). Specifically, rank correlation will be computed between the distances in the original graph (where the spectrum have been computed with the entire data set) and those in the reduced graph after we have extended the embedding. We note that, unlike in the CG case, each pairwise distance is represented in both cases but the distance will be exact for the original graph and an approximate projection for the reduced graph.

Our first observation is that just removing 1 point will generally give a good rank correlation, generally over 0.95 for both Spearman's rho and Kendall's tau. Furthermore, the standard error is fairly small, often the difference between upper and lower bound is under 0.03. But as more points are removed the mean rank correlation decreases and the standard error increases, on average. This change is not linear and happens fastest in the beginning when only a relative small number of points have been removed. We once again observe that Spearman's rho indicates a better rank correlation than Kendall's tau.

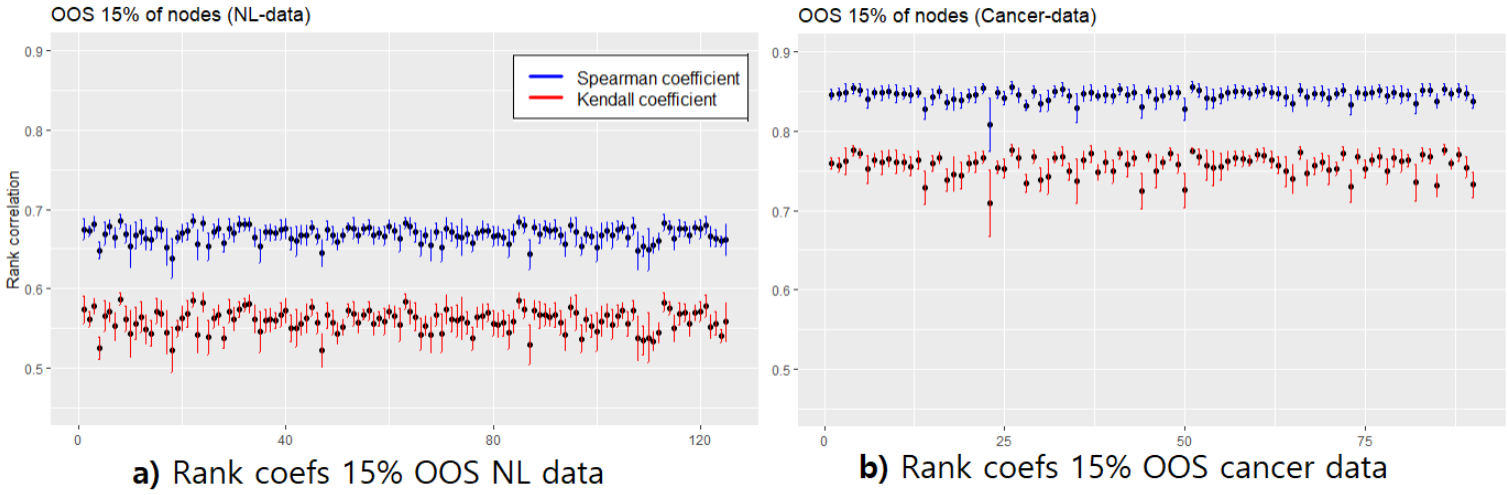



Figure 12: a) Rank correlation for the NL-data when extending to 15% of the total data points with standard error bars. The blue bars represent Spearman's rho and the red Kendall's tau. Each bar is 1 OOS-extended data point. x-axis are the number of data points added but they are all added independently. b) Same but for the cancer data.

Figure 12 shows the result when 15% of the data is removed (for the NL-data in Figure 12a and cancer data in 12b). Each SE-bar represents one OOS-extended data point, the black dot is the mean value. Note that the bars are independent of each other. That is, when we have extended the spectrum for one point, $x_{n+1}$, we do not use the extended spectrum when extending for the next point, $x_{n+2}$. That would result in multiplicative error. In Figure 12 we can see that rank correlation is similar for all points but different points seems to give slightly different results. Further studies could look into if this is caused by randomness or if some points (perhaps those in denser regions) give better embeddings. We also note that the cancer data have better rank correlation, this is likely because of the same reason we discussed for CG but "in reverse".

Recall that one of the motivations behind using this formula is that equation (12) converges to equation (11) as $n \to \infty$. And specifically that $\lambda'_k \to \frac{1}{n}\lambda_k$ as $n \to \infty$. Of course, in practice this will never have infinite data and this will always be an approximation. Williams and Seeger show in [19] that this approximation is better if $\lambda_k$ is large (for fixed $n$), Shawe-Taylor and Williams gave bounds for this in [15]. The implications of this for us here is that if the eigenvalues are smaller, generally speaking, then the OOS-extension will be worse.

It is hard to directly test this since the approximation statement is for eigen-

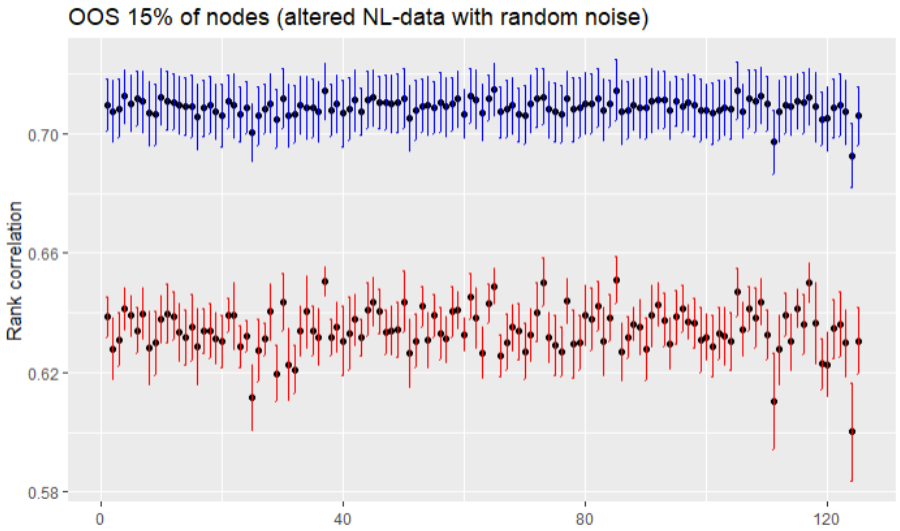



Figure 13: Same test as in Figure 12a but with added random Gaussian noise to the NL-data. Blue bars represent Spearman's rho and red Kendall's tau.

values of the same spectrum and the OOS-extension method requires the whole spectrum (equation (10) for all $k = 1, ..., n$). But perhaps we can indirectly test it using similar data sets with similar spectrums. Specifically, by adding a dimension of random Gaussian noise to the data (as we did for the NL-data set, see section 4.1.1). Adding random noise like this decreases the edge weights, as the local Euclidean distance used to construct the graph increases (see the graph construction section). This means that the graph is less connected and thus, by Cheeger's inequality, larger eigenvalues.

Figure 13 shows the rank correlations for the NL-data when an additional 5 dimensions of Gaussian noise have been added. The rank correlations are now computed against this extended NL-data so they are not directly comparable to the result in Figure 12a but we note that the rank correlations are better with the noisy NL-data. The other properties discussed above still holds, just with slightly better rank correlation at each step. One can show even better rank correlation by adding more noise. Similar result holds for the cancer data. This shows that OOS-extension works better when the smallest eigenvalues are larger. However, we will point out that this noisy data is a worse representation of the true manifold (from which we assume data is sampled) so it does not follow that adding noise improves the overall result. In fact, it probably does not. But we can be more comfortable using this method if the graph already have large eigenvalues to begin with.

# 5 Discussion

In this thesis we have given an introduction to spectral graph theory and examined ways to reduce the time complexity of the eigenproblem. We did this using two methods: 1) Coarse graining (CG), where we reduced the graph by combining nodes based on a random walk on the graph, and 2) Out-of-sample (OOS) extension, where we tried to find an embedding for new nodes in a spectrum that we had already computed. Both methods have some merits and we can envision them being used for different purposes.

One advantage with the OOS-extension method is that it is computationally cheaper, especially if we do not need to compute $f_k$ for all $k = 1, ..., n$. For example, in spectral clustering one only uses a few leading eigenvectors. It would then be possible to just compute a few eigenpairs using an iterative method and then extend them to new points. But even if the full embedding is computed OOS will still be faster than CG, under direct comparison. If the full data set consist of $n = k + m$ data ponints then the dominant terms in the time complexity will be at best $O(kn^2 + k^3)$ for CG but "only" $O(k^3 + mk^2) = O(kn)$ for OOS-extension (assuming that the matrix is dense and that the whole spectrum is computed).

This might be reason enough to use OOS-extension over CG for applications in spectral clustering or dimensional reduction. However, CG essentially just combines nodes. The end result is still a proper graph, the same is not the case for OOS-extension. Therefore, any algorithm that works on the input graph will also work on the CG graph but not necessarily on the OOS-extension.

Other things to consider is the underlying assumptions made for each method. CG requires self-loops and this might not always be justifiable. We do not need the self-loop assumption in the OOS method but we need the matrix that represents the graph (adjacency matrix or some Laplacian) to be symmetric and (probably[12]) positive semi-definite.

[12] Bengio *et al.* [3] claims that the PSD assumption can be dropped but I'm not completely convinced as I think that assumption is needed to justify using results from function analysis in the first place.

# References


[1] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, Vol. 68(3), 1950.

[2] Christopher T. H. Baker. *The numerical treatment of integral equations.* Clarendon Press, Oxford, 1977.

[3] Yoshua Bengio, Jean-françcois Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, 2003.

[4] Fan Chung. *Spectral graph theory.* Conference Board of the Mathematical Sciences, Washington, 1997.

[5] Dragoš Cvetković and Slobodan Simić. Graph spectra in computer science. *Linear Algebra and its Applications*, vol. 434(6), 2011.

[6] Ben. Hareman Willy. van der Walt Stéfan de la Porte, J. Herbst. An introduction to diffusion maps. In *Conference: Proceedings of the 19th Symposium of the Pattern Recognition Association of South Africa (PRASA 2008)*, 2008.

[7] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[8] David Gfeller and Paolo De Los Rios. Spectral coarse graining of complex networks. *Physical review letters*, 99, june 2007.

[9] Gene H. Golub. *Matrix Computations.* Johns Hopkins University Press, 4 edition, 2013.

[10] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[11] Roger Horn and Charles R. Johnson. *Matrix Analysis.* Cambridge University Press, 2 edition, 2013.

[12] Maurice Kendall. *Rank Correlation Methods.* Charles Griffin and Company, 4 edition, 1970.

[13] Yaron Lipman, Raif Rustamov, and Thomas Funkhouser. Biharmonic distance. *ACM Transactions on Graphics*, 29(3), June 2010.

[14] Milena Mihail and Christos Papadimitriou. On the eigenvalue power law. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, 2002.

[15] John Shawe-Taylor and Christopher Williams. The stability of kernel principal components analysis and its relation to the process eigenspectrum. In *Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference*, 2002.

[16] Daniel A. Spielman. Spectral and algebraic graph theory, 2019. Incomplete Draft, dated December 4, 2019.

[17] Elias M Stein and Rami Shakarchi. *Fourier Analysis: An Introduction*. Princeton University Press, 2003.

[18] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, vol. 17, 2007.

[19] Christopher Williams and Matthias Seeger. The effect of the input density distribution on kernel-based. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[20] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, 2000.

# Appendix

## Diffusion distance

The squared diffusion distance can be written as

$$D_t^2(x_i, x_j) = \sum_{y\in\Omega} \frac{1}{d(y)} \left(\boldsymbol{P}^t(y|x_i) - \boldsymbol{P}^t(y|x_j)\right)^2 = \sum_{k=2}^{n} \alpha_k^{2t} \left(\boldsymbol{\psi}_k(x_i) - \boldsymbol{\psi}_k(x_j)\right)^2$$

*Proof.* From the result in section 2.2.3 we know that the spectral decomposition of $\boldsymbol{P}^t(x_i, x_j)$ is given by

$$\boldsymbol{P}_{ij}^t = \sum_{k=1}^{n} \alpha_k^t \boldsymbol{\psi}_k(x_j)\boldsymbol{\phi}_k(x_i).$$

If we insert this into $D_t^2(x_i, x_j)$ we get

$$\begin{aligned}
D_t^2(x_i, x_j) &= \sum_{y\in\Omega} \frac{1}{d(y)} \left(\sum_{k=1}^{n} \alpha_k^t \boldsymbol{\psi}_k(x_i)\boldsymbol{\phi}_k(y) - \sum_{r=1}^{n} \alpha_r^t \boldsymbol{\psi}_r(x_j)\boldsymbol{\phi}_r(y)\right)^2 \\
&= \sum_{y\in\Omega} \sum_{k,r=1}^{n} \alpha_k^t \alpha_r^t \big(\boldsymbol{\psi}_k(x_i) - \boldsymbol{\psi}_k(x_j)\big)\big(\boldsymbol{\psi}_r(x_i) - \boldsymbol{\psi}_r(x_j)\big) \frac{\boldsymbol{\phi}_l(y)\boldsymbol{\phi}_r(y)}{d(y)} \\
&= \sum_{k,r=1}^{n} \alpha_k^t \alpha_r^t \big(\boldsymbol{\psi}_k(x_i) - \boldsymbol{\psi}_k(x_j)\big)\big(\boldsymbol{\psi}_r(x_i) - \boldsymbol{\psi}_r(x_j)\big)\delta_{kr} \\
&= \sum_{k=2}^{n} \alpha_k^{2t} \left(\boldsymbol{\psi}_k(x_i) - \boldsymbol{\psi}_k(x_j)\right)^2
\end{aligned}$$

The $k = 1$ term is 0 as the first right eigenvector is $\mathbb{1}$. On the second line we used $\boldsymbol{\psi}_k = \boldsymbol{D}^{1/2} v_k$ and the fact that $v_k$ is orthogonal (see section 2.2.3).