Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Secure spectral clustering: the approximation of eigenvectors in the integer domain

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Marie-Louise Christina Steverink

Delft, the Netherlands
August 2017

MSc THESIS APPLIED MATHEMATICS

"Secure spectral clustering: the approximation of eigenvectors in the integer domain"

MARIE-LOUISE CHRISTINA STEVERINK

Delft University of Technology

August, 2017 Delft

Summary

The upswing of big data and cloud storage services brings along a myriad of possibilities to compare data points on a large scale. However, privacy concerns may limit the applications of outsourced data mining. These arguments have motivated research in privacy preserving data mining, in which multiple parties interact to evaluate a function without obtaining information about any other party's input. One of the cryptographic tools to make a function privacy preserving is Paillier encryption, which allows linear operations to be performed on encrypted values. This thesis focuses on a data mining technique called spectral clustering, which groups data points according to a measure of connectivity in a data graph. A pivotal part of spectral clustering is the partial eigendecomposition of the graph Laplacian. Two numerical algorithms are used to approximate the eigenvectors of the Laplacian: the Lanczos algorithm and the QR algorithm. In this thesis, these numerical methods are adapted to work on Paillier encrypted data values. The main challenge is the fact that Paillier encryption can only be applied to a field of positive integers. Furthermore, cryptographic protocols have to be invoked to perform non-linear operations. Also, the square root and division operations are computationally heavy in the privacy preserving domain. The numerical algorithms are adapted to overcome these challenges and be more suitable to work on encrypted values. Moreover, designs were given for the privacy preserving approximation of eigenvectors with the use of cryptographic protocols between two parties. The convergence and the accuracy of the adapted algorithms were investigated, along with the influence of the adaptation on the performance of the spectral clustering algorithm for datasets with two, five and ten clusters. This research shows that the numerical algorithms approximate the eigenvalues with high accuracy for all the datasets. For two and five clusters, the eigenvectors are approximated accurately and the spectral clustering algorithm performs well. For ten clusters, the Lanczos algorithm fails in the reconstruction of the eigenvectors and therefore the spectral clustering algorithm cannot be performed.

Acknowledgements

This thesis was submitted as the final requirement for the Applied Mathematics Master of Science degree at the Delft University of Technology. I owe a debt of gratitude to many people. First of all, I would like to thank my supervisors at the TU Delft and at TNO. Martin van Gijzen, thank you for your enthusiasm and positive attitude, even during our meetings at 5 PM. One does not encounter your great combination of friendliness and originality so often. I also want to thank Thijs Veugen for giving me the opportunity to write my thesis at TNO in a notoriously difficult project. I was always impressed by your detailed knowledge and your patience with a beginning learner in cryptography. Moreover, I would like to thank Zekeriya Erkin and Kees Vuik for taking the time and effort to be part of my graduation committee.

I would also like to show gratitude to a few people who may not have contributed to this thesis directly, but without whom it would have been difficult to find the perseverance and spirit that was required for the process. I want to thank the interns at TNO for making me feel like a part of a group and Cindy and Saskia for the warm welcome when I came to Delft. Eva and Marie Beth, thank you for sharing the joys and sorrows of a mathematics thesis and understanding how many "teamschotels" it required. Thank you Eva, Hanne and Shila for providing a perfect home in which the kitchen always smells of something delicious. A big thanks to my lovely girlfriend Evita for the dreams that we share and for being arguably the most patient person in the world. Finally, I thank my dear parents Paul and Maja and my brother Lars for their endless love and trust in me. You taught me more than writing a thesis ever will.

Lisa Steverink
Amsterdam, August 2017

Contents

Nomenclature

Note that some symbols have multiple definitions. The context will clarify the use of the symbol. Furthermore, in this thesis, a column of a matrix is denoted by a lowercase letter, e.g. the jth column of T is denoted by t_j.

(N, g)	Public key for Paillier encryption
$[m]$	Encrypted message/ciphertext
α_i	Diagonal entries of Ritz matrix T
β_i	Lower diagonal entries of Ritz matrix T
ℓ_c	Bit length of the ciphertext space
ℓ_N	Bit length of the message space
γ_i	Upper diagonal entries of Ritz matrix T
κ	Security parameter
λ	Secret key for Paillier encryption
λ_i	The ith smallest eigenvalue of L
ϕ	Fixed point arithmetic map
ψ	Map to encode signed integers
σ	Gaussian similarity parameter
θ_i	Ritz values
$\tilde{U}$	Approximated eigenvectors of L, also called Ritz vectors
A	Adjacency matrix
a_{ij}	Weight on the edge between point x_i and x_j
D	Degree matrix

d	Scaling parameter in the Lanczos algorithm
e	Scaling parameter in the QR algorithm
f	Scaling parameter in the secure domain
G	Similarity graph
g	Scaling parameter in the back substitution phase
k	Number of clusters
L	Laplacian matrix
m	Message
m	Number of iterations of the Lanczos algorithm
N	Product of primes p and q, defines the message space
p	Number of iterations of the QR algorithm
p	Prime number
Q	Orthogonal matrix in QR decomposition
q	Prime number
$Q_{\prod}$	Product of orthogonal matrices in QR algorithm
R	Upper triangular matrix in QR decomposition
s_{ij}	Similarity between data point x_i and x_j
T	Tridiagonal Ritz matrix, result of the Lanczos algorithm
U	Eigenvectors of L
V	Lanczos matrix, orthogonal basis of Krylov subspace
W	Eigenvectors obtained after back substitution
x_i	Data point corresponding to user i

Chapter 1

Introduction

The upswing of big data and cloud storage services brings along a myriad of possibilities to compare data points on a large scale. An important technique in data mining is *clustering*, in which data points are grouped according to a similarity measure. Examples of clustering applications are the generation of recommendations in social networks and the identifcation of trends in medical data [45]. The size and real-time updated character of datasets and the wish to combine data from different sources makes it appealing to use an external server or cloud infrastructure to perform a clustering algorithm. However, data owners may be reluctant to upload data to a server because their data is privacy sensitive or holds commercial value. These arguments have motivated research in privacy preserving data mining, in which randomization, perturbation or encryption can be used to obfuscate data values before performing analysis of the data. However, even when measures have been taken to anonymize a dataset, it may not be sufficient to guarantee user privacy. A famous example of this phenomenon is the Netflix dataset that was published in a contest to improve the company's data science practices [17]. The identifying information had been removed from the dataset and the data was perturbed, but still only a small amount of information about a user was necessary to identify the Netflix movie ratings that were given by the user [31]. It is far from straightforward to design a clustering algorithm in which data privacy is guaranteed. A function is said to be *privacy preserving* when "no information can be obtained about any party's inputs other than what follows from the output of the function" [2]. For clustering techniques, this means that values of user data cannot be identified during the algorithm while each user still receives the cluster index to which he or she belongs. This thesis will look at a specific clustering technique: the *spectral clustering algorithm.* One of the cryptographic tools to make a function privacy preserving is *homomorphic encryption.* This is a form of encryption that allows computations to be performed on encrypted values [23].

The aim of this thesis is to research whether the spectral clustering algorithm can be adapted to work on Paillier encrypted data values. The Paillier cryptosystem is an additively homomorphic encryption technique [38]. A pivotal part of spectral clustering is the partial eigendecomposition of a matrix. The computation of eigenvectors is a complex process that scales superlinearly with the size of the matrix. Because of this complexity, less intensive numerical methods are generally used to approximate the eigenvectors of a large matrix. The focus of this thesis will be the approximation of eigenvectors with the Lanczos algorithm and the QR algorithm. When these algorithms are translated to the Paillier message space, new challenges are encountered. The fields of numerical mathematics and cryptography have to be brought together. First of all, Paillier encryption can only be applied to a field of positive integers, $\mathbb{Z}_N$, while the spectral clustering algorithm is generally applied in the real domain. Also, to perform non-linear operations on encrypted data, additional cryptographic protocols have to be invoked. Multiple parties have to participate in the execution of these protocols. Certain operations, such as the division and square root operation, increase the computational complexity of a privacy preserving algorithm significantly. The numerical algorithms will be adapted to be more suitable to work on encrypted values. Figure 1.1 shows the phases of the adaptation of an algorithm to the privacy preserving domain. The algorithms have to be translated to work exclusively on positive integers. Then, the required computations have to be investigated. For privacy preserving non-linear operations, multiple parties need to be involved. Protocols for the computations are designed or adjusted to perform optimally in the algorithmic context. Lastly, the privacy preserving algorithm needs to be implemented so that it can work on encrypted values. This research focuses mainly on phase 1 of the adaptation. Phase 2 will also be discussed, but an in-depth investigation is beyond the scope of this thesis.

The following research questions will be answered:

1. How can the approximation of eigenvectors be performed in the integer domain?

2. How does this influence the performance of the spectral clustering algorithm?

These questions are centered around phase 1 of Figure 1.1. The investigation of phase 2 will take the form of a possible design for the privacy preserving Lanczos algorithm and QR algorithm.

This report is organized as follows. Chapter 2 gives an overview of the spectral clustering algorithm. Chapter 3 zooms in on one part of this algorithm: the computation of several eigenvectors of the Laplacian matrix. The Lanczos algorithm

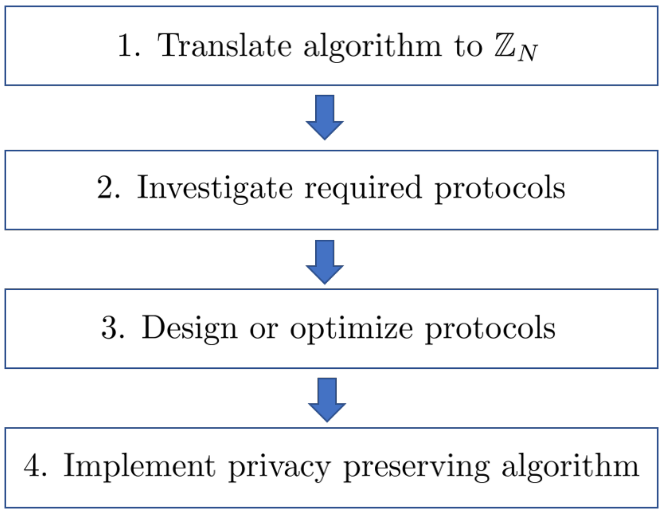

1. Translate algorithm to $\mathbb{Z}_N$
2. Investigate required protocols
3. Design or optimize protocols
4. Implement privacy preserving algorithm

Figure 1.1: Phases in the adaptation of an algorithm to the privacy preserving domain.

and the QR algorithm will be used to approximate the eigenvectors. These algorithms, their convergence properties and the characteristics of importance will be discussed in this chapter. We turn towards the cryptographic background in Chapter 4. The foreseen set-up for a privacy preserving spectral clustering algorithm will be described. Moreover, Paillier encryption and computations on encrypted numbers are discussed. In this chapter the integer domain $\mathbb{Z}_N$ and its arithmetic operations will be defined. In Chapter 5, the algorithms are adapted to the integer domain $\mathbb{Z}_N$. Furthermore, designs of the privacy preserving versions of the algorithms are given. Chapter 6 contains the assessment of the Lanczos and QR algorithm in $\mathbb{Z}_N$. Convergence and cluster quality results will be discussed. Finally, Chapter 7 presents an answer to the research questions. We finish with an extensive discussion of the results and suggestions for future research. Since this thesis attempts to bridge a gap between numerical mathematics and cryptography, some chapters will start from the absolute basics and elaborate widely. This is mainly the case in Chapters 3 and 4.

Chapter 2

Spectral clustering

This chapter will give an introduction to the spectral clustering algorithm. First, the k-means clustering algorithm will be used as an introductory clustering technique. Spectral clustering makes use of k-means clustering and is capable of clustering more complex data structures. The steps of the spectral clustering algorithm and its characteristics will be discussed.

Clustering is a widely used data mining technique to group data points according to a certain degree of similarity. Data points within a cluster should have a high degree of similarity while the similarity between clusters should be low. A long-established clustering technique is *k-means clustering*. Figure 2.1 shows the division of a dataset into seven convex clusters using the k-means clustering technique. The crossed points are the cluster *centroids*, the arithmetic means of the clusters. Per iteration, every data point is assigned to one of the k clusters based on the cluster centroid to which the data point has the lowest Euclidean distance. The centroids are updated until the clusters no longer change.

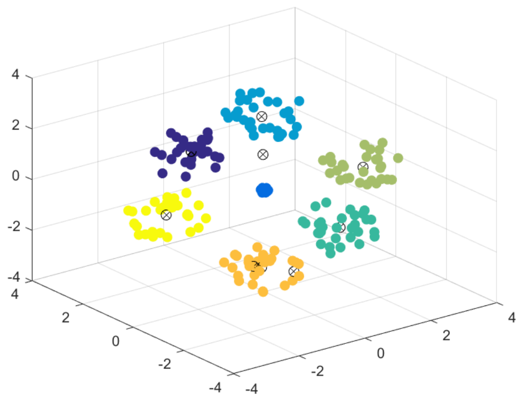


Figure 2.1: The k-means algorithm groups data points into convex clusters. Source of dataset: [46].

The k-means clustering algorithm is given in Algorithm 1 [19].

```
Algorithm 1: The k-means clustering algorithm
1 Set a stopping criterion $\epsilon$.
2 Set $c_1, \ldots, c_k \leftarrow 0$.
3 Initialize $k$ cluster centroids $\tilde{c}_1, \ldots, \tilde{c}_k$.
4 while $\|\tilde{c}_j - c_j\|_2 > \epsilon$ do
5     $c_j \leftarrow \tilde{c}_j$
6     Compute $\|x_i - c_j\|_2$ for every data point $i$ and centroid $j$.
7     Assign every user $i$ to the closest cluster $C_j$.
8     Update the cluster centroids by computing the mean of the cluster points:
       $\tilde{c}_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$.
9 end
```

Now let us consider the result of k-means clustering when we want to recognize two intertwining rings as separate clusters. The result is shown in Figure 2.2. The k-means clustering algorithm fails to distinguish the rings as clusters. Since the rings are not convex sets, data clustering cannot be based solely on the Euclidean distance. Spectral clustering can be used to solve such problems. In spectral clustering, the data points are mapped to a k-dimensional space in which the data points form convex sets. These sets can be clustered with a k-means clustering algorithm. The spectral clustering algorithm will now be described.

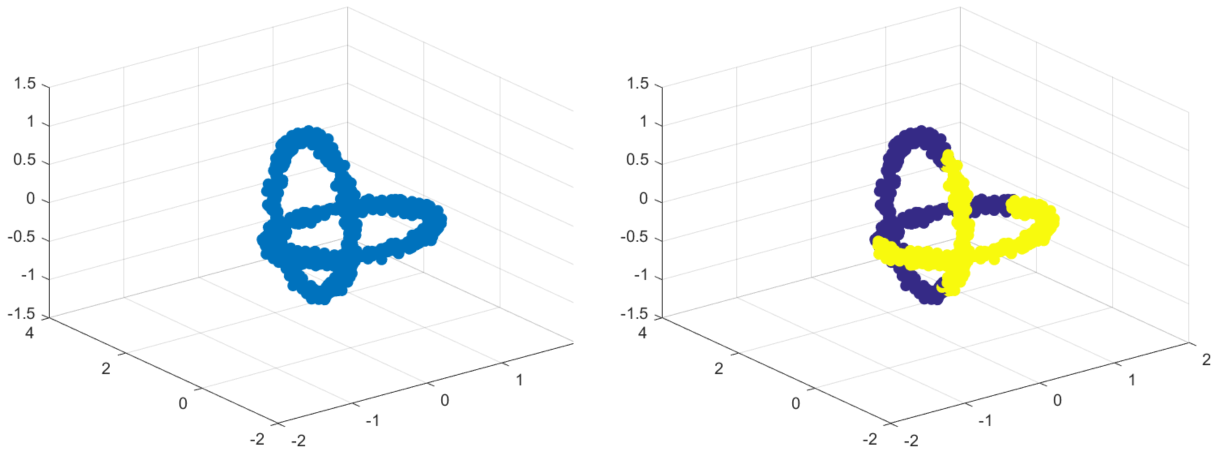


Figure 2.2: The k-means clustering algorithm is not able to differentiate between two intertwining rings. Source of dataset: [46].

In spectral clustering, the set of data points is represented by a *similarity graph*, in which a certain notion of similarity $s_{ij} = s(x_i, x_j) = s(x_j, x_i) \geq 0$ is used to define weights on the edges between all pairs of points x_i and x_j [51]. In Figure

2.3, an example is shown of a weighted undirected similarity graph. The goal of spectral clustering is to cluster the dataset such that the weights on the edges within a cluster are high and the weights on the edges between clusters are low. In Figure 2.3, the division of the graph into two clusters is shown.

A popular example of a similarity function is the Gaussian similarity function

$$s(x_i, x_j) = \exp(-\|x_i - x_j\|^2/(2\sigma^2)). \tag{2.1}$$

With the Gaussian similarity function, the similarity quickly diminishes when two data points are further apart. The values $s(x_i, x_j)$ can be used to construct a graph in different ways. We will discuss three popular constructions. A fully connected graph can be constructed, in which all points are connected with weights $s(x_i, x_j)$ on the edges. Another construction is to determine a threshold ϵ and connect all points x_i and x_j for which $s(x_i, x_j) > \epsilon$. Finally, in a k-nearest neighbors graph, x_i and x_j are connected if x_i is among the k most similar data points to x_j or vice versa.

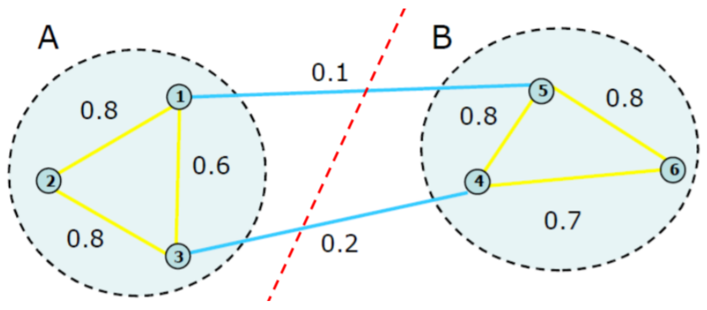


Figure 2.3: The dataset is represented as a graph with weights between the edges. Source: [18].

Given a set of data points $x_1, \ldots, x_n$ and a similarity measure $s_{ij} \geq 0$ that determines a weight $a_{ij} \geq 0$ for the edge between every pair of points x_i and x_j, suppose we have defined a certain undirected similarity graph G based on this set. For a weighted graph $G = (V, E)$, the *adjacency matrix* A is defined as $A = (a_{ij})_{i,j=1,\ldots,n}$. The degree of a vertex $v_i \in V$ is defined as

$$d_i = \sum_{j=1}^{n} a_{ij}. \tag{2.2}$$

Degree matrix D is defined as the diagonal matrix with degrees $d_1, \ldots, d_n$ on the diagonal. Matrices D and A are used in the computation of the unnormalized[1]

[1]A normalized Laplacian can be computed with $L_{norm} = I - D^{-1/2}AD^{-1/2}$ [51]. The normalized Laplacian is also symmetric positive semi-definite and is known to optimize the within-cluster similarity better than an unnormalized Laplacian.

graph Laplacian matrix L :

$$L = D - A. \tag{2.3}$$

Stated intuitively, the Laplacian matrix L contains information about the connected components of graph G. The eigenvalues and eigenvectors of L can be used to map this information to a space in which the connected components form convex sets. In order to find these convex sets, we need the smallest eigenvalues of L. The following proposition shows that 0 is always the smallest eigenvalue of L [51].

Proposition 2.0.1 (Properties of L)**.** *$L \in \mathbb{R}^{n\times n}$ has n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \lambda_n$. The constant one vector $\mathbb{1}$ is the eigenvector that corresponds to eigenvalue 0.*

Proof. Since D and A are symmetric, it follows that L is also symmetric. We will now show that L is positive semi-definite. For every vector $v \in \mathbb{R}^n$ we have

$$\begin{aligned} v^T L v = v^T D v - v^T A v &= \sum_{i=1}^{n} d_i v_i^2 - \sum_{i,j=1}^{n} v_i v_j a_{ij} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} v_i^2 - \sum_{i,j=1}^{n} v_i v_j a_{ij} \\ &= \frac{1}{2}(\sum_{i,j=1}^{n} a_{ij} v_i^2 - 2\sum_{i,j=1}^{n} v_i v_j a_{ij} + \sum_{i,j=1}^{n} a_{ij} v_j^2) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}(v_i - v_j)^2 \geq 0. \end{aligned}$$

So L is positive semi-definitive and it follows that L has non-negative, real-valued eigenvalues. It remains to show that 0 is an eigenvalue of L with the constant one vector $\mathbb{1}$ as corresponding eigenvector. So we want to verify that $L\mathbb{1} = 0$ or equivalently that $D\mathbb{1} = A\mathbb{1}$. Since D is a diagonal matrix and $D\mathbb{1}$ is a vector of length n that contains the row sums of D, we see that $D\mathbb{1}$ is a vector with d_i as ith entry. By definition of d_i, this is equal to the row sums of matrix A. So, $D\mathbb{1} = A\mathbb{1}$. □

When the graph G is not entirely connected, the eigenvalue 0 has multiple eigenvectors. These eigenvectors are the indicator vectors of the connected components of the similarity graph. This is stated in the following proposition of which the proof can be found in [51].

Proposition 2.0.2 (Number of connected components and the spectrum of L)**.** *Let G be an undirected graph with non-negative weights. Then the multiplicity k*

of the eigenvalue 0 of L equals the number of connected components $B_1, \dots, B_k$ *in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors* $\mathbb{1}_{B_1}, \dots, \mathbb{1}_{B_k}$ *of those components.*

In the ideal case in which the similarity between points in different clusters is 0, the first k eigenvectors $u_1, \dots, u_k$ of L (i.e. those corresponding to eigenvalue 0) are indicator vectors of the clusters. Let $U \in \mathbb{R}^{n \times k}$ be the matrix with $u_1, \dots, u_k$ as columns. Then the rows $y_i \in \mathbb{R}^k$ of U are of the ideal form $(0, \dots, 0, 1, 0, \dots, 0)$ in which the position of the 1 indicates to which cluster data point x_i belongs. The mapped data points y_i form convex sets in $\mathbb{R}^k$ and will therefore be trivially correctly clustered by the k-means clustering algorithm. An example of a nearly ideal case is given in Figure 2.4.

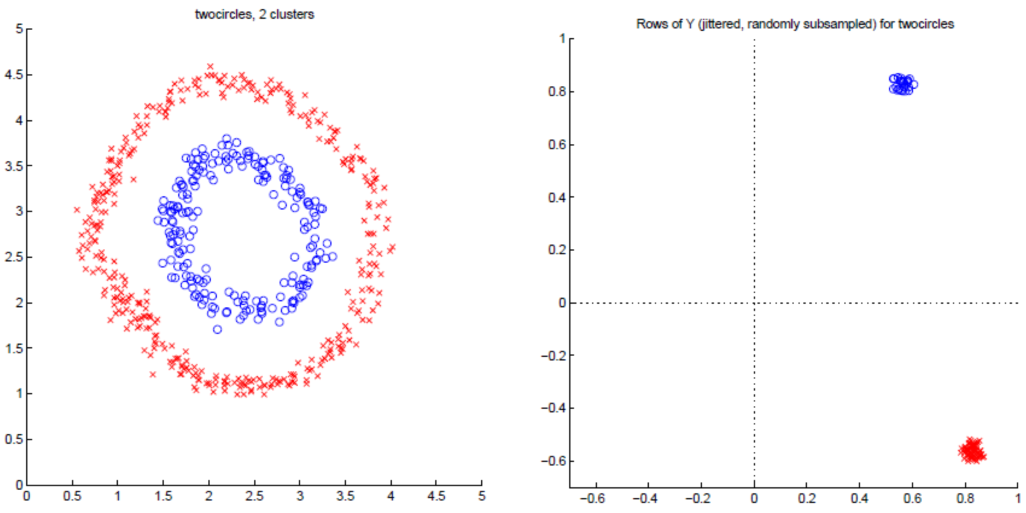


Figure 2.4: After mapping the data points to the first two eigenvectors, they form tight clusters that lie roughly 90° to each other relative to the origin. Source: [32].

The unnormalized spectral clustering algorithm is summarized in Algorithm 2.

Algorithm 2: The spectral clustering algorithm

```
1 Construct a similarity graph $G$ using similarity function $s(x_i, x_j)$.
2 Construct weighted adjacency matrix $A$ from $G$.
3 Construct degree matrix $D$.
4 $L = D - A$.
5 Compute the $k$ smallest eigenvalues of $L$.
6 Compute the first $k$ eigenvectors $u_1, \dots, u_k$ of $L$.
7 Let $U \in \mathbb{R}^{n \times k}$ be the matrix with $u_1, \dots, u_k$ as columns.
8 For $i \in \{1, \dots, n\}$ let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th row of $U$.
9 Cluster the points $y_i$ with the $k$-means algorithm into clusters $C_1, \dots, C_k$.
```

When spectral clustering is applied to real data, we should take into account that there is often not a perfect distinction between connected components. We can look at this real case as a perturbed version of the ideal case. Suppose that we have an ideal Laplacian $L \in \mathbb{R}^{n \times n}$ with eigenvalues $\lambda_1, \dots, \lambda_n$. Let V be the eigenspace that corresponds to the eigenvalues of L. Define the perturbed "real" Laplacian as

$$\tilde{L} := L + H, \tag{2.4}$$

where $H \in \mathbb{R}^{n \times n}$ is a symmetric matrix that introduces weighted edges between the connected components of the graph to which L corresponds. Suppose that $\tilde{L}$ has eigenvalues $\tilde{\lambda}_1, \dots, \tilde{\lambda}_n$ and corresponding eigenspace $\tilde{V}$. Define the *eigengap* $\delta = |\lambda_{k+1} - \lambda_k|$, which can be interpreted as the distance between the interval $[0, \lambda_k]$ and the first eigenvalue outside of the interval. A result by Davis and Kahan now tells us that the following bound on the distance between the eigenspaces V and $\tilde{V}$ exists [8]:

$$d(V, \tilde{V}) \leq \frac{\|H\|_F}{\delta}, \tag{2.5}$$

where $\|.\|_F$ is the Frobenius norm. The eigenvectors of the ideal and the perturbed matrices are closer when the perturbation H is smaller or when the eigengap δ is larger. Therefore, the cluster results of the ideal and the real datasets will be more similar for a smaller perturbation or a larger eigengap. The value of k that maximizes the eigengap will generally give the best clustering result. In this thesis, we will only work with graph Laplacians whose eigenvalue 0 has multiplicity 1. This correponds to a similarity graph that consists of only one connected component. The comparison of the ideal Laplacian L and the real Laplacian $\tilde{L}$ shows that it is still possible to cluster a dataset whose similarity graph consists of only one connected component. However, we should be able to make a distinction between data points which are connected with higher weights or lower weights, so that clusters can be assigned.

In the next chapter, we will zoom in on one step of the spectral clustering algorithm: the computation of the k smallest eigenvalues of the Laplacian and the corresponding eigenvectors.

Chapter 3

Numerical methods

We have seen that spectral clustering uses the eigenvectors of the Laplacian L to isolate connected components in the graph of a dataset. However, the computation of eigenvectors scales badly with the size of the Laplacian. The complexity of computing the entire eigendecomposition of $L \in \mathbb{R}^{n \times n}$ is $\mathcal{O}(n^3)$ [39]. For large systems, this complexity is too high. Moreover, if the data set needs to be clustered into k clusters, only k eigenvectors are required. Therefore, we wish to use numerical methods to approximate k of the eigenvectors efficiently. This chapter will give an overview of the two numerical methods that were applied: the Lanczos algorithm and the QR algorithm. Section 3.1 discusses the Lanczos algorithm, which turns the Laplacian into a smaller tridiagonal matrix whose eigenvalues approximate some of the eigenvalues of L. This algorithm is particularly useful to approximate a few of the extremal eigenvalues of a matrix. The QR algorithm can compute these eigenvalues efficiently. This algorithm is discussed in Section 3.2. The final step in the computation of the eigenvectors is the subject of Section 3.3. Details of the numerical methods will be provided when this helps to gain insight into the convergence properties or the limitations of the methods. Such insight is necessary in order to make informed decisions in the design of these algorithms in a privacy preserving manner.

3.1 The Lanczos algorithm

This section contains a derivation of the Lanczos algorithm and an overview of its properties. Recall the following definition of *orthogonality*:

Definition 3.1.1. *A matrix $A \in \mathbb{R}^{n \times n}$ is said to be orthogonal if $A^T A = I$.*

The Lanczos algorithm generates an orthogonal basis

$$V_k = \{v_1, \ldots, v_k\} \tag{3.1}$$

such that the extremal eigenvalues of L are approximated in a projection of L onto the subspace $\mathcal{V}_k$ of dimension k. The following derivation of the Lanczos algorithm is based on the excellent standard work by Golub and Van Loan [16].

3.1.1 A derivation of the method

Any symmetric matrix $A \in \mathbb{R}^{n\times n}$ has real eigenvalues and its eigenvectors form an orthonormal basis. In other words, the *Schur decomposition* always exists. The proof of the following theorem can be found in [16].

Theorem 3.1.1 (Symmetric Schur decomposition). *If $A \in \mathbb{R}^{n\times n}$ is symmetric, then there exists an orthonormal matrix $U \in \mathbb{R}^{n\times n}$ such that*

$$U^T AU = \Lambda = \; diag(\lambda_1, \dots, \lambda_n) \tag{3.2}$$

and

$$Au_k = \lambda_k u_k \tag{3.3}$$

for $k \in \{1, \dots, n\}$.

The eigenvalues of a symmetric matrix can be found by solving an optimization problem of the *Rayleigh quotient*.

Definition 3.1.2. *Given a symmetric matrix A and a nonzero vector y, the Rayleigh quotient $r(A, y)$ is defined as*

$$r(A, y) = \frac{y^T Ay}{y^T y}. \tag{3.4}$$

When there is no confusion with respect to which matrix the Rayleigh quotient corresponds to, we will write $r(y)$ for the Rayleigh quotient. We want to compute the eigenvectors of the Laplacian L that correspond to the k smallest eigenvalues. Suppose that $\lambda_1 < \ldots < \lambda_n$ are the ordered eigenvalues of L. If the Rayleigh quotient $r(L, y)$ is optimized over $y \neq 0$, we can find the largest and smallest eigenvalue of L [42]:

Theorem 3.1.2. $\lambda_1 = \min\limits_{y\neq 0} \dfrac{y^T Ly}{y^T y}$, $\lambda_n = \max\limits_{y\neq 0} \dfrac{y^T Ly}{y^T y}$.

Proof. The Laplacian L is a symmetric matrix. According to Theorem 3.1.1, the eigenvectors $\{u_i\}_{i=1}^n$ of L form an orthonormal basis. Therefore, any $y \neq 0$ can be constructed with this basis: $y = \sum_{i=1}^n a_i u_i$ for certain scalars a_i. Now

$$\begin{aligned} r(y) &= \frac{y^T L y}{y^T y} && (3.5)\\ &= \frac{(\sum_{i=1}^n a_i u_i^T) L (\sum_{i=1}^n a_i u_i)}{\sum_{i=1}^n a_i^2} && (3.6)\\ &= \frac{(\sum_{i=1}^n a_i u_i^T)(\sum_{i=1}^n \lambda_i a_i u_i)}{\sum_{i=1}^n a_i^2} && (3.7)\\ &= \frac{\sum_{i=1}^n a_i^2 \lambda_i}{\sum_{i=1}^n a_i^2}. && (3.8)\end{aligned}$$

So the Rayleigh quotient is a weighted average of the λ_i. The weights are determined by the a_i. Without proof we will state that there is no loss of generality when we restrict ourselves to $\|y\| = 1$ [16]. Then the a_i have to satisfy $\sum_{i=1}^n a_i^2 = 1$. The smallest eigenvalue λ_1 can be found by choosing $a_1 = 1$, $a_2, \ldots, a_n = 0$, thus minimizing $r(y)$ over y. The largest eigenvalue λ_n is found by choosing $a_1, \ldots, a_{n-1} = 0$, $a_n = 1$, thus maximizing $r(y)$ over y. $\square$

The Lanczos method simplifies the optimization problem from finding $y \in \mathbb{R}^n$ to finding the optimal u in a subspace $\mathcal{V}_k$. When this subspace $\mathcal{V}_k$ is increased, a better approximation for the original Rayleigh quotient can be achieved. Suppose that we have an orthonormal matrix $V_k = \begin{bmatrix} v_1 & \ldots & v_k \end{bmatrix}$ whose columns form an orthonormal basis of subspace $\mathcal{V}_k$. The matrix V_k is called the *Lanczos matrix*. We can write any $u \in \mathcal{V}_k$ in terms of these basis vectors: $u = \sum_{i=1}^k y_i v_i = V_k y$ for some $y \in \mathbb{R}^{k\times 1}$. Moreover, since $V_k^T V_k = I$,

$$r(u) = r(V_k y) = \frac{y^T V_k^T L V_k y}{y^T y}. \tag{3.9}$$

Suppose that the maximum and minimum of the Rayleigh quotient are attained for $u, v \in \mathcal{V}_k$ respectively. So $u, v \in \operatorname{span}\{v_1, \ldots, v_k\}$. We want to expand the optimization problem to $\mathcal{V}_{k+1}$ by finding a basis vector v_{k+1} that is orthogonal to the previous basis. To guarantee the discovery of a more optimal answer in a larger subspace, it makes sense to search the new basis vectors v_{k+1} in the direction of the gradient $\nabla r(u), \nabla r(v)$. So

$$\nabla r(u), \nabla r(v) \in \operatorname{span}\{v_1, \ldots, v_k, v_{k+1}\}. \tag{3.10}$$

Now the gradient of a general Rayleigh quotient $r(x)$ is computed as

$$\nabla r(x) = \left[\frac{\partial r(x)}{\partial x_1} \dots \frac{\partial r(x)}{\partial x_k}\right] \tag{3.11}$$

$$\begin{aligned}
\text{where } \frac{\partial r(x)}{\partial x_j} &= \frac{\partial}{\partial x_j}\left(\frac{x^T Lx}{x^T x}\right) \\
&= \frac{\frac{\partial}{\partial x_j}(x^T Lx)x^T x - x^T Lx \frac{\partial}{\partial x_j}(x^T x)}{(x^T x)^2} \\
&= \frac{\frac{\partial}{\partial x_j}(x^T)Lx + x^T \frac{\partial}{\partial x_j}(Lx)}{x^T x} - \frac{(x^T Lx)2x_j}{(x^T x)^2} \\
&= \frac{2(Lx)_j}{x^T x} - \frac{(x^T Lx)2x_j}{(x^T x)^2}.
\end{aligned} \tag{3.12}$$

When all the partial derivatives are combined, we obtain

$$\nabla r(x) = \frac{2}{x^T x}(Lx - r(x)x). \tag{3.13}$$

So $\nabla r(x) \in \text{ span}\{x, Lx\}$ and therefore

$$\nabla r(u), \nabla r(v) \in \text{ span}\{v_1, \dots, v_k, Lv_1, \dots, Lv_k\}. \tag{3.14}$$

Both (3.10) and (3.14) hold for a set of vectors $\{v_1, \dots, v_k, v_{k+1}\}$ which satisfies

$$\text{span}\{v_1, \dots, v_k, Lv_1, \dots, Lv_k\} = \text{ span}\{v_1, \dots, v_k, v_{k+1}\}. \tag{3.15}$$

We will now define a subspace for which the basis will satisfy (3.15). This subspace is called the *Krylov subspace.*

Definition 3.1.3. *The Krylov subspace $\mathcal{K}_k(L, v)$ is the linear subspace spanned by the images of L applied to v: $\mathcal{K}_k(L, v) = \text{ span}\{v, Lv, L^2v, \dots, L^{k-1}v\}$.*

Hence, we want to find an orthonormal basis for the Krylov subspace $\mathcal{K}_k(L, v_1)$ in order to be able to approximate the extremal eigenvalues of L. A useful property of this orthonormal basis is that it can be used for a similarity transformation of L to a tridiagonal matrix T. This is stated in the following theorem:

Theorem 3.1.3. *Suppose that $V_k = \begin{bmatrix} v_1 & \dots & v_k \end{bmatrix}$ is an orthonormal basis for the Krylov subspace $\mathcal{K}_k(L, v_1)$. Then $V_k^T LV_k = T$ is a tridiagonal matrix.*

Proof. We want to prove that $T_{ij} = v_i^T Lv_j = 0$ for $i > j + 1$ and $i < j - 1$. Now $Lv_j \in \text{ span}\{v_1, Lv_1, \dots, L^j v_1\} = \text{ span}\{v_1, \dots, v_{j+1}\}$. The v_i are mutually orthogonal so $v_i^T v_j = 0$ when $i > j + 1$. So $T_{ij} = v_i^T Lv_j = 0$ when $i > j + 1$. Furthermore, $V_k^T LV_k = T$ is symmetric so $T_{ij} = 0$ when $i < j - 1$. □

We can interpret T as the projection of the Laplacian onto the Krylov subspace $\mathcal{V}_k$. The eigenvalues $\theta_1, \dots, \theta_m$ of T are called *Ritz values* and are increasingly better estimates of the eigenvalues of L as the dimension of the Krylov subspace increases.

3.1.2 The Lanczos algorithm

The eigenvalues of the tridiagonal matrix T are less computationally complex to compute than those of the original matrix $L \in \mathbb{R}^{n\times n}$. Suppose that

$$V_n = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix} \tag{3.16}$$

is an orthonormal basis for the Krylov subspace $\mathcal{K}_n(L, v_1)$. The Lanczos method exploits the sparsity of L by making use of the relation $LV_n = V_nT$ to compute the elements of T, called the *Ritz matrix*, directly:

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & & & \beta_n & \alpha_n \end{pmatrix}. \tag{3.17}$$

From $LV = VT$ follow the relations

$$\begin{aligned} Lv_k &= \beta_k v_{k-1} + \alpha_k v_k + \beta_{k+1} v_{k+1}, && (3.18)\\ v_k^T L v_k &= \beta_k v_k^T v_{k-1} + \alpha_k v_k^T v_k + \beta_{k+1} v_k^T v_{k+1}, && (3.19)\\ v_k^T L v_k &= \alpha_k v_k^T v_k, && (3.20)\\ \alpha_k &= \frac{v_k^T L v_k}{v_k^T v_k}. && (3.21) \end{aligned}$$

From (3.18) it is also possible to compute the β_{k+1} and v_{k+1}:

$$\begin{aligned} \beta_{k+1} v_{k+1} &= (L - \alpha_k I) v_k - \beta_k v_{k-1}, && (3.22)\\ v_{k+1} &= \frac{(L - \alpha_k I) v_k - \beta_k v_{k-1}}{\beta_{k+1}}. && (3.23) \end{aligned}$$

By setting $r_k = (L - \alpha_k I)v_k - \beta_k v_{k-1}$ and $\beta_{k+1} = \|r_k\|_2$, the orthonormal v_i are obtained iteratively. The starting vector can be chosen randomly. This derivation of the Lanczos algorithm is summarized as follows:

Algorithm 3: Normalized Lanczos algorithm

```
1 Generate a random vector $v_1 \in \mathbb{R}^n$.
2 Set $v_0 = \underline{0}$ and $\beta_1 = 1$.
3 for $j = 1, 2, \dots, n-1$ do
4     $\alpha_j \leftarrow (Lv_j \cdot v_j)/(v_j \cdot v_j)$
5     $r_j \leftarrow Lv_j - \alpha_j v_j - \beta_j v_{j-1}$
6     $\beta_{j+1} \leftarrow \|r_j\|_2$
7     $v_{j+1} \leftarrow r_j / \beta_{j+1}$
8 end
9 $\alpha_n \leftarrow (Lv_n \cdot v_n)/(v_n \cdot v_n)$
```

The complexity of numerical algorithms is measured in the number of elementary floating point operations or *flops*. We will count the number of flops for a dense matrix L. Note that the complexity is actually reduced for a sparse L because the matrix vector product costs fewer operations. Per iteration, the Lanczos method costs one matrix vector product ($2n^2 - n$ flops), two inner products ($4n - 2$ flops), two scalar vector multiplications ($2n$ flops) and two vector subtractions ($2n$ flops). Also, the norm of r_j has to be computed which requires a square root operation. This is not counted in the number of flops. In m iterations, the complexity can be estimated at $\mathcal{O}(mn^2)$ flops.

3.1.3 Convergence properties

If $L \in \mathbb{R}^{n\times n}$ and $d =$ rank$(\mathcal{K}_n(L, v_1)$, the Lanczos algorithm will terminate at iteration d. The Krylov subspace is invariant after this iteration and a complete basis for the subspace has been found. A value $\beta_{d+1} = 0$ is encountered in the algorithm after which it breaks down. When L has a full rank, the Lanczos algorithm will only terminate after n iterations. Since we only need a few of the smallest eigenvalues of the Laplacian matrix and extremal eigenvalues are the first to converge in matrix T, we would like to terminate the Lanczos algorithm prematurely. Suppose that we terminate the algorithm at step m after obtaining tridiagonal $m \times m$ matrix T_m and Lanczos matrix V_m. We need to know how well the spectrum of T_m approximates the spectrum of L.

Suppose that the eigendecomposition of T_m is

$$T_m S_m = S_m \Theta_m.$$

Here, Θ_m is a diagonal matrix whose entries are the Ritz values. Ritz values are approximations of the eigenvalues of L. The columns s_i of S_m are the corresponding eigenvectors of T_m. The approximations of the eigenvectors of L are called *Ritz vectors* and can be computed with $u_i = V_m s_i$. The Ritz vectors form an equivalent orthonormal basis for the Krylov subspace. The quality of the approximation of the Ritz value and its Ritz vector to an eigenpair of L can be assessed with the following theorem [16]:

Theorem 3.1.4. *Suppose that m steps of the Lanczos algorithm have been performed and that $S_m^T T_m S_m =$diag$(\theta_1, \ldots, \theta_m)$ is the eigendecomposition of T_m. If $U_m = [u_1 \ldots u_m] = V_m S_m \in \mathbb{R}^{n\times m}$, then for $i = 1, \ldots, m$ we have*

$$\|Au_i - \theta_i u_i\|_2 = |\beta_m||s_i(m)| \tag{3.24}$$

where $s_i(m)$ denotes the entry on position (m, i) of matrix S.

Proof. By construction, $LV_m = V_mT_m + r_me_m^T$ with $r_m = (L - \alpha_mI)v_m - \beta_{m-1}v_{m-1}$. Multiply this relation by the matrix of Ritz vectors to obtain:

$$LV_mS_m = V_mT_mS_m + r_me_m^TS_m, \tag{3.25}$$

$$LU_m = V_mS_m\Theta_m + r_me_m^TS_m. \tag{3.26}$$

So the i-th column satisfies the relation

$$Lu_i = \theta_iu_i + r_me_m^TS_me_i. \tag{3.27}$$

It follows that $\|Lu_i - \theta_iu_i\|_2 = |\beta_m| \cdot |s_{mi}|$ because $|\beta_m| = \|r_m\|_2$. □

Since S is an orthonormal matrix, its entries are bounded. Therefore, a Ritz value will have converged to an eigenvalue of A when β_m is small. However, even the occurrence a small β_m is a rarity in practice [16]. Therefore, the accuracy of the Ritz value is often used as a stopping criterion. After a certain number of iterations, the Ritz values of the Ritz matrix T are computed and the bound on their accuracy is computed using Theorem 3.1.4. When this bound is small enough, the Lanczos algorithm is considered to be terminated. If m is the total number of iterations and the input matrix has size $n \times n$, the algorithm can often be stopped at values of m as small as $2\sqrt{n}$ [40].

Until the first convergence, the Lanczos algorithm should behave as predicted in exact arithmetic. After convergence, duplicate copies of Ritz values tend to show up. The next section will explain why this occurs.

3.1.4 Limitations

Loss of orthogonality

In exact arithmetic, the Lanczos vectors in matrix V are perfectly orthogonal. However, rounding errors due to finite precision computations cause a loss of orthogonality. This phenomenon was researched in the pioneering work of Christopher Paige. One of the most important results of Paige was to express the loss of orthogonality at iteration j between a Ritz vector u_i $(i < j)$ and Lanczos vector v_{j+1} as follows [35, 37]:

$$u_i^Tv_{j+1} \approx \frac{\epsilon\|T_j\|_2}{\beta_{i+1} \cdot s_{ji}}, \tag{3.28}$$

where ϵ is a small scalar that quantifies the rounding error and s_{ji} is the bottom element of T_j's eigenvector s_i. Equation 3.28 shows that loss of orthogonality occurs when β_{i+1} is small. On the other hand, from Theorem 3.1.4 it was deducted that the convergence of a Ritz pair is indicated by a small value of β_{i+1}.

Thus, Paige discovered that loss of orthogonality and the convergence of a Ritz pair go hand in hand.

Paige also gave an analysis for this phenomenon [5, 35]. When a Ritz value converges, all the Lanczos vectors v_j are perturbed in the direction of the converged Ritz vector. The orthogonality of the subsequently computed eigenvectors is lost in this direction. This Ritz vector becomes part of the subsequently computed Lanczos vectors again. As an effect, the eigenvalue and corresponding eigenvector are discovered again by the algorithm. These double eigenvalues are referred to as "ghost eigenvalues" or *spurious eigenvalues.* These multiple eigenvalues of Ritz matrix T_j at iteration j correspond to the same eigenvalue of L with algebraic multiplicity 1. The basis $\{v_1, \dots, v_k\}$ of the Krylov subspace now contains almost dependent vectors, leading to a loss of efficiency as unnecessarily many vectors have to be computed and stored. Thankfully, there are methods to restore the orthogonality of the Lanczos vectors.

Dealing with spurious eigenvalues

Three of the most widely applied techniques to deal with spurious eigenvalues are full reorthogonalization, selective orthogonalization and the detection of spurious eigenvalues, after which they can be discarded. These techniques will now be discussed briefly.

Orthogonality of Lanczos vectors can be guaranteed by *full reorthogonalization.* The orthogonality coefficients can be computed by applying the Gram-Schmidt process to vector v_{j+1} in every iteration j . Matrix V_j contains the normalized vectors $v_1, \dots, v_j$. The improved v'_{j+1} is obtained as follows:

$$\begin{aligned} h &= V_j^T v_{j+1}, \\ v'_{j+1} &= v_{j+1} - V_j h. \end{aligned} \tag{3.29}$$

However, full reorthogonalization destroys the simple character of the Lanczos algorithm. The number of operations in every iteration increases since an additional matrix-vector product is introduced in every iteration [6]. Parlett and Scott designed a more efficient reorthogonalization method based on the phenomenon that loss of orthogonality occurs in the direction of the converged Ritz vectors [40]. They proposed *partial reorthogonalization*, a method in which a new Lanczos vector v_{j+1} is reorthogonalized against all Ritz vectors u_i whose Ritz values θ_i satisfy

$$|\lambda_i - \theta_i| = \beta_j |s_{ji}| \leq \sqrt{\epsilon} \|L\|_2, \tag{3.30}$$

where ϵ and s_{ji} are defined as in (3.28). Although partial reorthogonalization requires considerably fewer arithmetic operations, a disadvantage is that the Ritz vectors have to be computed intermediately and the $\beta_j|s_{ji}|$ have to be compared to a threshold.

A third method is to accept the occurrence of spurious eigenvalues and to try to identify and discard them. Cullum and Willoughbly proposed to look at a submatrix $\hat{T}_j$ of T_j at every iteration j [6]. The submatrix $\hat{T}_j$ is obtained by removing the first row and the first column of T_j. If an eigenvalue θ_j^i of T_j is also an eigenvalue of $\hat{T}_j$, this eigenvalue is labelled as spurious and discarded. The argument behind this method is that the tridiagonal matrix T_j is *unreduced.* Unreduced tridiagonal matrices only have simple eigenvalues, i.e. eigenvalues with multiplicity 1. This property follows from the following definitions and theorem.

Definition 3.1.4. *A matrix is called an upper Hessenberg matrix when all the entries below the first subdiagonal are zero.*

Definition 3.1.5. *An upper Hessenberg matrix A is called unreduced when its subdiagonal entries are nonzero.*

Theorem 3.1.5. *An unreduced diagonalizable tridiagonal matrix $T \in \mathbb{R}^{n\times n}$ has n distinct eigenvalues.*

Theorem 3.1.5 also implies that it is not possible to find eigenvalues with a multiplicity larger than 1 with the standard Lanczos method. The next section will expound on this limitation.

Finding multiple eigenvalues

The approximation of eigenvalues of L with multiplicity larger than 1 is not possible with the standard Lanczos algorithm, since the matrices T_j are unreduced and tridiagonal. In order to compute linearly independent eigenvectors corresponding to a multiple eigenvalue, the *block Lanczos method* can be applied [6, 16]. The block Lanczos method starts with p starting directions instead of a single starting vector. Let $B = [b_1, \dots, b_p]$ be the matrix with the starting directions as columns, where p is the multiplicity of the desired eigenvalue. The goal of the algorithm is to compute an orthonormal basis for the Krylov subspace

$$\mathcal{K}_k(L, B) = span\{LB, L^2B, \dots, L^{k-1}B\}. \tag{3.31}$$

In every iteration j, an orthonormal basis V_j for Krylov subpace $L^{j-1}B$ is constructed. The concatenated matrix $V = [V_1 \dots V_m]$ is the orthonormal basis after final iteration m. A block tridiagonal matrix T is also constructed by the algorithm:

$$T = \begin{pmatrix} A_1 & B_2^T & & 0 \\ B_2 & A_2 & B_3^T & \\ & B_3 & \ddots & \\ & & & \\ 0 & & & A_m \end{pmatrix}. \tag{3.32}$$

The block tridiagonal matrix T is similar to L and the Ritz values have the same multiplicities as the eigenvalues of L. The block Lanczos method has a large disadvantage. Whereas the construction of a linearly dependent basis vector v_{j+1} means the termination of the algorithm in the standard Lanczos algorithm, this is not the case in the block Lanczos algorithm. Moreover, linearly dependent vectors have to be detected and removed because the similarity property between L and T is lost otherwise. The process of detection and removal of linearly dependent vectors is called *deflation*. The current research is focused on the standard Lanczos algorithm, so the block Lanczos method is not considered. However, since spectral clustering may require the computation of multiple eigenvectors corresponding to the same eigenvalue, the block Lanczos algorithm is an interesting consideration for further research.

After m iterations of the Lanczos algorithm, a tridiagonal matrix $T \in \mathbb{R}^{m \times m}$ is obtained. The QR algorithm will be applied to T to obtain its Ritz values.

3.2 The QR algorithm

This section gives an overview of the QR algorithm. The algorithm makes use of a QR decomposition in every iteration. First, the QR decomposition will be explained, after which the QR algorithm is derived. The convergence and possible accelerations hereof are also discussed.

3.2.1 The QR decomposition

The following theorem states that the QR decomposition exists for any matrix $A \in \mathbb{R}^{n \times n}$:

Theorem 3.2.1. *Any matrix $A \in \mathbb{R}^{n \times n}$ can be decomposed into a unique pair of an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n \times n}$ with positive diagonal entries such that*

$$A = QR. \tag{3.33}$$

When the columns of A are linearly independent, the columns of Q form an orthonormal basis for the column space of A. The QR factorization of A can be computed directly by making use of the following relation for every column in a_k of $A = QR$:

$$a_k = \sum_{i=1}^{k-1} r_{ik} q_i + r_{kk} q_k \tag{3.34}$$

$$q_k = \frac{a_k - \sum_{i=1}^{k-1} r_{ik} q_i}{r_{kk}}. \tag{3.35}$$

To ensure that q_k is orthogonal to $\{q_1, \ldots, q_{k-1}\}$ and that it is normalized, the entries of R are defined for $i \in \{1, \ldots, k-1\},\ k \in \{1, \ldots, n\}$ as

$$r_{ik} = q_i^T a_k, \tag{3.36}$$

$$r_{kk} = \|a_k - \sum_{i=1}^{k-1} r_{ik} q_i\|_2. \tag{3.37}$$

So, the projections of a column of A onto the previous q_i are subtracted from this column to generate a new orthogonal basis vector. The computation of an orthogonal set of vectors in this way is called the *classical Gram-Schmidt method.* Unfortunately, this method is numerically unstable since subsequent columns of Q are often not entirely orthogonal due to rounding errors. A more stable variant of the Gram-Schmidt method makes a new column of Q orthogonal to every previous column of Q separately. In this way, a new column q_{k+1} is also orthogonalized against errors that were introduced into the previous columns. This variant is called the *modified Gram-Schmidt algorithm.* The QR decomposition using the Gram-Schmidt method is summarized in Algorithm 4.

Algorithm 4: QR decomposition using Gram-Schmidt orthogonalization

```
1  $q_1 \leftarrow \underline{0}$
2  for $j = 1, \ldots, n$ do
3      $v_{GS} = a_j$
4      for $i = 1, \ldots, j-1$ do
5          $r_{ij} \leftarrow q_i^T a_j$ in classical Gram-Schmidt or
6          $r_{ij} \leftarrow q_i^T v_{GS}$ in modified Gram-Schmidt
7          $v_{GS} \leftarrow v_{GS} - r_{ij} q_i$
8      end
9      $r_{jj} \leftarrow \|v_{GS}\|_2$
10     $q_j \leftarrow v_{GS} / r_{jj}$
11 end
```

Let us now discuss the complexity of the modified Gram-Schmidt algorithm on a $n \times n$ matrix A. In iteration k, the following computations are performed:

$k-1$ inner products cost $(k-1)(2n-1)$ flops, multiplication and subtraction of these inner products costs $2(k-1)n$ flops, and the computation of r_{kk} and the new column of Q costs $3n$ flops. In total, the complexity of the modified Gram-Schmidt algorithm can be estimated at $2n^3$ flops. Later in this chapter, we will see that the structure of T is exploited to reduce the complexity of the QR algorithm.

Other methods to decompose A into Q and R make use of *Householder reflections* or *Givens rotations*. Both of these methods transform A into an upper triangular matrix R by introducing zero elements below the diagonal in a column of A through a matrix multiplication. These methods are numerically more stable, but are approximately twice more complex than the modified Gram-Schmidt algorithm [16]. Therefore, we prefer to apply the Gram-Schmidt method in the QR decomposition if the resulting decomposition is accurate enough.

The QR decomposition is the basis for the QR algorithm. This will be shown in the next section.

3.2.2 The QR algorithm

We set $A = A_0$ and will use its QR decomposition in an iterative algorithm. Suppose that we have found the QR decomposition of matrix A_{k-1} for a certain k:

$$A_{k-1} = Q_k R_k. \tag{3.38}$$

The reversed multiplication yields the next matrix A_k:

$$A_k = R_k Q_k. \tag{3.39}$$

The QR decompositions are stored in the following sequences:

$$\begin{aligned} Q^{(k)} &= Q_1 \cdot \ldots \cdot Q_k, & (3.40)\\ R^{(k)} &= R_1 \cdot \ldots \cdot R_k. & (3.41) \end{aligned}$$

Both orthonormal matrices and upper triangular matrices are closed under multiplication, so $Q^{(k)}$ is an orthonormal matrix and $R^{(k)}$ is upper triangular. The next theorem states that all subsequent A_k are similar to the original matrix A_0.

Theorem 3.2.2. *Let $A_0 \in \mathbb{R}^{m\times m}$ and let A_k and $Q^{(k)}$ be defined as in (3.39) and (3.40) respectively. Then A_k and A_0 have the same eigenvalues since the following relation holds: $A_0 Q^{(k)} = Q^{(k)} A_k$.*

Proof. We will prove this theorem with induction. For $k = 1$, we have $A_0 = Q_1R_1$ so $Q^{(1)} = Q_1$ and $R^{(1)} = R_1$. So

$$\begin{aligned} A_0 &= Q^{(1)}R^{(1)} \\ A_0 &= Q^{(1)}R^{(1)}Q^{(1)}(Q^{(1)})^T \\ A_0Q^{(1)} &= Q^{(1)}R^{(1)}Q^{(1)} \\ A_0Q^{(1)} &= Q^{(1)}A_1. \end{aligned} \tag{3.42}$$

The hypothesis holds for the basis step. Now suppose that $A_0Q^{(k)} = Q^{(k)}A_k$ for some value of k. We isolate A_k and use its QR decomposition to find:

$$\begin{aligned} A_k &= (Q^{(k)})^T A_0 Q^{(k)} \\ Q_{k+1}R_{k+1} &= (Q^{(k)})^T A_0 Q^{(k)} \\ R_{k+1}Q_{k+1} &= (Q_{k+1})^T(Q^{(k)})^T A_0 Q^{(k)} Q_{k+1} \\ A_{k+1} &= (Q^{(k+1)})^T A_0 Q^{(k+1)} \\ Q^{(k+1)}A_{k+1} &= A_0Q^{(k+1)}. \end{aligned} \tag{3.43}$$

This proves the result for $k + 1$ and we have thus proven the relation for all $k \in \mathbb{N}$. □

Moreover, the A_k converge to an upper triangular matrix. In order to give the arguments for this convergence, we first need the following result:

Theorem 3.2.3. *Let $A_0 = A \in \mathbb{R}^{m\times m}$ and let $A_k, Q^{(k)}$ and $R^{(k)}$ be defined as in* (3.39)*,* (3.40) *and* (3.41)*. The QR decomposition of A^k, the kth power of A, can be written as $A^k = Q^{(k)}R^{(k)}$.*

Proof. We will prove the result with induction. The basis step $A_0 = Q^{(1)}R(1)$ is trivially true. So suppose that $A^k = Q^{(k)}R^{(k)}$ for some k. By Theorem 3.2.2 we can write A as $A = Q^{(k)}A_k(Q^{(k)})^T$ and so

$$\begin{aligned} A^{k+1} = AA^k &= Q^{(k)}A_k(Q^{(k)})^T A^k \\ &= Q^{(k)}A_k(Q^{(k)})^T Q^{(k)}R^{(k)} \\ &= Q^{(k)}A_kR^{(k)} \\ &= Q^{(k)}Q_{k+1}R_{k+1}R^{(k)} \\ &= Q^{(k+1)}R^{(k+1)}. \end{aligned}$$

We can conclude that the result holds for all $k \in \mathbb{N}$. □

We will now explain why A_k converges to an upper triangular matrix. The formal proof is lengthy and can be found in [16]. A discussion of the entire proof does not contribute to the aim of this chapter, which is to give an intuitive idea of the effectiveness of the QR algorithm. The inspiration for this overview was taken from [4].

Suppose that the eigenvalues of A are $\lambda_1, \ldots, \lambda_m$ which are the entries of the diagonal matrix Λ. We assume that the eigenvalues are ascending and distinct, so

$$|\lambda_1| < |\lambda_2| < \ldots < |\lambda_m|. \tag{3.44}$$

The corresponding eigenvectors $u_1, \ldots, u_m$ constitute the columns of matrix U. Now we can decompose $A = U\Lambda U^{-1}$ and hence the k-th power of A satifies

$$A^k = U\Lambda^k U^{-1} = \sum_{j=1}^{m} u_j \lambda_j^k w_j^T,$$

where w_j is the jth row of U^{-1}. Since λ_m^k dominates the other eigenvalues strongly, we can approximate A^k by

$$A^k \approx \lambda_m^k u_m w_m^T. \tag{3.45}$$

From Theorem 3.2.3 we know that this matrix can also be decomposed as $A^k = Q^{(k)} R^{(k)}$. We can use these two relations to express the QR decomposition in terms of the eigendecomposition. The first vector q_1 of $Q^{(k)}$ can be found by normalizing the first column of A^k. Equation (3.45) tells us that we can approximate q_1 by normalizing u_m. In the computation of subsequent orthonormal vectors $q_2, \ldots, q_m$, we need more approximation terms because the orthogonal vectors cannot be multiples of each other. For example, to compute q_2 an extra term in the approximation of A^k is used:

$$A^k \approx \lambda_m^k u_m w_m^T + \lambda_{m-1}^k u_{m-1} w_{m-1}^T. \tag{3.46}$$

So the second column of A^k is a linear combination of u_m and u_{m-1}. Therefore, q_2 can be found by applying Gram-Schmidt orthogonalization to $\{u_m, u_{m-1}\}$. In an analogous manner, q_j is computed by applying the Gram-Schmidt procedure to the basis $\{u_m, \ldots, u_{m-j+1}\}$. This boils down to the following relation between $Q^{(k)}$ and U:

$$\begin{bmatrix} q_1 & \ldots & q_m \end{bmatrix} = \begin{bmatrix} u_m & \ldots & u_1 \end{bmatrix} R, \tag{3.47}$$

for an upper triangular matrix R. The quality of this approximation can be measured by defining the following value:

$$\tau = \max_j \left\{ \frac{|\lambda_j|}{|\lambda_{j+1}|} \right\}. \tag{3.48}$$

The approximation is better when τ becomes smaller. We will now show that $A_k = (Q^{(k)})^T A_0 Q^{(k)}$ becomes upper triangular when k is sufficiently large:

$$\begin{aligned}(Q^{(k)})^T A_0 Q^{(k)} &= (Q^{(k)})^T A_0 (UR)\\ &= (Q^{(k)})^T U \Lambda R\\ &= (Q^{(k)})^T Q^{(k)} R^{-1} \Lambda R\\ &= R^{-1} \Lambda R.\end{aligned}$$

Now R and R^{-1} are upper triangular and Λ is a diagonal matrix, so the product is again upper triangular. The QR algorithm converges linearly with a convergence rate that is approximated by τ, so more distinct eigenvalues lead to a better convergence. The eigenvalues of an upper triangular matrix are the entries on its diagonal, so the QR algorithm can be used to find the eigenvalues of A.

The QR algorithm has been discussed by making use of general matrix $A \in \mathbb{R}^{m\times m}$. In this research, the QR algorithm will be applied to the tridiagonal matrix T that is obtained in the Lanczos algorithm. From now on, T will be used instead of A. The QR algorithm is given in Algorithm 5.

Algorithm 5: QR algorithm

```
1 Set $T_1 \leftarrow T$ and $Q_{\prod} \leftarrow I \in \mathbb{R}^{m\times m}$
2 for $k = 1, 2, \dots$ do
3     Compute $T_k = Q_k R_k$ using a QR decomposition method.
4     $T_{k+1} \leftarrow R_k Q_k$.
5     $Q_{\prod} \leftarrow Q_{\prod} Q_k$
6 end
```

The QR algorithm is generally expensive, with a complexity of $\mathcal{O}(n^3)$ per iteration. However, the structure of T can be exploited. Since T is upper Hessenberg, then the amount of work per iteration is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ [16].

Stopping criterion

The A_k converge to an upper triangular matrix. Since the QR decomposition of an upper triangular matrix is

$$A_k = IR = IA_k, \tag{3.49}$$

the subsequent matrix $A_{k+1} = RQ$ is equal to A_k. Therefore, a stopping criterion could be to measure the difference between A_k and A_{k+1} with the Frobenius norm and to stop the algorithm when

$$\|A_k - A_{k+1}\|_F < \epsilon \tag{3.50}$$

for a certain $\epsilon > 0$. Another stopping criterion is to check whether the subdiagonal entries of A_k have converged closely enough to zero.

3.2.3 Using shifts

If the eigenvalues of tridiagonal matrix T are ordered according to (3.44), the jth subdiagonal entry of the subsequent T_k converges to zero with rate $(\lambda_j/\lambda_{j+1})^k$ in iteration k [16]. We will improve this convergence rate by applying a shift μ to T. The eigenvalues of $T - \mu I$ are

$$|\lambda_1 - \mu| < \ldots < |\lambda_m - \mu|. \tag{3.51}$$

Hence, when applying the QR algorithm to $T - \mu I$, the convergence rate of the jth subdiagonal entry becomes

$$\left| \frac{\lambda_j - \mu}{\lambda_{j+1} - \mu} \right|^k . \tag{3.52}$$

If μ is chosen such that it is closer to λ_j than to λ_{j+1}, the new convergence rate is an improvement. The algorithm now converges quadratically instead of linearly. In order to guarantee similarity between subsequent T_k, the shift is added to T_{k+1} after an iteration of the QR algorithm so that:

$$\begin{aligned} T_{k+1} &= RQ + \mu I \\ &= Q^T QRQ + Q^T Q\mu I \\ &= Q^T(QR + \mu I)Q = Q^T T_k Q. \end{aligned} \tag{3.53}$$

The shifted QR algorithm is given in Algorithm 6.

Algorithm 6: Shifted QR algorithm

```
1 Set $T_1 \leftarrow T$ and $Q_\prod \leftarrow I \in \mathbb{R}^{m \times m}$
2 for $k = 1, 2, \ldots$ do
3     Determine shift $\mu$
4     Compute $T_k - \mu I = Q_k R_k$ using a QR decomposition method
5     $T_{k+1} \leftarrow R_k Q_k + \mu I$
6     $Q_\prod \leftarrow Q_\prod Q_k$
7 end
```

The approximation of an eigenvalue is a good choice for a shift. A commonly picked shift is the *Rayleigh quotient shift.* In iteration k the Rayleigh quotient shift is

$$\mu^{(k)} = \frac{q_m^T T_k q_m}{q_m^T q_m} = q_m^T T_k q_m, \tag{3.54}$$

where q_m is the last column of matrix Q_k. Since Q_k is an orthonormal matrix, the Rayleigh quotient shift corresponds to setting $\mu^{(k)} = T_{k(m,m)}$, the last diagonal entry of T. According to Golub and Van Loan, it is "a good heuristic to regard $T_{k(m,m)}$ as the best approximate eigenvalue along the diagonal" [16]. Alternatively, one can compute the eigenvalues of a square subset of T_k and use them as shifts [53].

Looking back at Theorem 3.1.1, we see that the QR algorithm actually gives a symmetric Schur decomposition when it is applied to a symmetric matrix. In this case, the T_k converge to a diagonal matrix whose entries are the eigenvalues. The columns of matrix $Q_{\prod}$ converge to the corresponding eigenvectors. The eigenvectors of L are the columns of the matrix $VQ_{\prod} \in \mathbb{R}^{n \times m}$. Therefore, the QR algorithm is a direct eigenvalue and eigenvector method for symmetric matrices. In Chapter 5 we will encounter a version of the Lanczos algorithm that yields an unsymmetric matrix. An extra step is required to compute the eigenvectors of an initial unsymmetric matrix T after the QR algorithm is applied. This step will now be discussed.

3.3 Back substitution

After p iterations of the QR algorithm, upper triangular matrix T_p has been obtained. Suppose that w_k is the eigenvector of T_p corresponding to the Ritz value θ_k. Then the eigenvector s_k of T_1 can be computed with

$$s_k = Q_{\prod} w_k, \tag{3.55}$$

where $Q_{\prod}$ is constructed with Algorithm 6. This section discusses the computation of eigenvector w_k of T_p. The computation boils down to solving a simple linear system.

The eigenvalues of the upper triangular matrix T_p are on its diagonal and are called the Ritz values. To compute the eigenvector w_k that corresponds to Ritz value θ_k, one needs to solve the following equation:

$$(T_p - \theta_k I) w_k = \mathbf{0}. \tag{3.56}$$

Now $\tilde{T}_p := T_p - \theta_k I$ has a zero entry on the diagonal. Suppose that diagonal element $\tilde{T}_{p(l,l)}$ is zero. An illustration of the situation is given in Figure 3.1.

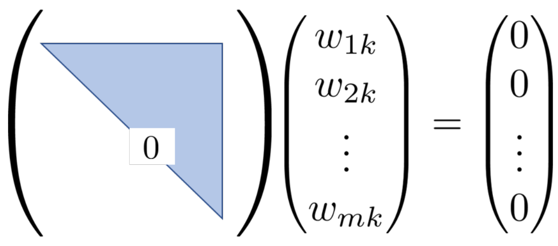


Figure 3.1: The linear system $\tilde{T}_p w_k = \mathbf{0}$ to compute eigenvector w_k corresponding to Ritz value θ_k.

Starting at the last row of $\tilde{T}_p$, we find that the entries $w_{mk}, \dots, w_{l+1,k}$ have to be zero. The system can be solved from bottom to top as follows

$$\begin{aligned} w_{mk}, \dots, w_{l+1,k} &= 0, \\ w_{lk} &= -\tilde{T}_{p(l-1,l-1)}, \\ w_{l-1,k} &= \tilde{T}_{p(l,l-1)}, \\ w_{ik} &= \frac{0 - \sum_{j=i+1}^{n} \tilde{T}_{p(i,j)} w_{jk}}{\tilde{T}_{p(i,i)}} \qquad \text{for } i \in \{l-2, \dots, 1\}. \end{aligned} \tag{3.57}$$

By choosing the desired Ritz values from the diagonal of T_p and solving the corresponding linear systems, the eigenvectors of T_p can be found. These eigenvectors can be used to compute approximations of the eigenvectors of the Laplacian L. Put the approximated eigenvectors in matrix W. The eigenvectors of L are the columns of the matrix $VQ_{\prod}W \in \mathbb{R}^{n \times m}$.

3.4 Summary

We end this chapter with a summary of the steps that will be taken to compute approximations of the eigenvectors of Laplacian L:

1. Decompose $L \in \mathbb{R}^{n \times n}$ with m iterations of the Lanczos algorithm into an orthogonal matrix $V \in \mathbb{R}^{n \times m}$ and a tridiagonal matrix $T \in \mathbb{R}^{m \times m}$.

2. Decompose $T \in \mathbb{R}^{m \times m}$ with p iterations of the QR algorithm into an orthogonal matrix $Q_{\prod} \in \mathbb{R}^{m \times m}$ and upper triangular matrix $T_p \in \mathbb{R}^{m \times m}$ with the eigenvalues of T on the diagonal.

3 a. When T is symmetric, the columns of $Q_{\prod}$ converge to the eigenvectors of T. The eigenvectors of L are the columns of the matrix $VQ_{\prod} \in \mathbb{R}^{n \times m}$.

3 b. When T is unsymmetric, the eigenvectors of T_p can be computed with back substitution. Put these eigenvectors in matrix W. The eigenvectors of L are the columns of the matrix $U = VQ_{\prod}W \in \mathbb{R}^{n \times m}$.

The discussed algorithms will be translated to the message space of the Paillier encryption scheme. Computations work differently in this domain, as will be explained in the next chapter.

Chapter 4

Computations on encrypted numbers

Data mining algorithms may be limited by privacy constraints, for example because datasets of different institutions are not allowed to be combined. Therefore, the ability to cluster a dataset into k clusters in a privacy preserving manner could broaden the applications in data mining. Let us first define the term *privacy preserving* [30]. In this thesis, *secure* and privacy preserving will be interpreted as synonyms.

Definition 4.0.1. *A function is privacy preserving when no party learns anything more than its prescribed output. In particular, the only information that should be learned about other parties' inputs is what can be derived from the output itself.*

In a privacy preserving spectral clustering algorithm, every data contributor should only learn the cluster index to which he or she belongs. Other information, such as the number of points in a cluster or the values of other contributors' data, should remain private. Therefore, a secure spectral clustering algorithm would be performed on encrypted data. This chapter will give an introduction to computing with encrypted numbers. First, an overview will be given of the parties that could participate in the algorithm. The Paillier encryption scheme will be discussed in Section 4.2. Only positive integers can be encrypted. Computations in the integer domain are the topic of Section 4.3. Sections 4.4 and 4.5 expound on the computations after encryption. *Secure multiparty computation* techniques will be discussed in which a function is computed by multiple parties over combined inputs while the inputs stay private.

Lastly, it remains to add that the discussed set-up and secure multiparty techniques are by no means meant to be the optimal choices. Such an ambition is beyond the scope of this thesis. The purpose of this chapter is to clarify the context to which the spectral clustering algorithm is translated. A good understanding of the challenges, possibilities and limitations of privacy preserving computation techniques is vital to a successful adaptation of the algorithm. Thus, this chapter lays the foundations for the choices that are made in Chapter 5.

4.1 Introduction and set-up

Multiple parties will be involved in a privacy preserving clustering algorithm. We will refer to the parties whose data is clustered as *users*. Without introducing a third party, user participation is required to process user data [11]. Chapters 2 and 3 have shown that both the k-means clustering step and the approximation of eigenvectors make use of iterative algorithms. Moreover, many steps are required to perform the entire spectral clustering algorithm, and the Laplacian grows with the number of users. We would like to spare the users from performing the computations themselves. Therefore, a *service provider* is introduced that has resources for storage and processing. After providing their data, the users do not have to participate (they are *offline*) until they receive their cluster index. An illustration of the foreseen set-up is given in Figure 4.1. The data contributors first encrypt their data before sending it to the service provider, so that their privacy is preserved. However, the data also needs to be clustered. We will see in Section 4.2 that certain computations are possible on encrypted data. For other computations, the data needs to be decrypted. The data can only be decrypted with a secret *decryption key*. Since the service provider is not allowed to learn the values of the user data, it cannot have the decryption key. Inspired by [10], a *privacy service provider* (PSP) that owns this private decryption key is introduced. The service provider and PSP perform an interactive protocol to cluster the encrypted user data. The service provider and PSP will be referred to as the *computing parties*. After the privacy preserving spectral clustering algorithm, the service provider sends encrypted cluster indices to the data contributors. These indices can be decrypted with the help of the PSP.

This research is carried out in a *semi-honest* model in which all the parties follow the protocol that they agreed on, but which also store the intermediate iteration results [15]. The parties may use these results to gain more knowledge. The service provider and PSP are assumed to be *non-colluding parties*, which means that they are not allowed to collaborate to learn more about the data. If they colluded, they could obtain all the values of the data points, since the service provider stores the encrypted data and the PSP owns the decryption key.

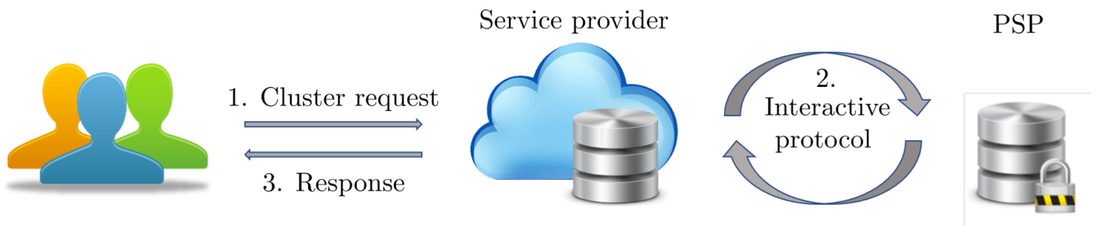


Figure 4.1: Users can request a clustering result to a service provider, which only receives encrypted data. A privacy service provider (PSP) who owns the decryption key is required to facilitate the computations.

4.2 Paillier encryption

The computing parties cannot know the content of the data, so they will work on encrypted data. Therefore, an encryption scheme that allows computations to be performed on encrypted data is necessary.[1] Such schemes are called *homomorphic encryption schemes*. In 2009, Craig Gentry showed that a fully homomorphic encryption scheme is realizable [14]. Let $\mathcal{E}$ be a fully homomorphic encryption scheme and let $m_1, \ldots, m_n$ be data points that should be secret. Gentry's encryption scheme allows encrypted messages $\mathcal{E}(m_1), \ldots, \mathcal{E}(m_n)$ to be evaluated with any desired function f. The encrypted result $\mathcal{E}(f(m_1, \ldots, m_n))$ can be computed by anyone without leaking any information about $m_1, \ldots, m_n$. After decryption, the evaluation $f(m_1, \ldots, m_n)$ is obtained. However, for the moment the computational complexity is impractical to use for mining applications of large datasets. The communication complexity and memory costs are too high. Therefore, secure multiparty algorithms that make use of homomorphic encryption generally use encryption schemes that are either additively homomorphic or multiplicatively homomorphic, i.e. either the addition or multiplication of encrypted messages is allowed.

The Paillier encryption scheme has a proven track record in privacy preserving data mining [38]. Paillier encryption is often used in secure multiparty computation because of its computational efficiency and the relatively small size of encrypted messages. Suppose that we want to encrypt a *message* m. Define $N = pq$ to be the product of two large prime numbers p and q. The encryption

[1]Many other secure multiparty computation techniques can be applied, such as garbled circuits or secret sharing [43, 55]. Homomorphic encryption was chosen because this thesis builds upon previous work by Thijs Veugen. Other secure multiparty computation techniques also work on positive integers so the current research is partially applicable to these techniques.

function is defined as

$$\begin{aligned} E_g : \mathbb{Z}_N \times \mathbb{Z}_N^* &\rightarrow \mathbb{Z}_{N^2}^* \\ (m, r) &\mapsto g^m r^N \mod N^2. \end{aligned} \tag{4.1}$$

The messages are mapped from the *message space* to the *ciphertext space* when they are encrypted. We will refer to unencrypted messages as *plaintext* messages. The message space is $\mathbb{Z}_N$.[2] The finite field $\mathbb{Z}_N$ consists of the following N elements:

$$\mathbb{Z}_N = \{0, 1, \dots, N-1\}. \tag{4.2}$$

Modular arithmetic is used on $\mathbb{Z}_N$. Therefore, computations on messages have to abide certain rules. This is the subject of Section 4.3. After encryption, the message has become a *ciphertext* in $\mathbb{Z}_{N^2}^*$. This set consists of the integers that are *coprime* to N^2, i.e. they do not share a divisor larger than 1 with N^2:

$$\mathbb{Z}_{N^2}^* = \{z | z \in \mathbb{Z}, 0 < z < N^2, gcd(z, N^2) = 1\}. \tag{4.3}$$

To encrypt a message $m \in \mathbb{Z}_N$, one uses the *public key* (N, g). We denote the encryption of the message by $[m]$. A new random number $r \in \mathbb{Z}_N^*$ is generated for every encryption. This ensures that two encryptions of the same message have a different value. The security of the Paillier cryptosystem is based on the property that given $[m] = g^m r^N \mod N^2$ and the public key (N, g), there does not exist an algorithm in polynomial time to find the original value of m [50].

Anyone can encrypt data using the public key (N, g). The *secret key* $\lambda = lcm(p-1, q-1)$ is necessary to be able to decrypt a ciphertext:

$$m = \frac{\mathcal{L}(c^\lambda \mod N^2)}{\mathcal{L}(g^\lambda \mod N^2)} \mod N. \tag{4.4}$$

Here, the function $\mathcal{L}$ is defined for $u \in \{0 < u < N^2 | \ u = 1 \mod N\}$ as

$$\mathcal{L}(u) = \frac{u-1}{n}. \tag{4.5}$$

Details of the correctness of the decryption can be found in [38, 50]. An encryption scheme in which different keys are used for encryption and decryption is called an *asymmetric scheme.* By contrast, in a *symmetric* encryption scheme

[2] Actually, the message space is $\mathbb{Z}_N^* = \{z | z \in \mathbb{Z}, 0 < z < N, gcd(z, N) = 1\}$. However, the probability of finding a divisor of N is negligible and would break the cryptosystem. Therefore, we call the Paillier message space $\mathbb{Z}_N$ for notational convenience. The set $\mathbb{Z}_N^*$ is a finite field on which addition, subtraction, multiplication and division are defined. We assume $\mathbb{Z}_N$ to satisfy these conditions for now.

the same key is used for encryption and decryption and therefore this key needs to be secret. The asymmetric property of the Paillier encryption scheme makes it suitable for a setting in which multiple parties participate, since the encryption key can be distributed publicly.

The Paillier encryption scheme is *additively homomorphic*, which means that the scheme allows for the following computations with messages $m_1, m_2 \in \mathbb{Z}_N$ and constant $k \in \mathbb{Z}_N$:

$$E_g(m_1) \cdot E_g(m_2) = E_g(m_1 + m_2), \tag{4.6}$$
$$E_g(m_1)^k = E_g(k \cdot m_1), \tag{4.7}$$
$$E_g(m_1)^{-1} = E_g(-m_1). \tag{4.8}$$

Let us now consider an example of how an additively homomorphic encryption scheme such as Paillier encryption can be used for secure two-party computation. Suppose that Alice and Bob want to compute the inner product of their vectors v_A and v_B. Alice has the secret decryption key. Alice encrypts the entries of her vector separately to obtain entries $[v_{A,i}]$ and sends these encryptions to Bob. Bob may then perform the following computations by making use of his own plaintext vector v_B and the additively homomorphic properties of the Paillier cryptosystem:

$$[\langle v_A, v_B \rangle] = [\sum_{i=1}^{n} v_{A,i} \cdot v_{B,i}] = \prod_{i=1}^{n} [v_{A,i} \cdot v_{B,i}] = \prod_{i=1}^{n} [v_{A,i}]^{v_{B,i}}. \tag{4.9}$$

Bob never learns the entries of v_A. Alice can decrypt the obtained encrypted inner product and can either share the result with Bob or keep it to herself.

Linear operations such as the inner product can be computed in the encrypted domain by one party. However, when we want to compute non-linear operations such as the product $[a \cdot b]$ of two encrypted values $[a]$ and $[b]$, other computation protocols are necessary. These protocols will be discussed in Section 4.5. First, the operations that can be performed in the message space will be discussed.

4.3 Computations in the message space

The message space of the Paillier cryptosystem is $\mathbb{Z}_N$, which is defined as

$$\mathbb{Z}_N = \{z | z \in \mathbb{Z}, 0 \leq z < N\} = \{0, 1, \dots, N-1\}. \tag{4.10}$$

Therefore, in order to encrypt the entries of the matrices and vectors in the algorithms that have been discussed so far, these entries should be elements of

$\mathbb{Z}_N$. When spectral clustering is applied in the plaintext domain, entries can be elements of $\mathbb{R}$. Arithmetic operations in $\mathbb{Z}_N$ and $\mathbb{R}$ are different. This section will explain how data can be represented and processed in $\mathbb{Z}_N$.

4.3.1 Arithmetic operations

A congruence relation is introduced on $\mathbb{Z}_N$ in order to relate the operations as defined in $\mathbb{Z}$ to operations in $\mathbb{Z}_N$. The congruence relation $a \equiv b \pmod{N}$ is the remainder of the Euclidean division of a by N. This congruence relation is compatible with addition, subtraction and multiplication. If $a_1 \equiv b_1 \pmod{N}$ and $a_2 \equiv b_2 \pmod{N}$, then

$$\begin{aligned} a_1 + a_2 &\equiv b_1 + b_2 \pmod{N}, \\ a_1 - a_2 &\equiv b_1 - b_2 \pmod{N}, \\ a_1 a_2 &\equiv b_1 b_2 \pmod{N}. \end{aligned}$$

Section 4.2 has shown that computations can be performed on an encrypted message $[m]$. Care should be taken that the resulting message m' does not become larger than $N - 1$, since it will be mapped to the value $m' \mod N$. This phenomenon is called *overflow* and is not acceptable if we want to be able to derive the original message with decryption. Thankfully, N is chosen to have a large size of 1024 or 2048 bits, so overflow does not occur too soon.

The message space $\mathbb{Z}_N$ is not closed under division: if $a, b \in \mathbb{Z}$, the division $\frac{a}{b}$ need not be an integer. Therefore an integer division operation is defined, in which the remainder is discarded:

Definition 4.3.1. *Let $a, b \in \mathbb{Z}$. The integer division $a \div b$ is defined as the integer q such that $a = qb + r$ with remainder $r \in \mathbb{Z}$, where $0 \leq r < b$.*

In modular arithmetic, the division operation is sometimes interpreted as multiplication by the multiplicative inverse. We stress here that division as defined in Definition 4.3.1 is not to be confused with multiplication by the multiplicative inverse. Division as multiplication by the multiplicative inverse may not even exist if not all elements in $\mathbb{Z}_N$ have a multiplicative inverse [2]. Suppose that the multiplicative inverse $b^{-1} \in \mathbb{Z}_N$ of a certain $b \in \mathbb{Z}_N$ exists, so $bb^{-1} = 1$. The following example shows that $a/b = ab^{-1}$ differs from $a \div b$. In $\mathbb{Z}_{21}$, the multiplicative inverse of 2 is 11 since $2 \cdot 11 = 22 \equiv 1 \pmod{21}$. So $5/2 = 5 \cdot 2^{-1} = 5 \cdot 11 \equiv 13 \pmod{21}$. However, when using Definition 4.3.1, the result $5 \div 2 = 2$ is obtained.

Finally, the integer square root is defined as follows:

Definition 4.3.2. *The integer division operation $\lfloor \sqrt{n} \rfloor \in \mathbb{Z}_N$ is the largest $m \in \mathbb{Z}_N$ such that $m^2 \leq n$.*

For convenience of notation, the integer square root $\lfloor\sqrt{n}\rfloor$ will be denoted with $\sqrt{n}$. The context will clarify the domain in which the square root operation is performed.

4.3.2 Bit length

When performing computations on values in the message space $\mathbb{Z}_N$, the messages may comprise a growing number of bits. Let us first define the *bit length* of a number.

Definition 4.3.3. *A number $b \in \mathbb{Z}_N$ is said to have a bit length of ℓ bits when $2^{\ell-1} \leq b \leq 2^{\ell} - 1$. The number of bits can be calculated from b by*

$$\ell = \lfloor \log_2(b) \rfloor + 1. \tag{4.11}$$

In this thesis, the terms bit length and bit size will be used interchangeably. *Overflow* occurs when the result of a computation is too large to be represented by the alloted number of bits of N. Let us determine the required bit space after an addition or multiplication operation. Subtraction and division decrease a number, so overflow is not a problem. Suppose that b_1 has ℓ_1 bits and b_2 has ℓ_2 bits. The addition $b_1 + b_2$ has a maximum size of $\max(\ell_1, \ell_2) + 1$ bits. The multiplication $b_1 b_2$ is in the bit size range $[\ell_1 + \ell_2 - 1, \ell_1 + \ell_2]$. Therefore, it should be taken into account that there is a limit on the number of additions and multiplications that can be performed on encrypted data. However, since N is generally chosen to comprise 1024 or 2048 bits, this only happens after many operations.

4.3.3 Representation of numbers

When user data consist of real-valued numbers, the inputs have to be represented as messages in $\mathbb{Z}_N$. Suppose that the user data is a d-dimensional vector. The Euclidean space $\mathbb{R}^d$ is approximated via $\mathbb{Q}^d$ to $\mathbb{Z}_N^d$ and the input of the cryptographic protocol is restricted to this range.[3] From this point on, we consider numbers to be represented explicitly in the binary system. This is to remain consistent with the literature, since privacy preserving protocols regularly use bitwise computations.

We will first discuss how to represent $\mathbb{Q}$ in $\mathbb{Z}_{\langle\ell\rangle}$, the set of ℓ-bit signed integers:

$$\mathbb{Z}_{\langle\ell\rangle} = \left\{ x \in \mathbb{Z} \mid -2^{\ell-1} < x \leq 2^{\ell-1} \right\}. \tag{4.12}$$

[3]This discretization is natural in computer science since, due to memory constraints, the use of a finite number of decimals is a standard procedure [2]. Therefore, we can consider all user inputs to be in $\mathbb{Q}^d$.

Fixed point arithmetic will be used to represent fractions as signed integers. In fixed point arithmetic, a fixed number of bits is reserved for the fractional part [20]. Define the set

$$\mathbb{Q}_{\langle \ell \rangle, f} = \{x \in \mathbb{Q} | \; x = \bar{x} \cdot 2^{-f}, \bar{x} \in \mathbb{Z}_{\langle \ell \rangle}\}. \tag{4.13}$$

Suppose that a precision of f digits of the fractional part is desired. By multiplying numbers in $\mathbb{Q}_{\langle \ell \rangle, f}$ by 2^f, a signed integer with a maximum of ℓ bits is obtained. The map ϕ from $\mathbb{Q}_{\langle \ell \rangle, f}$ to $\mathbb{Z}_{\langle \ell \rangle}$ is defined as follows:

$$\begin{aligned} \phi : \mathbb{Q}_{\langle \ell \rangle, f} &\longrightarrow \mathbb{Z}_{\langle \ell \rangle}, \\ x &\longmapsto 2^f \cdot x. \end{aligned} \tag{4.14}$$

Suppose that the number $N = pq$ that defines the message space comprises $\ell_N := \lfloor \log_2 N \rfloor$ bits. Since a fixed number of f bits is reserved for the fractional part of a message, $\ell_N - f$ bits can be used to represent the integer part without causing overflow.

The map ϕ preserves addition and subtraction. So let $\odot \in \{+, -\}$. The following relation holds:

$$\phi^{-1}(z) = x \odot y \text{ if } z = \phi(x) \odot \phi(y). \tag{4.15}$$

However, multiplication requires more attention. Let $x, y \in \mathbb{Q}_{\langle \ell \rangle, f}$, then

$$z := \phi(x) \cdot \phi(y) = 2^{2f} xy = 2^f \phi(xy). \tag{4.16}$$

So the result is scaled twice while it should only be scaled once. In order to obtain a correctly scaled result, *truncation* has to be applied. A truncation operation discards the f least significant bits as follows:

$$\text{Trunc } (z) = \frac{z}{2^f} = \frac{2^f \phi(xy)}{2^f} = \phi(xy) \in \mathbb{Z}_{\langle k \rangle}. \tag{4.17}$$

When two fixed point numbers are divided using integer division, the scaling factor is eliminated:

$$\phi(x) \div \phi(y) = (2^f x) \div (2^f y) = x \div y. \tag{4.18}$$

Therefore, in the division of numbers in fixed point arithmetic, the enumerator is always scaled first:

$$\phi(x \div y) = (2^f \phi(x)) \div \phi(y) = \phi(2^f x) \div \phi(y). \tag{4.19}$$

We now have a means to represent fractions as signed integers. However, only integers in $\mathbb{Z}_N$ can be encrypted. Therefore, a way to represent negative numbers in $\mathbb{Z}_N$ is required. The following injective map ψ can encode signed integers as positive integers when $N > 2^\ell$:

$$\psi : \mathbb{Z}_{\langle \ell \rangle} \longrightarrow \mathbb{Z}_N, \tag{4.20}$$

$$x \longmapsto x \mod N. \tag{4.21}$$

Its inverse $\psi^{-1} : \mathbb{Z}_N \to \mathbb{Z}_{\langle \ell \rangle}$ is given by

$$\psi^{-1}(x) = \begin{cases} x, & \text{if } x < N/2, \\ x - N, & \text{otherwise.} \end{cases} \tag{4.22}$$

Informally stated, the upper half of the domain $\mathbb{Z}_N$ will be taken to represent the negative integers of maximum bit length ℓ. The map ψ preserves addition, subtraction, multiplication and integer division. So let $\odot \in \{+, -, \cdot, \div\}$. The following relation holds:

$$\psi^{-1}(z) = x \odot y, \quad \text{if } z = \psi(x) \odot \psi(y). \tag{4.23}$$

We finish this section by defining $\xi := \psi \circ \phi$ with ψ and ϕ as defined in Definitions 4.21 and 4.14 respectively. The function ξ will be used to represent elements of $\mathbb{Q}_{\langle \ell \rangle, f}$ in $\mathbb{Z}_N$. Addition and subtraction are preserved by ξ. Multiplication and division require the adjustments for fixed points arithmetic that were explained in (4.17) and (4.19).

4.4 Computations in the ciphertext space

After encryption with the Paillier encryption function (4.1), the messages are mapped to the ciphertext space $\mathbb{Z}^*_{N^2}$. This section will give an introduction to computing on ciphertexts. These computations are heavier due to the large size of the messages. The complexity of basic computations and techniques to make the computations more efficient will be discussed. It will be investigated later on whether these techniques can be used in a secure spectral clustering algorithm. We have seen that at least two parties are required to perform the secure computations: the service provider and privacy service provider. These two parties will be given the more general names Alice and Bob in the upcoming sections, to indicate that these protocols are not limited to the current context. Bob owns the decryption key.

4.4.1 Complexity of operations

In order to compare the protocols in Section 4.5, they will be assessed in terms of the number of operations on encrypted numbers [11, 12]. The complexity of computations on plaintext numbers is negligible in comparison with computations on encrypted numbers. This due to the size of the encrypted numbers, which have a maximum bit length of $\ell_c := 2\ell_N$. Moreover, addition and multiplication over mod N on plaintext numbers become multiplication and exponentiation over mod N^2 on encrypted numbers. The complexity of a negation is approximately equal to the complexity of a multiplication. The number of negations, multiplications, exponentiations, encryptions and decryptions will be counted in the complexity analysis. Furthermore, the privacy preserving protocols will require the cooperation of multiple parties. To give an indication of the communication complexity of secure multiparty protocols, the back and forth communication of encrypted messages between two parties will be referred to as a *round*. The *bandwidth* of a protocol is the maximum amount of data to be transmitted at a certain point in time. The *total bandwidth* is the total amount of data that needs to be transmitted. The generation of the public and private keys and of the random numbers can be computed beforehand and their complexity will therefore not be taken into account.

4.4.2 Data packing

An encrypted message is an element of $\mathbb{Z}^*_{N^2}$ and is therefore a very large integer. This is true for any original message size. This is illustrated in Example 1.

Example 1.

$$\begin{aligned} m &\xmapsto{E_g} [m] \\ 88 &\xmapsto{E_g} 270541034898823582508234719 2\ldots \\ 17083762124792945 &\xmapsto{E_g} 1832994261736908802356782035\ldots \end{aligned}$$

Since operations on encrypted data are expensive, *data packing* can be used to improve the protocol in terms of efficiency. Data packing is a technique that puts multiple entries that need to be processed into one message. Every entry fills a *compartment* of the message. This is illustrated in Example 2, in which five messages are packed into one encryption.

Example 2.

$$\boxed{88 \mid 24 \mid 57 \mid 50 \mid 16} \xmapsto{E_g} 9368217908688528064048285943\ldots$$

The use of data packing can be motivated by two observations during the execution of an algorithm [10]:

1. The same operations are performed multiple times, e.g. the multiplication of a vector's entries by a constant.

2. The Paillier message space $\mathbb{Z}_N$ is larger than the necessary message bit space as the messages are relatively small.

Suppose that we want to add an encrypted constant $[\alpha]$ to an encrypted vector $[w]$. Instead of encrypting the entries of $w = (w_1, \ldots, w_k)^T$ separately and multiplying these entries with constant $[\alpha]$, we can pack the entries of w in one encryption and multiply the packed encrypted vector by $[\alpha]$. Suppose that the entries of w are represented with t bits. Some operations, such as addition by α, cause an increase in bit length of the entries of w. Therefore, suppose that b more bits need to be reserved as "expansion space". The message space $\mathbb{Z}_N$ comprises ℓ_N bits. The entries of w can be concatenated into one large number with a maximum bit length ℓ_N as follows:

$$\begin{aligned} w^{\text{Packed}} &= \sum_{i=0}^{k-1} w_{i+1} \cdot (2^{t+b})^i && (4.24) \\ &= \boxed{w_k \mid \ldots \mid w_1}\,. && (4.25) \end{aligned}$$

We can use a single encryption to pack this vector. The α can also be concatenated such that every compartment of the packed α^{Packed} contains α:

$$\begin{aligned} \alpha^{\text{Packed}} &= \sum_{i=0}^{k-1} \alpha \cdot (2^{t+b})^i && (4.26) \\ &= \boxed{\alpha \mid \ldots \mid \alpha}\,. && (4.27) \end{aligned}$$

The packed encrypted constant $[\alpha]$ can be added to every entry of w with one multiplication:

$$[w^{\text{Packed}}] \cdot [\alpha^{\text{Packed}}] = [w_k + \alpha \mid \ldots \mid w_1 + \alpha]\,. \tag{4.28}$$

Since the communication complexity of a computation is mainly determined by the exchange of Paillier encrypted messages, it is a large advantage to reduce the number of encryptions through data packing. Of course, there is a limitation to the number of entries that can be packed in one encryption. The number of entries that fit in one encryption is $\ell_N/(t+b)$, where b depends on the desired operations that are performed on the encrypted data. It should be noted that data packing causes an increase in the size of a message. The message can have a bit length as large as ℓ_N. Therefore, computations on packed plaintext numbers are not necessarily negligible in terms of complexity. In this research, the bit lengths of numbers during an algorithm will be mentioned in order to investigate whether data packing can be applied.

4.4.3 Blinding numbers

Due to the additively homomorphic property of Paillier, linear operations can be performed on encrypted numbers. To perform non-linear operations, we need to decrypt the numbers at some point. Therefore, we need a method to hide the values of numbers even after decryption. Such a method is *additive blinding* or *additive obfuscation*. Suppose that Alice has an encrypted message $[m]$ and chooses a random number $r \in \mathbb{Z}_N$. Alice can hide the value of m as follows:

$$[m] \cdot [r] \mod N^2 = [m + r]. \tag{4.29}$$

Now Alice can send $[m+r]$ to Bob, who owns the decryption key. We want to ensure that $m + r$ does not provide any information about the value of m, i.e. that $m+r$ appears to be a random number. Therefore, random numbers r should be κ bits larger than m in order to guarantee that $m+r$ hides m well enough. The parameter κ is called the *security parameter*. A value of $\kappa = 80$ is often applied [48].

We now have the preliminary knowledge to investigate the required computations on encrypted numbers.

4.5 Privacy preserving computation protocols

Chapter 3 introduced the numerical methods that will be used to approximate eigenvectors. It became clear from Algorithms 3, 5 and the back substitution process that scalar multiplication, inner products, matrix products, divisions and square roots to compute vector norms are required. This section gives examples of privacy preserving protocols that could be used to implement these numerical methods securely. These protocols were found in related literature on privacy preserving data mining. The authors proved the protocols to be secure under the semi-honest adversary model. The composition of secure protocols in the semi-honest adversary model is again secure in this model, as was proven by Canetti [3]. We will see that secure division and secure square root protocols are computationally heavy. The complexity of protocols influences the applicability of a secure algorithm. Therefore, the upcoming section is vital to understand the considerations that will be made in Chapter 5.

In the following sections, ℓ_c is the bit size of a ciphertext. Recall that the relation $\ell_c = 2\ell_N$ holds. For a secure cryptosystem, ℓ_N has to be chosen as 1024 or 2048 bits. Moreover, we will take some notational freedom in the encryption of vectors and matrices. An encrypted vector v and encrypted matrix M will be denoted with $[v]$ and $[M]$ respectively. They are encrypted entry-wise, so the encrypted vector and matrix are filled with separately encrypted entries.

4.5.1 Scalar multiplication

Suppose that Alice has $[a]$ and $[b]$, and Bob owns the decryption key SK. Alice wants to know the encrypted value $[a \cdot b]$ without Bob learning either a or b. A secure multiplication protocol that achieves these goals is given in Protocol 2 [10]. Suppose that a has bit length ℓ_a and b has bit length ℓ_b. First, Alice obfuscates $[a]$ and $[b]$ with two random values r_1 and r_2 of size $\kappa + \ell_a$ and $\kappa + \ell_b$ respectively, where κ is the security parameter. The obfuscated values $[\tilde{a}]$ and $[\tilde{b}]$ are sent to Bob, who can safely decrypt them. Bob computes the product $\tilde{a} \cdot \tilde{b}$ and sends the encrypted $[\tilde{a} \cdot \tilde{b}]$ back to Alice. Alice can remove the obfuscation with additive homomorphic properties as follows:

$$\begin{aligned} [a \cdot b] &= [\tilde{a} \cdot \tilde{b} - ar_2 - br_1 - r_1 r_2] \qquad (4.30) \\ &= [\tilde{a} \cdot \tilde{b}] \cdot [a]^{-r_2} \cdot [b]^{-r_1} \cdot [-r_1 r_2]. \qquad (4.31) \end{aligned}$$

Protocol 2: $[ab] \leftarrow \mathsf{Mult}([a], [b])$ [10]

```
Input  : A: $[a], [b]$, random $r_1, r_2$.
         B: decryption key $SK$.
Output: A obtains $[ab]$.
1 $A|$  $[\tilde{a}] \leftarrow [a] \cdot [r_1]$
2 $A|$  $[\tilde{b}] \leftarrow [b] \cdot [r_2]$
3 $A$ sends $[\tilde{a}]$ and $[\tilde{b}]$ to $B$.
4 $B|$  $\tilde{a} \leftarrow \mathsf{Decrypt}([\tilde{a}])$
5 $B|$  $\tilde{b} \leftarrow \mathsf{Decrypt}([\tilde{b}])$
6 $B|$  $[\tilde{a} \cdot \tilde{b}] \leftarrow \mathsf{Encrypt}(\tilde{a} \cdot \tilde{b})$
7 $B$ sends $[\tilde{a} \cdot \tilde{b}]$ to $A$.
8 $A|$  $[a \cdot b] \leftarrow [\tilde{a} \cdot \tilde{b}] \cdot [a]^{-r_2} \cdot [b]^{-r_1} \cdot [-r_1 r_2]$
```

Protocol 2 requires one round with a bandwidth of $2\ell_c$. The total bandwidth is $3\ell_c$. In total 2 multiplications are required to obfuscate a and b, and 3 multiplications, 2 negations and 2 exponentiations to recover $[ab]$ from $[\tilde{a}\tilde{b}]$. Moreover, 2 decryptions and 1 encryption are performed.

4.5.2 Inner product

The privacy preserving inner product is an extension of the privacy preserving multiplication protocol. Suppose that Alice holds two entry-wise encrypted vectors $[v]$ and $[w]$, both of length n. Bob holds the decryption key SK. To compute the encrypted inner product $[\langle v, w \rangle]$, the vectors are first obfuscated. Alice generates two random vectors r_v and r_w, whose entries are at least κ bits larger than the entries of v and w respectively. Alice sends the obfuscated encrypted vectors

$[\tilde{v}] = [v + r_v]$ and $[\tilde{w}] = [w + r_w]$ to Bob, who can decrypt them to compute the obfuscated inner product:

$$\langle \tilde{v}, \tilde{w} \rangle = \sum_{i=1}^{n} \tilde{v}_i \tilde{w}_i. \tag{4.32}$$

The encrypted resulted is sent back, after which the obfuscation can be removed by Alice. She makes use of the following relation:

$$\langle \tilde{v}, \tilde{w} \rangle = \sum_{i=1}^{n} (v_i + r_{v_i})(w_i + r_{w_i}) \tag{4.33}$$

$$= \sum_{i=1}^{n} (v_i w_i + v_i r_{w_i} + w_i r_{v_i} + r_{v_i} r_{w_i}). \tag{4.34}$$

So, Alice performs the following computations:

$$[\langle v, w \rangle] = [\langle \tilde{v}, \tilde{w} \rangle - \sum_{i=1}^{n} v_i r_{w_i} - \sum_{i=1}^{n} w_i r_{v_i} - \sum_{i=1}^{n} r_{v_i} r_{w_i}] \tag{4.35}$$

$$= [\langle \tilde{v}, \tilde{w} \rangle] \cdot \prod_{i=1}^{n} [v_i]^{-r_{w_i}} \cdot \prod_{i=1}^{n} [w_i]^{-r_{v_i}} \cdot [-\sum_{i=1}^{n} r_{v_i} r_{w_i}]. \tag{4.36}$$

The privacy preserving inner product protocol is given in Protocol 3 [12].

Protocol 3: $[\langle v, w \rangle] \leftarrow \mathsf{Inner}([v], [w])$[12]

```
Input  : A: Entry-wise encrypted vectors $[v], [w]$, random vectors $r_v, r_w$.
         B: Decryption key $SK$.
Output: A obtains $[\langle v, w \rangle]$.
1 $A|$  $[\tilde{v}_i] \leftarrow [v_i] \cdot [r_{v_i}]$ for $i \in 1, \ldots, n$
2 $A|$  $[\tilde{w}_i] \leftarrow [w_i] \cdot [r_{w_i}]$ for $i \in 1, \ldots, n$
3 $A$ sends $[\tilde{v}]$ and $[\tilde{w}]$ to $B$.
4 $B|$  $\tilde{v}_i \leftarrow \mathsf{Decrypt}([\tilde{v}_i])$ for $i \in 1, \ldots, n$
5 $B|$  $\tilde{w}_i \leftarrow \mathsf{Decrypt}([\tilde{w}_i])$ for $i \in 1, \ldots, n$
6 $B|$  $[\langle \tilde{v}, \tilde{w} \rangle] \leftarrow \mathsf{Encrypt}(\langle \tilde{v}, \tilde{w} \rangle)$
7 $B$ sends $[\langle \tilde{v}, \tilde{w} \rangle]$ to $A$.
8 $A|$  $[\langle v, w \rangle] \leftarrow [\langle \tilde{v}, \tilde{w} \rangle] \cdot \prod_{i=1}^{n} [v_i]^{-r_{w_i}} \cdot \prod_{i=1}^{n} [w_i]^{-r_{v_i}} \cdot [-\sum_{i=1}^{n} r_{v_i} r_{w_i}]$
```

One round of communication is required for Protocol 3 with a bandwidth of $2n\ell_c$. The total bandwidth is $(2n + 1)\ell_c$ because two encrypted vectors and one encrypted inner product are communicated. The computational complexity is $2n$ decryptions, 1 encryption, $2n$ multiplications to obfuscate the vectors and $2n$ exponentiations and negations and $2(n - 1) + 3$ multiplications to remove the obfuscation.

4.5.3 Secure matrix product

Suppose that Alice has two entry-wise encrypted matrices $[M_1]$ and $[M_2]$ of size $n \times n$.[4] Bob owns the decryption key SK. Since the entry-wise encrypted product $[M_1 M_2]$ can be computed with n^2 secure inner products, an explicit protocol for the secure matrix product will not be given. An improvement of the product of two Paillier-encrypted matrices is a topic for further research. We will only state the complexity of the secure matrix product that makes use of n^2 secure inner products as were presented in Protocol 3. One round of communication is necessary with a bandwidth of $2n^2\ell_c$ for the communication of two encrypted matrices. Bob performs $2n^2$ decryptions and n^2 encryptions. The inner products cost $2n^3$ exponentiations and negations and $n^2(2n+1)$ multiplications in total.

4.5.4 Division protocols

Many divisions are performed in the approximation of eigenvectors, as was shown in Chapter 3. Even on plaintext numbers, the division operation is more complex than addition, subtraction and multiplication, since an iterative algorithm is generally used to approximate the reciprocal of a number [52]. In the previous sections it became clear that computations in the encrypted domain are computationally more heavy than the same computations in the plaintext domain. This is certainly true for division, which is a relatively expensive operation in the encrypted domain [48]. Moreover, the division operation will be much more computationally intensive when the divisor d is encrypted than when d is public. The aim of this section is to give an intuition for this difference, since it will be of great importance for the translation of the spectral clustering algorithm to the encrypted domain. First, division by a public divisor d will be discussed.

Secure integer division with public divisor

Suppose that Alice wants to divide an encrypted number $[x]$ by a public divisor d. Thijs Veugen has designed division protocols in which either an exact integer division $[xd] = [x \div d]$, or approximate integer division $[xd] \in \{[x \div d], [x \div d + 1]\}$ is computed [48]. In exact integer division an additional comparison protocol has to be performed, which can be explained with the following example.

Example 3. Suppose that Alice wants to divide $[x] = [23]$ by $d = 5$. Bob owns the decryption key. She strives to find the result $x \div d = 4$ in a privacy preserving manner, so Bob cannot learn the value of x. Therefore, Alice blinds the value x

[4]For simplicity, only the product of two equally sized matrices is considered. The secure matrix product for matrices of general sizes $m \times k$ and $k \times n$ follows in an analogous manner.

with $r = 13$ and sends it to Bob.[5] The value $[z] = [x + r] = [36]$ is decrypted by Bob and he computes $z \div d$ in plaintext. Bob encrypts the result and sends it back to Alice. Alice knows how to correct the result $[z \div d]$. Now

$$\begin{aligned}[x \div d] &= [(z \div d) - (r \div d)] \\ &= [(36 \div 5) - (13 \div 5)] \\ &= [7 - 2] = [5].\end{aligned}$$

The subtraction operation can be performed with the additively homomorphic property of Paillier encryption. However, actually $x \div d = 23 \div 5 = 4$, so the additive blinding has influenced the division result. This has happened because the remainders of 23 and 13 after integer division by 5 add up to at least 5.

Example 3 showed that additive blinding can influence the division result. This occurs when

$$(x \mod d) + (r \mod d) \geq d. \tag{4.37}$$

Therefore, this inequality needs to be checked in order to perform exact division. An inequality can be checked in the encrypted domain with a *comparison protocol*, but this makes computations significantly more intensive. For now, it is assumed that approximate integer division is exact enough since the numbers are scaled with fixed point arithmetic before division. When more precision is required, more bits can be reserved to store the remainder of a division. The protocol for approximate integer division with a public divisor is given in Protocol 4. Note that there is a constraint on x. The sum $x + r$ should not overflow the message space $\mathbb{Z}_N$. Moreover, the additive blinding should be of at least size $\log_2 x + \kappa$ to be effective. Therefore, the size of x has to be constrained by $\log_2 x < \log_2 N - \kappa$. This is equivalent to the constraint $0 \leq x < N \cdot 2^{-\kappa}$.

Protocol 4: $[xd] \in \{[x \div d], [x \div d + 1]\} \leftarrow \mathsf{PubDiv}([x], d)$ [48]
Constraints: $0 \leq x < N \cdot 2^{-\kappa}$ and $0 < d < N$.

```
Input  : A: $[x], d$, random $r$ of size $\log_2 N - 1$.
         B: $d$, decryption key $SK$.
Output: A obtains $[xd] \in \{[x \div d], [(x \div d) + 1]\}$.
1 $A|$  $[z] \leftarrow [x] \cdot [r]$
2 $A$ sends $[z]$ to $B$.
3 $B|$  $z \leftarrow \mathsf{Decrypt}([z])$
4 $B|$  $[z \div d] \leftarrow \mathsf{Encrypt}(z \div d)$
5 $B$ sends $[z \div d]$ to $A$.
6 $A|$  $[xd] \leftarrow [z \div d] \cdot [-(r \div d)]$
```

[5]Actually, r should be at least κ bits larger than x. In this example, x is 5 bits long so with $\kappa = 80$, r should at least be 2^{84}. However, the example only means to illustrate the idea of additive blinding so we take a small value of r.

Protocol 4 requires one round with a bandwidth of $2\ell_c$. Furthermore, 2 multiplications, 1 encryption and 1 decryption are required.

In Section 4.3 the truncation operation was defined in (4.17) to correct the multiplication of two numbers in fixed point arithmetic. We note here that the truncation operation Trunc is the same as division by a public divisor, in which the divisor is a power of 2.

In Protocol 4, the divisor d is public. The divisor may contain valuable information and therefore we would like to be able to divide $[x]$ by an encrypted divisor d. An intermediate step would be to make divisor d known only to one party. Another protocol by Thijs Veugen ensures that the divisor is only known to Bob who has the decryption key [48]. Bob has to perform an extra encryption and Alice has to perform an extra exponentiation. There is an additional communication step from Bob to Alice. We will now discuss division by an encrypted divisor.

Secure integer division with encrypted divisor

In order to perform a computation in which the privacy is perfectly preserved, the divisor should be encrypted for both parties. An efficient solution that does not make use of an iterative algorithm was proposed by Dahl, Ning and Toft to compute $n \div d$ for Paillier encryptions of ℓ'-bit values n and d [7]. In their division protocol, the two participating parties secret share the decryption key, for example by making use of Shamir's secret sharing scheme [43]. The two parties can thus only decrypt with mutual consent. The inputs and intermediary values are held in encrypted form by both parties. The solution consists of two parts:

1. Compute encrypted approximation $[\tilde{a}]$ of $[a] = [2^k \div d]$.
2. Compute $[n \div d] = [(\tilde{a} \cdot n) \div 2^k]$.

In part 1, Dahl, Ning and Toft make use of the Taylor series of $\frac{1}{x}$ around 0:

$$\frac{1}{x} = \sum_{i=0}^{\infty}(1-x)^i \approx \sum_{i=0}^{\omega}(1-x)^i. \tag{4.38}$$

Now take $\omega = \ell'$, $\ell_d := \lfloor log_2(d) + 1 \rfloor$, $x = \frac{d}{2^{l_d}}$ and $k = \ell'^2 + \ell'$. Note that ℓ_d and

ℓ' have to be private, since information about n or d leaks otherwise. Now

$$\frac{2^k}{d} = 2^{k-\ell_d} \cdot \frac{1}{d/2^{\ell_d}} \tag{4.39}$$

$$\approx 2^{k-\ell_d} \cdot \sum_{i=0}^{\omega}(1 - d/2^{\ell_d})^i \tag{4.40}$$

$$= 2^{k-\ell_d} \cdot \sum_{i=0}^{\omega}((2^{\ell_d} - d) \cdot 2^{-\ell_d})^i. \tag{4.41}$$

Since $k \geq \ell_d$ and $2^{\ell_d} > d$, it follows that the approximate division $\tilde{a} \approx \frac{2^k}{d}$ can be computed with addition and multiplication in $\mathbb{Z}_N$. After obtaining $\tilde{a}$, part 2 consists of computing $n \cdot \tilde{a}$ and truncating it by 2^k. The result $\tilde{q}$ is in the interval $[(n \div d) - 1, n \div d]$. These computational steps are summarized in Protocol 5. The protocol makes use of a multiplication protocol such as the one given in Protocol 2. Additional sub-protocols $\pi_{\mathsf{bl}}, \pi_{\mathsf{inv}}$, π_{poly} and Trunc are invoked. The details of these protocols are beyond the scope of this thesis, but the functions of the protocols are as follows:

- $\pi_{\mathsf{bl}} : [2^{\ell_d}] \leftarrow [d]$ where $\ell_d = \lfloor \log_2(d) + 1 \rfloor$,
- $\pi_{\mathsf{inv}} : [x^{-1}] \leftarrow [x]$,
- $\pi_{\mathsf{poly}} : [A(p)] \leftarrow [p]$ for a known polynomial $A(x) = \sum_{i=0}^{\omega} x^i$,
- $\mathsf{Trunc} : [\tilde{q}] \leftarrow [\hat{q} \div 2^k]$ such that $\tilde{q} \in [\hat{q} \div 2^k, (\hat{q} \div 2^k) + 1]$.

Protocol 5: $[n \div d] \leftarrow \mathsf{EncDiv}([n], [d])$ [7]

```
Input  : $[n], [d]$.
Output: $[n \div d] \in [(n \div d) - 1, n \div d]$.
1 $[2^{\ell_d}] \leftarrow \pi_{\mathsf{bl}}([d])$
2 $[2^{-\ell_d}] \leftarrow \pi_{\mathsf{inv}}([2^{\ell_d}])$
3 $[p] \leftarrow [2^{\ell_d} - d] = [2^{\ell_d}] \cdot [d]^{-1}$
4 $[p] \leftarrow \mathsf{Mult}([p], [2^{-\ell_d}])$
5 $[\tilde{a}] \leftarrow 2^k \cdot [2^{-\ell_d}] \cdot \pi_{\mathsf{poly}}([\mathsf{p}]) = [2^{-\ell_d}]^{2^k} \cdot \pi_{\mathsf{poly}}([\mathsf{p}])$
6 $[\hat{q}] \leftarrow \mathsf{Mult}([n], [\tilde{a}])$
7 $[\tilde{q}] \leftarrow \mathsf{Trunc}([\hat{q}], k)$
```

This protocol requires $\mathcal{O}(\ell')$ encryptions to be exchanged. The invocation of the sub-protocols make Protocol 5 relatively complex. Details on the complexity can be found in [7]. In Protocol 4, the public divisor ensures that the division can be performed with the additive homomorphic property of Paillier encryption and one intermediate encryption. Therefore, Protocol 5 is computationally complex in comparison with Protocol 4. The decision on whether or not to publish the divisor is a trade-off between privacy and efficiency.

4.5.5 The square root

The square root operation is particularly complex in the encrypted domain. The square root of a number is found by an iterative method. The method of Newton-Raphson is often applied to approximate the square root, but this method may oscillate when performed over integer arithmetic [33]. Privacy preserving square root protocols have been designed that are based on the Newton-Raphson method, Goldschmidt's algorithm or on bit-by-bit computation [29, 33]. These protocols either need many iterations with a comparison protocol or they use multiple multiplication and division operations in every iteration in order to compute one square root. Moreover, if the number of iterations is public, information about the computation is leaked. The numbers of iterations can be hidden, but this will make the square root protocol even more complex. Therefore, we will strive to avoid the computation of an integer square root in the design of a privacy preserving spectral clustering algorithm.

To give the reader an intuition for the complexity of a square root protocol, we will summarize the approach that was designed by Manuel Liedel [29]. It uses a combination of the Newton-Raphson method and Goldschmidt's algorithm, which will be discussed first.

The Newton-Raphson method

The Newton-Raphson method can be used to approximate the zero of a differentiable function f. Starting with an initial approximation, the subsequent approximation is given by

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}. \tag{4.42}$$

When Equation 4.42 is applied to the function $f(R) = \frac{1}{R^2} - x$, the zero $R = \frac{1}{\sqrt{x}}$ can be approximated with quadratic convergence. By multiplying the result with x, an approximation of $\sqrt{x}$ is obtained. Substituting $f(R)$, the following recursion has to be implemented:

$$R_{j+1} = \frac{1}{2} R_j (3 - x \cdot R_j^2). \tag{4.43}$$

Goldschmidt's algorithm

Goldschmidt's algorithm is a variation of the Newton-Raphson method that allows one to approximate $\sqrt{x}$ and $\frac{1}{\sqrt{x}}$. An inital estimate $y_0 = \frac{1}{\sqrt{x_0}}$ is assumed such that $\frac{1}{2} < x_0 y_0^2 < \frac{3}{2}$. Set $g_0 = x_0 y_0$ and $h_0 = \frac{y_0}{2}$. Now g_i and h_i will converge to $\sqrt{x}$ and $\frac{1}{2\sqrt{x}}$ respectively. Goldschmidt's algorithm is given in Algorithm 7.

Algorithm 7: Goldschmidt's algorithm to approximate a square root

```
Input  : Initial approximation $y_0 = \frac{1}{\sqrt{x_0}}$ such that $\frac{1}{2} < x_0 y_0^2 < \frac{3}{2}$.
Output: Approximations $g_i$ of $\sqrt{x}$ and $h_i$ of $\frac{1}{2\sqrt{x}}$.
1 Set $g_0 = x_0 y_0$ and $h_0 = \frac{y_0}{2}$.
2 while $|g_i - g_{i-1}| > \epsilon$ do
3     $r_{i-1} = \frac{1}{2} - g_{i-1} h_{i-1}$
4     $g_i = g_{i-1}(1 + r_{i-1})$
5     $h_i = h_{i-1}(1 + r_{i-1})$
6 end
```

It is beyond the scope of this thesis to investigate the secure square root protocol in detail. The square root protocol that ensures an approximation of k bits accuracy is given in Protocol 6 in Appendix A. A fixed point representation of numbers is assumed with f bits for the fractional part. In the secure square root protocol in Appendix A, sub-protocols for the secure multiplication operation and the division operations Trunc and PubDiv are invoked multiple times in every iteration. This makes the computational and communication complexity of a square root operation in the encrypted domain large in comparison to other secure computations.

This section has discussed some of the necessary building blocks for a privacy preserving spectral clustering algorithm. These protocols are not sufficient. For example, a secure comparison protocol may be required to implement a stopping criterion for the QR algorithm. It is beyond the scope of this research to give an exhaustive design of a privacy preserving spectral clustering algorithm. However, in the Discussion section we will expound more on a privacy preserving design. Let us first consider the translation of the numerical algorithms from Chapter 3 to the message space of the Paillier cryptosystem, i.e. the integer domain.

Chapter 5

An investigation of secure spectral clustering

The focus of this thesis is the approximation of the eigenvectors of the Laplacian in the integer domain. However, we recall from Algorithm 2 that the spectral clustering algorithm consists of several steps. First, the Laplacian matrix is constructed. The eigenvectors of the Laplacian are computed and form the columns of a matrix U. The final step is to perform k-means clustering on the rows of U. This chapter will mainly be about the computation of eigenvectors in the integer domain. The numerical algorithms from Chapter 3 will be adapted to the integer domain. We will first, in Section 5.1, briefly discuss previous research that can be applied to the other steps of the spectral clustering algorithm. Next, the Lanczos algorithm will be adapted to $\mathbb{Z}_N$ in Section 5.2. A secure design of the Lanczos algorithm will also be proposed. The QR algorithm in $\mathbb{Z}_N$ is the subject of Section 5.3. The adaptation of the back substitution step is discussed in Section 5.4. The accuracy of the eigenvectors in the integer domain will be assessed in Chapter 6.

5.1 Constructing the Laplacian and k-means clustering

After this section, the research will be focused on the eigendecomposition of the Laplacian. This section shows how previous research on the other steps of the spectral clustering algorithm can be connected to the current research. Figure 5.1 gives an overview of the phases of the spectral clustering algorithm. The first and third blocks on the algorithm will now be discussed.

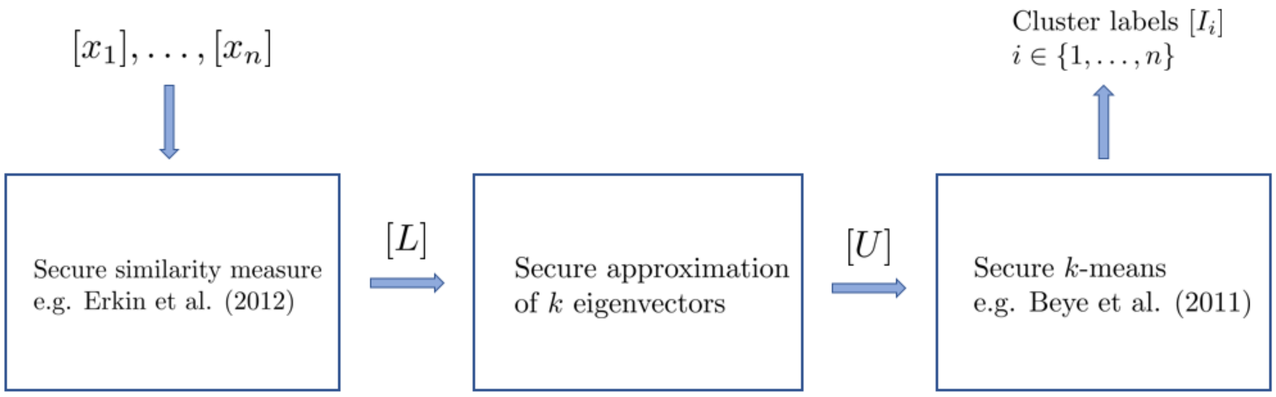

$[x_1], \ldots, [x_n]$
Secure similarity measure e.g. Erkin et al. (2012)
$[L]$
Secure approximation of k eigenvectors
$[U]$
Secure k-means e.g. Beye et al. (2011)
Cluster labels $[I_i]$ $i \in \{1, \ldots, n\}$

Figure 5.1: The spectral clustering algorithm consists of several steps.

5.1.1 The Laplacian

In Chapter 2 it was discussed that the Laplacian can be constructed in many different ways. The optimal choice depends on the context. However, a similarity measure $s(x_i, x_j)$ for every pair of data points x_i and x_j and a way to define the weights w_{ij} for matrix W are always required. Previous work by Erkin et al. can be applied to compute a similarity measure securely [10].

In [10], Erkin et al. use the cosine similarity between users A and B with data vectors x_A and x_B of length n. The cosine similarity is given by

$$\begin{aligned} sim(A,B) &= \frac{\sum_{j=1}^{n} x_{A,j} \cdot x_{B,j}}{\sqrt{\sum_{j=1}^{n} x_{A,j}^2 \cdot \sum_{j=1}^{n} x_{B,j}^2}} && (5.1)\\ &= \sum_{j=1}^{n} \frac{x_{A,j}}{\sqrt{\sum_{j=1}^{n} x_{A,j}^2}} \cdot \frac{x_{B,j}}{\sqrt{\sum_{j=1}^{n} x_{B,j}^2}} && (5.2)\\ &:= \sum_{j=1}^{n} \tilde{x}_{A,j} \cdot \tilde{x}_{B,j}. && (5.3) \end{aligned}$$

Every user I can compute $\tilde{x}_{I,j}$ individually and send the encrypted $[\tilde{x}_{I,j}]$ to the service provider. Once encrypted, a secure multiplication protocol is necessary to compute $[\tilde{x}_{A,j} \cdot \tilde{x}_{B,j}]$. The sum of the encrypted products can be computed with the additive homomorphic property of the Paillier cryptosystem.

The Lanczos algorithm efficiently exploits the sparsity of the input matrix. Therefore, we strive to use the similarity measure such that we obtain a sparse Laplacian L. For example, the cosine similarity may be compared to a threshold and the weight w_{AB} can be assigned the value $sim(A, B)$ if the similarity at least meets the threshold. The weight w_{AB} is 0 otherwise. This corresponds to a similarity graph in which vertices are connected, if and only if, the threshold is met. Once the encrypted weights w_{ij} on the edges between user pairs are computed, the entries of Laplacian L can be computed with linear operations:

$$[l_{ij}] = [w_{ij} - d_{ij}] = [w_{ij}] \cdot [d_{ij}]^{-1}. \tag{5.4}$$

5.1.2 Secure k-means clustering

Suppose that the eigenvectors of the Laplacian have been approximated securely and that these vectors form the columns of an encrypted matrix $[U]$. The rows of $[U]$ have to be clustered using the k-means clustering algorithm. Extensive research has been done on privacy preserving k-means clustering. This section succinctly discusses secure k-means algorithms on Paillier encrypted values with the aid of a third party. Note that many other privacy preserving k-means algorithms were designed and further research is necessary to investigate the optimal algorithm.

Erkin et al. provide a secure k-means algorithm in which multiple users and a server interact [9]. The proposed method requires the participation of all users and the computation and communication complexity is high. An improvement in terms of efficiency came from Erkin et al. in 2013 [11]. The users are divided into M groups and each group contains one helper user who owns a decryption key. The helper users and a server cooperate to obtain the clustering result. While these algorithms may be adapted to cluster the eigenvector matrix $[U]$, the users would have to participate in the computations. A secure algorithm in which the users are considered offline after a precomputation phase was designed in [1]. Beye et al. use a three-party setting in which encrypted user data is stored in a central entity, called the User Representative. The User Representative interacts with a Key Holder and an Efficiency Provider. The User Representative and the Efficiency Provider use permutations to prevent each other from learning which data belongs to which user. Blinding is used to prevent the Key Holder from learning data values after decryption. In this algorithm, the cluster centroids are encrypted. In order to determine the distances of the users to the centroids, division by an encrypted denominator is required. Moreover, a protocol to find the minimum distance of each user to the centroids is necessary. The algorithm stops after t rounds. The increased workload due to computation on encrypted centroids make the final complexity of this algorithm to the work of Erkin in [9] similar.

We will now turn to the adaptation of the numerical methods from Chapter 3 to the integer domain. From now on, it is assumed that the Laplacian $[L]$ is constructed in a privacy preserving manner. In this setting, a service provider and a privacy service provider would collaborate to approximate the encrypted eigenvectors of $[L]$. The situation is illustrated in Figure 5.2.

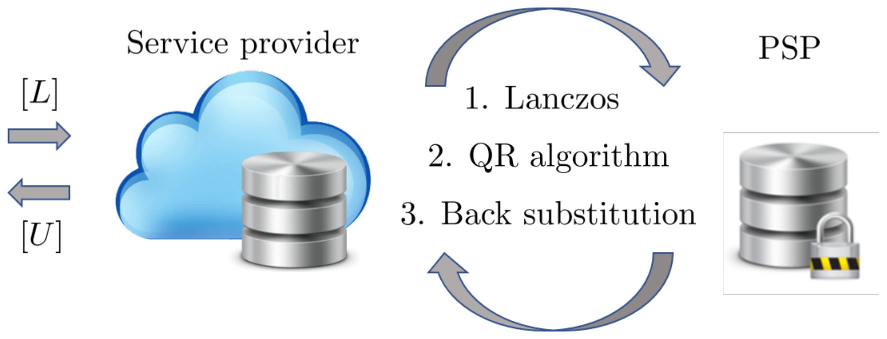


Figure 5.2: The two-party computation of the eigenvectors of a matrix L whose entries are encrypted with the Paillier cryptosystem. The PSP owns the decryption key.

5.2 The Lanczos algorithm

Recall that, given a symmetric matrix $L \in \mathbb{R}^{n\times n}$ the Lanczos algorithm computes an orthonormal basis $V \in \mathbb{R}^{n\times k}$ and a tridiagonal matrix $T \in \mathbb{R}^{k\times k}$ such that $LV = VT$ and the Ritz values of T are an approximation of the eigenvalues of L. The standard Lanczos algorithm incorporates normalization of the Lanczos vectors. However, in Section 4.5 it was discussed that the square root operation is expensive in a finite field. Therefore, it is proposed to perform an unnormalized version of the Lanczos algorithm in the integer domain. This algorithm will be tested in Chapter 6. At the end of this section, a secure design of the Lanczos algorithm will be discussed.

5.2.1 The unnormalized Lanczos algorithm

Christopher Paige became famous with his research on the performance of variants of the Lanczos algorithm [36, 35]. He described the unnormalized version of Lanczos, stemming from a different approach to the method. We no longer assume that $n\times n$ matrix T is symmetric. The unsymmetric equivalent of matrix T in (3.17) is

$$T = \begin{pmatrix} \alpha_1 & \gamma_2 & & & 0 \\ \beta_2 & \alpha_2 & \gamma_3 & & \\ & \beta_3 & \alpha_3 & \gamma_4 & \\ 0 & & \ddots & \ddots & \ddots \end{pmatrix}. \tag{5.5}$$

The unsymmetric equivalent of (3.18) is

$$Lv_j = \gamma_j v_{j-1} + \alpha_j v_j + \beta_j v_{j+1}. \tag{5.6}$$

Multiplication from the left by v_j^T or v_{j-1}^T respectively yields the following equations for α_j and γ_j:

$$\alpha_j = v_j^T L v_j / v_j^T v_j, \tag{5.7}$$
$$\gamma_j = v_{j-1}^T L v_j / v_{j-1}^T v_{j-1}. \tag{5.8}$$

These equations hold because $v_i^T v_j = 0$ for $i \neq j$. Now the γ_j can also be expressed in terms of β_j with the following relations:

$$\begin{aligned} \gamma_j v_{j-1}^T v_{j-1} &= v_{j-1}^T L v_j && \text{from (5.8)} \\ &= v_j^T L v_{j-1} && \text{since } L^T = L \\ &= v_j^T V t_{j-1} && \text{using } LV = VT \\ &= v_j^T (\gamma_{j-1} v_{j-2} + \alpha_{j-1} v_{j-1} + \beta_j v_j) \\ &= \beta_j v_j^T v_j. \end{aligned} \tag{5.9}$$

The coefficients and equation for forming the next vector v_{j+1} thus become

$$\gamma_j = \beta_j v_j^T v_j / v_{j-1}^T v_{j-1}, \tag{5.10}$$
$$\alpha_j = v_j^T L v_j / v_j^T v_j, \tag{5.11}$$
$$\beta_{j+1} v_{j+1} = L v_j - \alpha_j v_j - \gamma_j v_{j-1}. \tag{5.12}$$

Paige subsequently assumes that $\beta_{j+1} = 1$ and states that "this lack of normalization is unimportant [...] except that it may lead to exponent overflow in some computations"[35].

We thus obtain

$$T = \begin{pmatrix} \alpha_1 & \gamma_2 & & & 0 \\ 1 & \alpha_2 & \gamma_3 & & \\ & 1 & \alpha_3 & \gamma_4 & \\ 0 & & \ddots & \ddots & \ddots \end{pmatrix} \tag{5.13}$$

and an orthogonal basis V. Note that $\beta_j = 1$ implies that the columns of V are no longer orthonormal, but only orthogonal and not unitary. More concretely, this implies

$$V^T V = D_V, \tag{5.14}$$

for a diagonal matrix D_V. The entries of D_V are the squared Euclidean norms of V's columns, since these columns are not normalized:

$$D_V(ii) = \sum_{j=1}^{n} v_{ij}^2, \quad i \in \{1, \dots, n\}. \tag{5.15}$$

Due to this lack of normalization, the entries of v_j tend to grow as the algorithm progresses. One should pay attention to the possibility of overflow. This will be further investigated when the algorithm is tested in Chapter 6.

An overview of the unnormalized Lanczos algorithm is given in Algorithm 8.

Algorithm 8: Unnormalized Lanczos algorithm

```
1  $v_0 \leftarrow \underline{0}$
2  Generate a random vector $v_1 \in \mathbb{R}^n$.
3  $\beta_1 \leftarrow 0$
4  for $j = 1, 2, \ldots, m-1$ do
5      $\alpha_j \leftarrow (v_j \cdot Lv_j)/(v_j \cdot v_j)$
6      $\gamma_j \leftarrow \beta_j(v_j \cdot v_j)/(v_{j-1} \cdot v_{j-1})$
7      $\beta_{j+1} \leftarrow 1$
8      $v_{j+1} \leftarrow Lv_j - \alpha_j v_j - \gamma_j v_{j-1}$
9  end
10 $\alpha_m \leftarrow (Lv_m \cdot v_m)/(v_m \cdot v_m)$
```

We will now show the equivalence of the unnormalized and the normalized Lanczos algorithms. Define diagonal matrix $D_\gamma = \text{ diag}\left(1, \sqrt{\gamma_2}, \sqrt{\gamma_2\gamma_3}, \ldots, \sqrt{\gamma_2 \cdots \gamma_m}\right)$. Let C denote the symmetric Ritz matrix in (3.17) that is obtained with the normalized Lanczos algorithm. Matrix T is similar to C with the following transformation:

$$D\gamma T D\gamma^{-1} = C = \begin{pmatrix} \alpha_1 & \sqrt{\gamma_2} & & & 0 \\ \sqrt{\gamma_2} & \alpha_2 & \sqrt{\gamma_3} & & \\ & \sqrt{\gamma_3} & \alpha_3 & \sqrt{\gamma_4} & \\ 0 & & \ddots & \ddots & \ddots \end{pmatrix}. \tag{5.16}$$

Algorithm 8 can directly be translated to the integer domain using the integer operations that were defined in Section 4.3.

5.2.2 The Lanczos algorithm in $\mathbb{Z}_N$

The unnormalized Lanczos algorithm can be translated to $\mathbb{Z}_N$ in a straightforward manner. The Lanczos algorithm without square roots was also implemented in a finite field in [24, 26]. However, the authors did not investigate the influence of this adaptation on the convergence and accuracy of the algorithm. The Lanczos algorithm in $\mathbb{Z}_N$ is given in Algorithm 9. The addition, subtraction, multiplication and division operation in $\mathbb{Z}_N$ were defined in Section 4.3. In Algorithm 9, fixed point arithmetic is applied in the decimal number system because the algorithm was implemented in this format. The scaling parameter d can be adjusted. In a secure version of the algorithm, the binary number system will be used again. After the unnormalized Lanczos method is performed, α_j and γ_j are obtained. These values are scaled by 10^d. The Laplacian L is unscaled. Note that the computations are performed modulo N.

It was investigated whether the number of divisions could be reduced by only performing divisions every few iterations. However, the danger of overflow occurs quickly due to the number of multiplications and the growth of the entries of the vectors v_j. No systematical way of reducing the number of divisions was found.

Algorithm 9: Unnormalized fixed point Lanczos algorithm in $\mathbb{Z}_N$

```
1  $v_0 \leftarrow \underline{0}$
2  Generate a random vector $v_1 \in \mathbb{Z}_N^n$.
3  $v_1 \leftarrow 10^d \cdot v_1$
4  $\beta_1 \leftarrow 0$
5  for $j = 1, 2, \ldots, m-1$ do
6      $\alpha_j \leftarrow (10^d v_j \cdot L v_j) \div (v_j \cdot v_j)$
7      $\gamma_j \leftarrow (\beta_j v_j \cdot v_j) \div (v_{j-1} \cdot v_{j-1})$
8      $\beta_{j+1} \leftarrow 10^d$
9      $v_{j+1} \leftarrow L v_j - (\alpha_j v_j) \div 10^d - (\gamma_j v_{j-1}) \div 10^d$
10 end
11 $\alpha_m \leftarrow (10^d v_m \cdot L v_m) \div (v_m \cdot v_m)$
```

In Chapter 6, the performance of Algorithm 9 will be tested on three real datasets. It will be compared to the performance of Algorithm 8, in which the computations take place in $\mathbb{R}$.

5.2.3 The secure Lanczos algorithm

The privacy preserving protocols of Section 4.5 will now be used in a secure design of the Lanczos algorithm. The design gives the reader an idea of the shape that such an algorithm could take. Certainly, further research is required for an optimal design.

Suppose that matrix $[L]$ contains Paillier encrypted ciphertexts. The initial vectors v_0 and v_1 are known or random and can be in plaintext. Therefore, the first matrix vector product $[w] = [L \cdot v_1]$ can be computed with the additive homomorphic property of Paillier encryption. Let l_k denote the kth row of L. The entry $[w_k]$ is computed as follows:

$$[w_k] = [l_k \cdot v_1] = [\sum_{i=1}^{n} l_{ki} v_{1i}] = \prod_{i=1}^{n} [l_{ki}]^{v_{1i}}. \tag{5.17}$$

The subsequent v_j's and the α_j and γ_j are preferably not leaked as they can be used to obtain the eigenvalues of L. All the $[\alpha_j]$, $[\gamma_j]$ and $[v_j]$ have to be stored in order to compute the eigenvectors of $[L]$. The sub-diagonal entries β_j do not have to be computed since they are known to be equal to 2^f.

In Algorithm 10, vectors will be in bold notation in order to distinguish between scalars and vectors. Vectors are encrypted entry-wise. When computations on vectors such as addition, truncation or the multiplication by a scalar are performed, the computations are performed entry-wise. Encrypted numbers without a subscript, such as $[x]$ and $[\mathbf{w}]$, denote intermediate results that can be overwritten. Note that an alternative computation is performed for vector $[\mathbf{v_3}]$ because $\mathbf{v_1}$ is a public vector. Furthermore, scalars m, n and f are public values.

Data packing

We will now give the reader an idea of how data packing may be applied to improve the efficiency of a secure Lanczos algorithm. In Algorithm 10, the encrypted Laplacian $[L]$ is used in a secure matrix vector product $\mathsf{MatMult}([L], [v])$. A secure matrix vector product is relatively complex. A matrix vector product is a series of inner products of the rows of the matrix with the same vector v_j. Therefore, data packing may be used to reduce the number of operations over encrypted entries of L. A similar construction was applied in [10]. Suppose that we pack column l_j of L in one encryption:

$$l_j^{\text{Packed}} = [l_{1j}| \ \dots \ |l_{nj}]. \tag{5.18}$$

Let $\otimes$ denote a secure multiplication operation. If another encryption contains the jth entry of v, the entire column of L can be multiplied with this entry with one secure multiplication protocol:

$$[l_{1j}| \ \dots \ | \ l_{nj}] \otimes [v_j] = [l_{1j}v_j| \ \dots \ | \ l_{nj}v_j]. \tag{5.19}$$

Now the packed matrix vector product can be computed as

$$\prod_{j=1}^{n}[l_{1j}v_j| \ \dots \ | \ l_{nj}v_j] = [\sum_{j=1}^{n} l_{1j}v_j| \ \dots \ | \ \sum_{j=1}^{n} l_{nj}v_j] := [w_1| \ \dots \ |w_n]. \tag{5.20}$$

Let us discuss the necessary compartment size for such a packed matrix-vector product. Suppose that the entries of L are represented by t_L bits and the entries of v_j are represented by t_v bits. The packed entries $\sum_{j=1}^{n} l_{ij}v_j$ will comprise $t_L + t_v + \lceil \log(n) \rceil$ bits, so this should be the minimum size of the compartments. Since the message space has ℓ_N bits, a total of $s = \ell_N/(t_L + t_v + \lceil \log(n) \rceil)$ encryptions will be necessary per row of L. We need sn secure multiplications

Algorithm 10: Secure Lanczos algorithm

```
Input  : $n \times n$ symmetric encrypted matrix $[L]$
Output: $[\alpha_j]$, $[\gamma_j]$, $[\mathbf{v_j}]$, $j \in \{1, \dots, m\}$.
1  $\mathbf{v_0} \leftarrow \underline{0}$
2  $\gamma_1 \leftarrow 0$
3  Generate a random vector $v_1 \in \mathbb{Z}_N^n$ in plaintext.
4  $\mathbf{v_1} \leftarrow 2^f \cdot \mathbf{v_1}$.
5  $[\mathbf{w}] \leftarrow [L \cdot \mathbf{v_1}]$ using additive homomorphic properties
6  $[x] \leftarrow [\mathbf{v_1} \cdot \mathbf{w}]$ using additive homomorphic properties
7  $[\alpha_1] \leftarrow [(2^f x) \div (\mathbf{v_1} \cdot \mathbf{v_1})] = \mathsf{PubDiv}([x]^{2^f}, \mathbf{v_1} \cdot \mathbf{v_1})$
8  $[\mathbf{u}] \leftarrow \mathsf{Trunc}([\alpha_1]^{\mathbf{v_1}}, 2^f)$
9  $[\mathbf{v_2}] \leftarrow [\mathbf{w} - \mathbf{u}] = [\mathbf{w}] \cdot [\mathbf{u}]^{-1}$
10 for $j = 2, \dots, m-1$ do
11     $[\mathbf{w}] \leftarrow [L\mathbf{v_j}] = \mathsf{MatMult}([L], [\mathbf{v_j}])$
12     $[x] \leftarrow [\mathbf{v_j} \cdot \mathbf{w}] = \mathsf{Inner}([\mathbf{v_j}], [\mathbf{w}])$
13     $[y_j] \leftarrow [\mathbf{v_j} \cdot \mathbf{v_j}] = \mathsf{Inner}([\mathbf{v_j}], [\mathbf{v_j}])$
14     $[y_j] \leftarrow \mathsf{Trunc}([y_j], 2^f)$                store for next iteration
15     $[\alpha_j] \leftarrow [(2^f x) \div y_j] = \mathsf{PrivDiv}([x]^{2^f}, [y_j])$
16     $[\gamma_j] \leftarrow [(2^f y_j) \div y_{j-1}] = \mathsf{PrivDiv}([y_j]^{2^f}, [y_{j-1}])$
17     $[\mathbf{u}] \leftarrow [\alpha_j \mathbf{v_j}] = \mathsf{Mult}([\alpha_j], [\mathbf{v_j}])$
18     $[\mathbf{u}] \leftarrow \mathsf{Trunc}([\mathbf{u}], 2^f)$
19     $[\mathbf{z}] \leftarrow [\gamma_j \mathbf{v_{j-1}}] = \mathsf{Mult}([\gamma_j], [\mathbf{v_{j-1}}])$
20     $[\mathbf{z}] \leftarrow \mathsf{Trunc}([\mathbf{z}], 2^f)$
21     $[\mathbf{v_{j+1}}] \leftarrow [\mathbf{w} - \mathbf{u} - \mathbf{z}] = [\mathbf{w}] \cdot [\mathbf{u}]^{-1} \cdot [\mathbf{z}]^{-1}(*)$
22 end
23 $(*)[\mathbf{v_3}] \leftarrow [\mathbf{w} - \mathbf{u} - \gamma_2 \mathbf{v_1}] = [\mathbf{w}] \cdot [\mathbf{u}]^{-1} \cdot [\gamma_2]^{-\mathbf{v_1}}$
24 $[\mathbf{w}] \leftarrow [L\mathbf{v_m}] = \mathsf{MatMult}([L], [\mathbf{v_m}])$
25 $[x] \leftarrow [\mathbf{v_j} \cdot \mathbf{w}] = \mathsf{Inner}([\mathbf{v_j}], [\mathbf{w}])$
26 $[y_m] \leftarrow [\mathbf{v_j} \cdot \mathbf{v_j}] = \mathsf{Inner}([\mathbf{v_j}], [\mathbf{v_j}])$
27 $\alpha_m \leftarrow [(2^f x) \div y_m] = \mathsf{PrivDiv}([x]^{2^f}, [y_j])$
```

and $s(n-1)$ secure additions for a secure matrix vector product. When $s < n$, this is an improvement. This packed operation can also improve other steps of the Lanczos algorithm. For example, it may be beneficial to perform the multiplication of the vectors $[\mathbf{v}]$ by encrypted constants $[\alpha]$ or $[\gamma]$ in a packed manner.

However, attention should be paid to the fact that the entries of the columns of V grow due to the iterative nature of the algorithm. Therefore, a different number of packed entries may be allowed in each iteration. More research is required to see whether data packing is beneficial to the secure Lanczos algoritm.

After the unsymmetric Ritz matrix T has been obtained, the QR algorithm is applied. The QR algorithm will now be adapted to $\mathbb{Z}_N$.

5.3 The QR algorithm

The input of the QR algorithm is the $m \times m$ unsymmetric tridiagonal matrix T. Let t_j denote the jth column of T. Let r_{ij} denote the entries of upper triangular matrix R and v_{GS} the modified Gram-Schmidt vector. The scaling factor e can be varied. First, the QR decomposition and QR algorithm in $\mathbb{Z}_N$ will be given, using the arithmetic operations as defined in Section 4.3. Secure designs of the algorithms that make use of the protocols in Section 4.5 follow.

5.3.1 The QR decomposition in $\mathbb{Z}_N$

Algorithm 11 shows the QR decomposition in integer arithmetic. While a QR decomposition in $\mathbb{R}$ generally makes use of the Euclidean 2-norm to compute the diagonal entries r_{jj} of R, now the 1-norm is used.[1] Since mapping ψ in (4.21) is used to represent negative numbers in $\mathbb{Z}_N$, the absolute value can be defined for $x \in [0, N-1]$ as the operation

$$x \mapsto \begin{cases} x, & \text{if } x \in [0, N/2-1], \\ N-x, & \text{if } x \in [N/2, N-1]. \end{cases} \tag{5.21}$$

By using the 1-norm, the square root operation is eliminated while overflow of the vectors q_j of Q is avoided. Note that, as a consequence, the columns of Q are orthogonal but not normalized.

Algorithm 11: The adapted QR decomposition in $\mathbb{Z}_N$

```
1  $q_1 \leftarrow \underline{0}$
2  for $j = 1, \dots, m$ do
3      $v_{GS} \leftarrow t_j$
4      for $i = 1, \dots, j-1$ do
5          $r_{ij} \leftarrow (10^e\ q_i \cdot v_{GS}) \div (q_i \cdot q_i)$
6          $v_{GS} \leftarrow v_{GS} - (r_{ij} q_i) \div 10^e$
7      end
8      $r_{jj} \leftarrow \sum_{l=1}^{m} |v_l|$
9      $q_j \leftarrow (10^e\ v_{GS}) \div r_{jj}$
10 end
```

[1]The inf-norm $\|v\|_\infty = \max(|v_1|, \dots, |v_m|)$ was also attempted, but this caused overflow to occur quickly in the columns of Q.

5.3.2 The QR algorithm in $\mathbb{Z}_N$

The QR algorithm in $\mathbb{Z}_N$ makes use of the QR decomposition in $\mathbb{Z}_N$ and can directly be translated to $\mathbb{Z}_N$ with integer arithmetic operations:

Algorithm 12: The adapted QR algorithm with shifts in $\mathbb{Z}_N$

```
1 $T_1 \leftarrow T$
2 $Q_{\prod} \leftarrow I \in \mathbb{Z}^{m \times m}$
3 for $k = 1, 2, \ldots, p$ do
4     $\mu \leftarrow T_{k(m,m)}$
5     $T_k \leftarrow T_k - \mu I$
6     Compute $T_k = Q_k R_k$ with the integer QR decomposition
7     $T_{k+1} \leftarrow (R_k Q_k) \div 10^e + \mu I$
8     $Q_{\prod} \leftarrow (Q_{\prod} Q_k) \div 10^e$
9 end
```

The performance of Algorithms 11 and 12 will be discussed in Chapter 6.

5.3.3 Secure design of the QR algorithm

A privacy preserving version of Gram-Schmidt orthogonalization for two parties was designed by Failla and Barni [12]. They make use of a secure multiplication, division and inner product protocol. The normalization step is left out. The effect of the adaptation of the Gram-Schmidt algorithm to the integer domain was not investigated. The authors remark that "a study of the error introduced by quantization will be really needed for practical implementations."

A possible design for a secure QR decomposition is given in Algorithm 13. Note that only the three non-zero diagonals of $[T]$ need to be encrypted. The privacy preserving protocols change accordingly. The computation of q_1 is different than the other iterations, because no iteration over i takes place. Vector q_1 is directly computed by dividing t_1 with the 1-norm. Moreover, some adaptations are proposed for the privacy preserving QR decomposition. First of all, since the normalization factor is only used to avoid overflow, we hypothesize that it is acceptable to substitute the 1-norm for the following normalization factor:

$$r_{jj} = \sum_{l=1}^{m} v_l. \tag{5.22}$$

This "semi-norm" discards the absolute value and hereby reduces the complexity of the algorithm. However, care should be taken that this normalization factor may become either very small or even 0, causing the algorithm to break down. The 1-norm does not have this disadvantage and could be used instead when the normalization factor in (5.22) equals 0. The reader is referred to [22] for a privacy preserving absolute value protocol.

Furthermore, division takes place multiple times in each iterations: to compute the $[r_{ij}]$ and to compute the q_j in the final step. The algorithm would be speeded up considerably if the divisors $q_i \cdot q_i$ and r_{jj} were public instead of encrypted. We suggest to publish the value $q_i \cdot q_i$ since the norms of Q_k's columns do not reveal too valuable information. By contrast, the diagonal entries of T_k should definitely be private, because these entries converge to the eigenvalues in the QR algorithm. Since Q is a diagonal matrix and the matrix product RQ is used to compute the next T_{k+1}, we propose to keep the $[r_{jj}]$ encrypted. The effects of publishing these entries for the privacy of T_k's diagonal can not be overseen.

In Algorithm 13, the variables without a subscript can be overwritten in every iteration. Vectors are indicated in bold.

Algorithm 13: Secure QR decomposition

```
Input  : $m \times m$ tridiagonal encrypted matrix $[T]$
Output: $m \times m$ orthogonal matrix $[Q]$
        $m \times m$ upper triangular matrix $[R]$
1  $q_1 \leftarrow \underline{0}$
2  for $j = 1, \dots, m$ do
3      $[\mathbf{v_{GS}}] = [\mathbf{t_j}]$
4      for $i = 1, \dots, j-1$ do
5          $[x] \leftarrow \mathsf{Inner}([\mathbf{q_i}], [\mathbf{v_{GS}}])$
6          $[y] \leftarrow \mathsf{Inner}([\mathbf{q_i}], [\mathbf{q_i}])$
7          $y \leftarrow \mathsf{Decrypt}([y])$
8          $[r_{ij}] \leftarrow \mathsf{Div}([x]^{2^f}, y)$
9          $[\mathbf{z}] \leftarrow \mathsf{Mult}([r_{ij}], [\mathbf{q_i}])$
10         $[\mathbf{z}] \leftarrow \mathsf{Trunc}([\mathbf{z}], 2^f)$
11         $[\mathbf{v_{GS}}] \leftarrow [\mathbf{v_{GS}} - \mathbf{z}] = [\mathbf{v_{GS}}] \cdot [\mathbf{z}]^{-1}$
12     end
13     $[r_{jj}] \leftarrow [\sum_{l=1}^{m} v_l] = [v_1] \cdots [v_m]$
14     $[\mathbf{q_j}] \leftarrow \mathsf{PrivDiv}([\mathbf{v_{GS}}]^{2^f}, [r_{jj}])$
15 end
```

When the QR decomposition is performed securely, a design for a secure implementation of the QR algorithm is quite straightforward. The secure QR algorithm is given in Algorithm 14. Again, the first iteration is different from the other iterations because the first $Q_{\prod}$ is the identity matrix and can be public.

Algorithm 14: Secure QR algorithm with shifts

```
Input  : $m \times m$ tridiagonal matrix $[T]$
Output: $m \times m$ upper triangular matrix $[T_p]$
        $m \times m$ orthogonal $[Q_{\prod}]$
1  $[T_1] \leftarrow [T]$
2  $Q_{\prod} \leftarrow I \in \mathbb{Z}^{m \times m}$
3  for $k = 1, 2, \ldots, p$ do
4      $[\mu] \leftarrow [T_{k(m,m)}]$
5      $[T_k] \leftarrow [T_k - \mu I] = [T_k] \cdot [\mu I]^{-1}$
6      Compute $[Q_k], [R_k]$ with the secure QR decomposition
7      $[T_{k+1}] \leftarrow \mathsf{MatMult}([R_k], [Q_k])$
8      $[T_{k+1}] \leftarrow \mathsf{Trunc}([T_{k+1}], 2^f)$
9      $[T_{k+1}] \leftarrow [T_{k+1} + \mu I] = [T_{k+1}] \cdot [\mu I]$
10     $[Q_{\prod}] \leftarrow \mathsf{MatMult}([Q_{\prod}], [Q_k])$
11     $[Q_{\prod}] \leftarrow \mathsf{Trunc}([Q_{\prod}], 2^f)$
12 end
```

5.4 Back substitution

We will describe the computation of one eigenvector w with the use of back substitution. For every eigenvector such a system needs to be solved. After solving k linear systems, the eigenvectors constitute the columns of matrix W, which can be used to compute the eigenvectors of L.

5.4.1 Back substitution in $\mathbb{Z}_N$

After p iterations of the shifted QR algorithm, an upper triangular matrix T_p is obtained with the Ritz values in ascending order on the diagonal. We assume that the matrix T_p is scaled by 10^g at the beginning of the back substitution step. Suppose that we select diagonal entry (l, l) of T_p as Ritz value θ_l. The linear system $(T_p - \theta_l I)w = \mathbf{0}$ is solved in Algorithm 15. Note that the system is solved from bottom to top so the iterates are descending. Due to the scaled entries of $\tilde{T}_p$, no scaling or truncation is required.

Algorithm 15: Back substitution in $\mathbb{Z}_N$

```
1  $\theta_l \leftarrow T_{p(l,l)}$
2  $\tilde{T}_p \leftarrow T_p - \theta_l I$
3  for $i = m, \ldots, l+1$ do
4      $w_i \leftarrow 0$
5  end
6  $w_l \leftarrow -\tilde{T}_{p(l-1,l-1)}$
7  $w_{l-1} \leftarrow \tilde{T}_{p(l-1,l)}$
8  for $i = l-2, \ldots, 1$ do
9      $w_i \leftarrow (-\sum_{j=i+1}^{m} \tilde{T}_{p(i,j)} w_j) \div \tilde{T}_{p(i,i)}$
10 end
```

5.4.2 Secure back substitution

The design for a secure back substitution algorithm is given in Algorithm 16. Note that T_P is an upper triangular matrix and thus only the non-zero entries have to be encrypted. The privacy preserving protocols change accordingly.

Algorithm 16: Secure back substitution

```
   Input  : $m \times m$ upper triangular matrix $[T_p]$
   Output: eigenvector $[w]$ of $[T_p]$
1  $[\theta_l] \leftarrow [T_{p(l,l)}]$
2  $[\tilde{T}_p] \leftarrow [T_p - \theta_l I] = [T_p] \cdot [\theta_l I]^{-1}$
3  for $i = m, \ldots, l+1$ do
4      $w_i \leftarrow 0$
5  end
6  $[w_l] \leftarrow [-\tilde{T}_{p(l-1,l-1)}] = [\tilde{T}_{p(l-1,l-1)}]^{-1}$
7  $[w_{l-1}] \leftarrow [\tilde{T}_{p(l-1,l)}]$
8  for $i = l-2, \ldots, 1$ do
9      for $j = i+1, \ldots, l$ do
10         $[y_j] \leftarrow [\tilde{T}_{p(i,j)} w_j] = \mathsf{Mult}([\tilde{T}_{p(i,j)}], [w_j])$
11     end
12     $[x] \leftarrow [\sum_{j=i+1}^{l} \tilde{T}_{p(i,j)} w_j] = \prod_{j=i+1}^{l} [y_j]$
13     $[w_i] \leftarrow [(-\sum_{j=i+1}^{l} \tilde{T}_{p(i,j)} w_j) \div \tilde{T}_{p(i,i)}] = \mathsf{PrivDiv}([x], [\tilde{T}_{p(i,i)}])$
14 end
```

5.5 Connecting the algorithms

We end this chapter with an overview of how the eigenvectors of the Laplacian can be computed in $\mathbb{Z}_N$. If the Lanczos basis V, the orthogonal $Q_{\prod}$ and the eigenvectors W of T_p are computed in $\mathbb{Z}_N$, the approximated eigenvectors of L are computed as the columns of a matrix $\tilde{U}$ with

$$\tilde{U} = VQ_{\prod}W. \tag{5.23}$$

If $[V], [Q_{\prod}]$ and $[W]$ have been obtained securely, a secure matrix product has to be performed twice to compute $[\tilde{U}]$ securely:

$$[Y] = \mathsf{MatMult}([Q_{\prod}], [W]), \tag{5.24}$$
$$[\tilde{U}] = \mathsf{MatMult}([V], [Y]). \tag{5.25}$$

In this chapter we saw the adaptation of the approximation of eigenvectors to the integer domain. Theoretically, the eigenvectors should be computable in the message space of the Paillier cryptosystem. A design for the algorithms in the ciphertext space was also given. However, the influence of the adaptation of the algorithms to $\mathbb{Z}_N$ should be investigated. This is the topic of Chapter 6.

Chapter 6

Results

It has been assessed whether the adaptation of the algorithms to the integer domain has influence on the accuracy of the algorithms and on the cluster quality. This chapter will first explain in Section 6.1 how the algorithms were assessed. The results for the Lanczos algorithm, QR algorithm and back substitution are shown in Sections 6.2, 6.3 and 6.4 respectively.

6.1 Methodology

Three stages of the eigenvector approximation are distinguished: the Lanczos algorithm, the QR algorithm and back substitution. In order to investigate the influence of adapting the algorithms to the integer domain, the performance of the algorithms in $\mathbb{R}$ and $\mathbb{Z}_N$ are compared. The value of N is chosen to comprise 2048 bits. Therefore, we say that overflow occurs when a number becomes larger than 1024 bits, since half of the domain is used to represent negative numbers. The algorithms were implemented in Python 3.6 and tested on three real datasets. This section describes the assessment criteria, the datasets and their Laplacians.

6.1.1 Assessment criteria

Both the accuracy of the eigenvector approximation and the clustering accuracy are a topic of research. The accuracy of the Ritz value θ_i to eigenvalue λ_i of L is assessed with the relative error:

$$\frac{|\theta_i - \lambda_i|}{|\lambda_i|}. \tag{6.1}$$

The accuracy of the corresponding Ritz vector $\tilde{u}_i$ to an eigenvector u_i of L is measured with the absolute cosine of the angle α between the vectors:

$$|\cos(\alpha)| = \frac{|\tilde{u}_i \cdot u_i|}{\|\tilde{u}_i\| \cdot \|u_i\|}. \tag{6.2}$$

To investigate the applicability of the algorithms to the encrypted domain, the number of iterations, the bit lengths of the entries and the required scaling factor in fixed point arithmetic will be mentioned for every algorithm. Furthermore, two quality criteria are for the clustering results: the clustering accuracy and the average silhouette value of the data points.

The algorithm is tested on datasets for which the cluster indices of the data points are known. The spectral clustering algorithm also assigns a cluster index to every data point. The cluster indices are compared. The *clustering accuracy* is the percentage of correctly assigned indices:

$$\frac{\#\text{ correctly clustered data points}}{\text{total }\#\text{ data points}} \cdot 100\%. \tag{6.3}$$

The *silhouette value* is a measure of the compactness and separation of clusters [41]. The distance of the data point to other data points in the same cluster is compared to the distance to data points in other clusters. Formally, the silhouette value of data point i is defined as

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \tag{6.4}$$

where $a(i)$ is the average distance from point i to other points in the same cluster, and $b(i)$ is the minimum average distance from point i to points in a different cluster. The squared Euclidean distance is used in the computation of the silhouette value. From the above definition it follows that

$$-1 \leq s(i) \leq 1 \tag{6.5}$$

for each data point i. A positive silhouette value indicates that the data point is clustered well. From a negative silhouette value we may conclude that a data point has been misclassified.

6.1.2 Datasets

Three datasets from the UCI Machine Learning Repository were used to assess the spectral clustering algorithm in $\mathbb{Z}_N$: the Wisconsin Breast Cancer Dataset, the Yeast5 dataset and the Yeast10 dataset [28]. This section gives a description of the datasets and the corresponding Laplacian matrices. A part of the spectrum of the Laplacians will be given as the distance between eigenvalues may have an influence on the performance of the algorithms.

The Wisconsin Breast Cancer Dataset dataset contains samples of breast mass of 699 individuals whose breast mass contains either benign or malignant cells [28].

The size of the dataset is 699×9. The normalized Laplacian of this dataset was constructed with a full Gaussian similarity graph, using $\sigma = 22$. The entries of the Laplacian should be integers. Therefore, the entries were scaled by a factor 10^5 and rounded down. An accuracy of 96% can be achieved with this Laplacian. A part of the spectrum is shown in Figure 6.1.

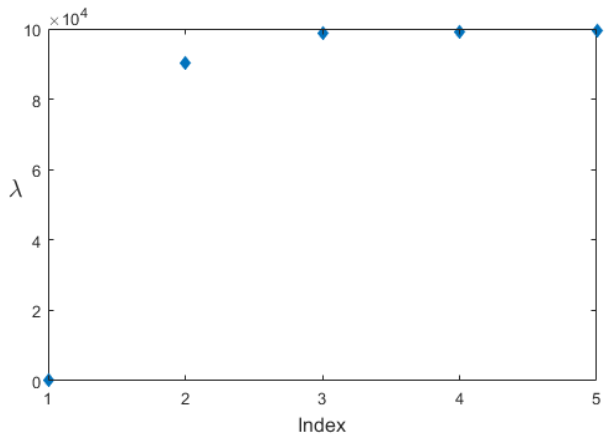


Figure 6.1: The first five eigenvalues of the Laplacian of the Wisconsin Breast Cancer dataset.

An illustration of the classification of the first two eigenvectors of the Laplacian is shown in Figure 6.2. A rough division of this dataset into two convex sets is possible, but some of the outlying malignant data points will be difficult to cluster.

The second real dataset is the Yeast5 dataset, a DNA microarray dataset that contains gene expression levels of yeast cell cycles. The fluctuation of 384 genes over 17 time instances is measured, so the dataset has size 384×17. The data points should be clustered into five phases of the cell cycle. The log-transformed data set was downloaded from [56]. In previous spectral clustering research, an accuracy of 65% was achieved [47]. These results were reproduced by using the correlation coefficient matrix of the gene expression levels as adjacency matrix A. The correlation coefficients were scaled to the interval $[0, 2]$ to mimic the similarity that was used in [47]. A normalized Laplacian was constructed and scaled by 10^5. A part of the spectrum is shown in Figure 6.3.

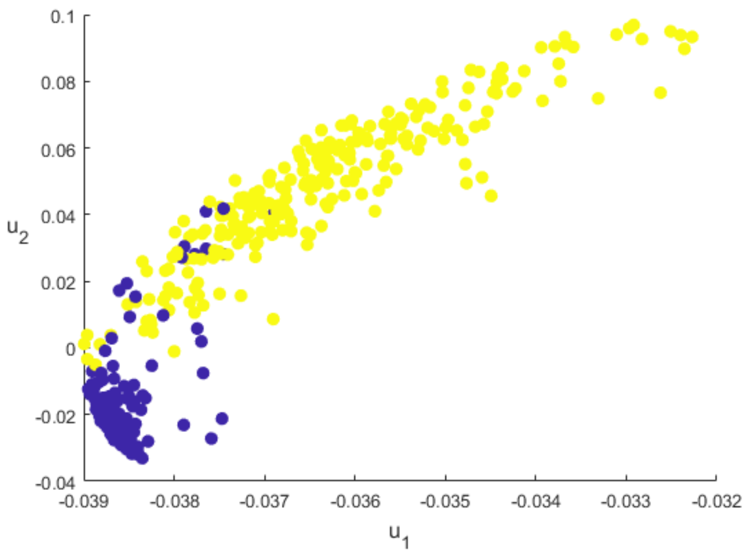


Figure 6.2: Clusters of the first two eigenvectors of the Laplacian for the Wisconsin Breast Cancer dataset. Yellow data points are benign cells, purple data points are malignant cells.

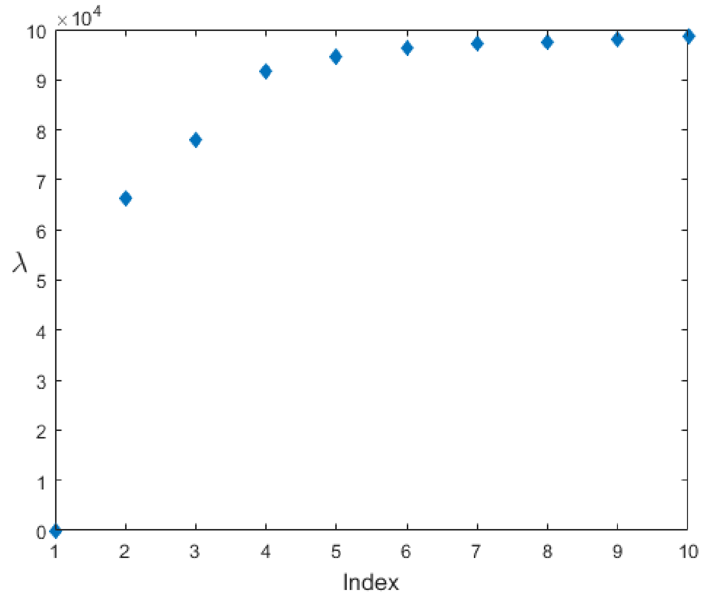


Figure 6.3: The first ten eigenvalues of the Laplacian of the Yeast5 dataset.

Three eigenvectors and their cluster division are shown in Figure 6.4. The clusters are not perfectly convex and the clustering task are therefore expected to be challenging.

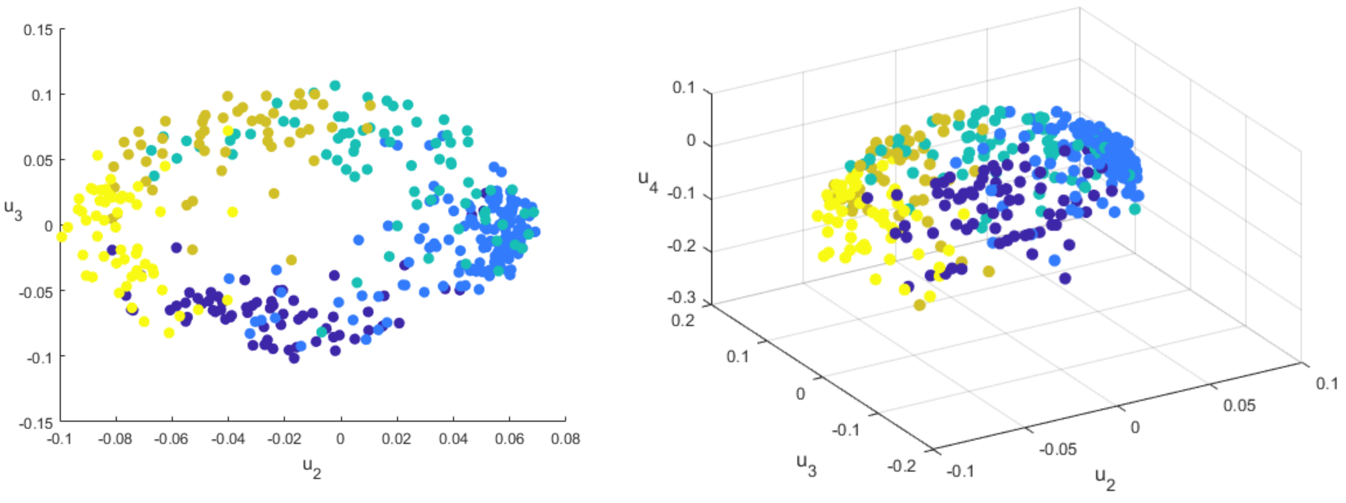


Figure 6.4: Two angles of the second to fourth eigenvectors of the Laplacian of the Yeast dataset divided into five clusters.

The Yeast10 dataset is a 1484 $\times$ 8 dataset in which ten classes of yeast proteins can be distinguished [21]. This dataset is notoriously difficult and an accuracy of 40% was considered acceptable in previous research [44, 54]. An unnormalized Laplacian with a 16-nearest neighbors similarity function and $\sigma = 3.4$ was used. This Laplacian yields an approximate accuracy of 38%. The Laplacian was scaled by a factor 10^1. A plot of the eigenvectors is omitted since the eigenvectors can not be visualized well in three dimensions. A part of the spectrum is given in Figure 6.5.

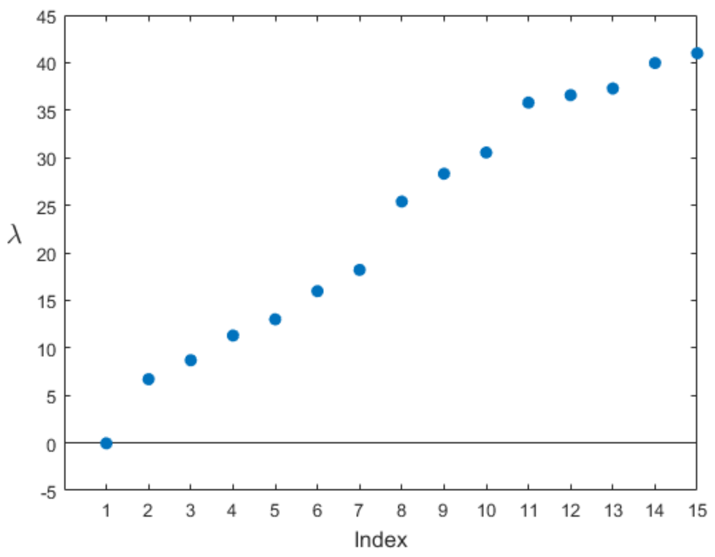


Figure 6.5: The first 15 eigenvalues of the Laplacian of the Yeast10 dataset.

6.2 The Lanczos algorithm

The performance of Algorithm 8 and Algorithm 9 are compared. After the Lanczos algorithm, the Ritz values and the eigenvectors of T are computed with the **scipy.linalg.eig** function in Python. After the algorithm in $\mathbb{Z}_N$, the Ritz values are rescaled by 10^d. The eigenvectors of T are stored in a matrix W. The Ritz vectors are computed with $\tilde{U} = VW$.

6.2.1 Wisconsin Breast Cancer Dataset

A scaling parameter $d = 6$ was required to obtain sufficient accuracy. Figure 6.6 shows the Ritz values in the first 16 iterations of the Lanczos algorithm in $\mathbb{Z}_N$. Note that a spurious eigenvalue occurs at iteration 7. The Lanczos algorithm converges quickly to the smallest eigenvalues because the large eigenvalues of L are bundled at one end of the spectrum.

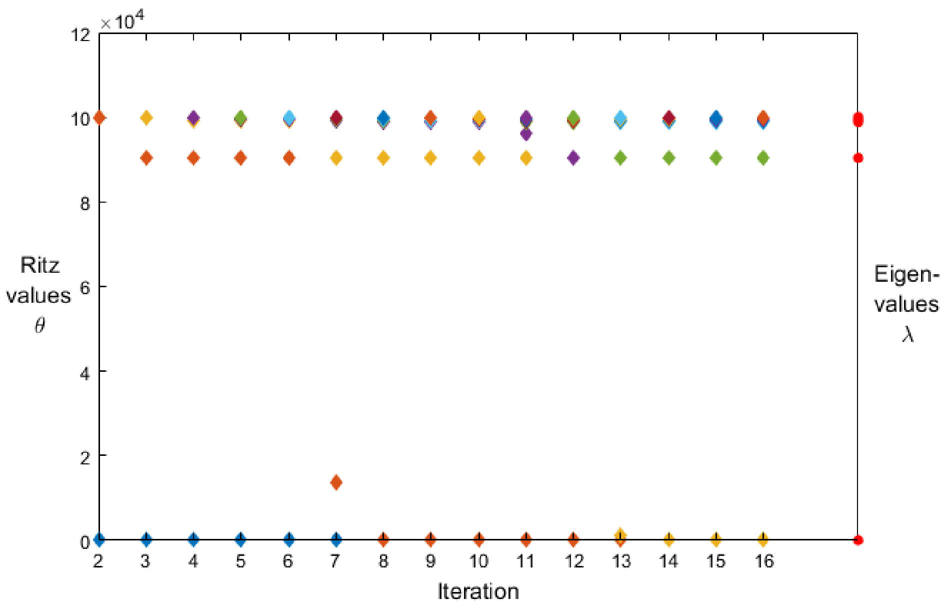


Figure 6.6: The Ritz values in the first 16 iterations of the Lanczos algorithm in $\mathbb{Z}_N$ for the Breast Cancer dataset. The eigenvalues of the Laplacian are shown in red on the right.

In the following results, the number of iterations is fixed at $m = 6$. Table 6.1 shows the relative accuracy of the first two Ritz values. Both in $\mathbb{R}$ and in $\mathbb{Z}_N$ the eigenvalues are approximated well. The accuracy is higher in $\mathbb{R}$.

i	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lvert\lambda_i\rvert}$ $\mathbb{R}$	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lvert\lambda_i\rvert}$ $\mathbb{Z}_N$
1	6.7998e-13	3.1379e-5
2	1.5898e-14	5.0171e-9

Table 6.1: The relative error of the two smallest Ritz values. Parameters: $d = 6$, $m = 6$.

Table 6.2 shows the cosine of the angle between the Ritz vectors and the exact eigenvectors. The values show that the eigenvectors are approximated with high accuracy.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
1	1.00000000	1.00000000
2	1.00000000	1.00000000

Table 6.2: The absolute cosine of the angle between Ritz vectors $\tilde{u}_i$ and eigenvector u_i for the Wisconsin Breast Cancer dataset.

Figure 6.7 shows the cluster results on the first two Ritz vectors that were computed with Lanczos basis V.

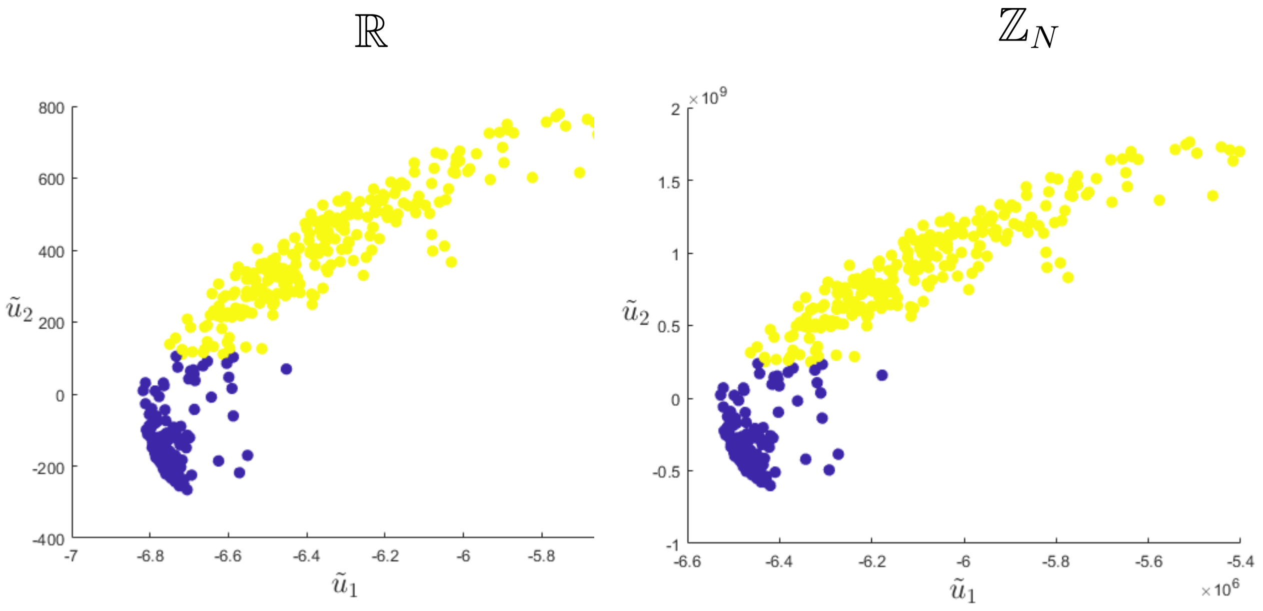


Figure 6.7: Comparison of first two eigenvectors of L for the Wisconsin Breast Cancer dataset as approximated by the Lanczos algorithm with $m = 6$ and $d = 6$ in $\mathbb{R}$ (left) and in $\mathbb{Z}_N$ (right). The colours indicate the clusters that are assigned by the k-means clustering algorithm.

Table 6.3 shows the cluster quality. Both in $\mathbb{R}$ and in $\mathbb{Z}_N$, the first two eigenvectors are approximated well enough to form the correct convex clusters.

	Lanczos $\mathbb{R}$	Lanczos $\mathbb{Z}_N$
Cluster accuracy	95.85%	95.85%
Silhouette value	0.9118	0.9118

Table 6.3: Cluster quality of the Wisconsin Breast Cancer dataset. Parameters: $k = 2$, $d = 6$, $n = 6$.

Finally, the maximum bit length is 51 in T and 76 in V.

6.2.2 Yeast5 Dataset

A scaling parameter of $d = 6$ is sufficient. After ten iterations, spurious eigenvalues show up. This is shown in Figure 6.8.

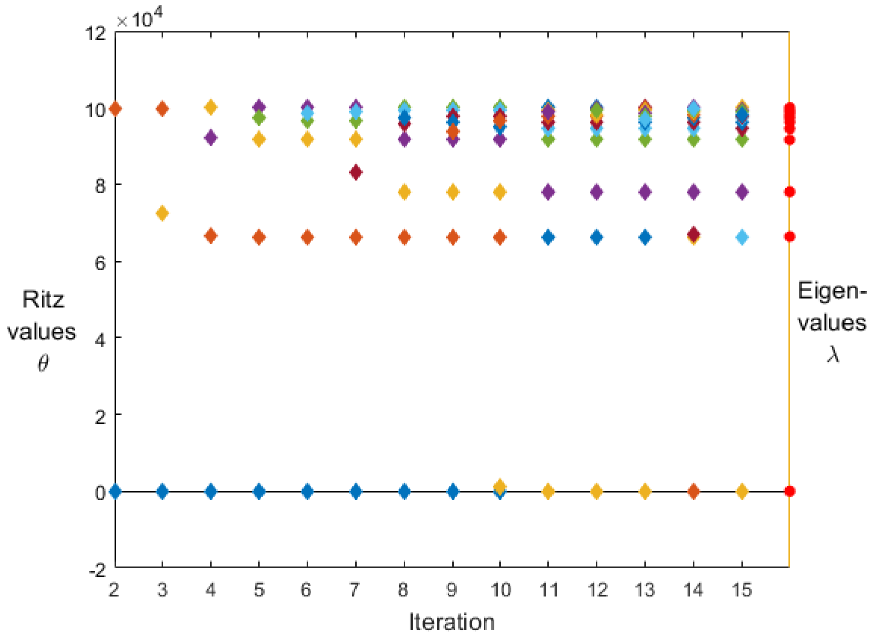


Figure 6.8: The Ritz values in the first 15 iterations of the Lanczos algorithm in $\mathbb{Z}_N$ for the Yeast5 dataset. The eigenvalues of the Laplacian are shown in red on the right.

The relative error of the first five eigenvalues after ten iterations is shown in Table 6.4.

i	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lvert\lambda_i\rvert}$ $\mathbb{R}$	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lvert\lambda_i\rvert}$ $\mathbb{Z}_N$
1	4.3646e-11	5.7123e-04
2	2.1933e-16	4.2349e-09
3	1.6780e-15	2.9113e-09
4	5.4441e-12	2.4535e-08
5	2.4629e-08	4.1245e-05

Table 6.4: The relative error of the smallest five Ritz values of the Yeast5 dataset as computed by the Lanczos algorithm in $\mathbb{R}$ and in $\mathbb{Z}_N$ with $d = 6$ after 10 iterations.

Table 6.5 shows the absolute cosine of the angles between the first five Ritz vectors and the first five eigenvectors. The eigenvectors are approximated well.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
1	1.00000000	1.00000000
2	1.00000000	1.00000000
3	1.00000000	1.00000000
4	0.99999971	0.99999985
5	0.99974208	0.99950233

Table 6.5: The absolute cosine of the angle between Ritz vectors $\tilde{u}_i$ and eigenvector u_i for the Yeast5 dataset.

The average performance of the clustering algorithm after the Lanczos phase in $\mathbb{R}$ and $\mathbb{Z}_N$ is shown in Table 6.6. An example of a clustering result is given in Figure 6.9.

	Lanczos $\mathbb{R}$	Lanczos $\mathbb{Z}_N$
Cluster accuracy	43.23%	54.69%
Silhouette value	0.5269	0.4775

Table 6.6: Clustering results after the Lanczos algorithm in $\mathbb{R}$ and $\mathbb{Z}_N$ on the Yeast5 dataset. Parameters: $d = 6$, $n = 10$.

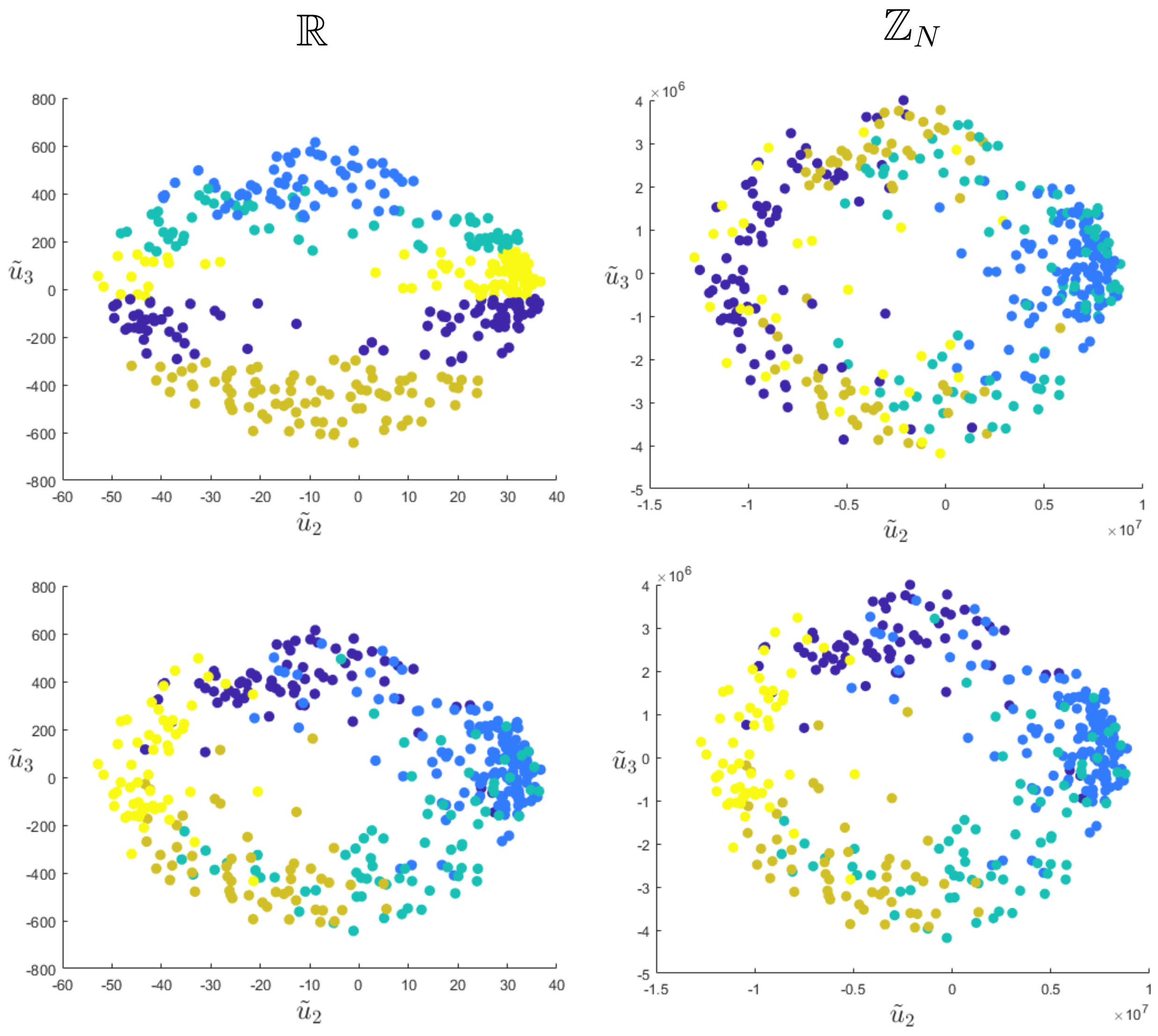


Figure 6.9: Cluster results after Lanczos in $\mathbb{R}$ (upper left) and $\mathbb{Z}_N$ (upper right). The actual clusters are indicated by the colours in the lower figures. Two of the four eigenvectors are shown.

Figure 6.9 shows that the spectral clustering algorithm has difficulties in distinguishing the cluster structure of the Ritz vectors. After the Lanczos algorithm in $\mathbb{R}$, the eigenvectors are clustered into five stripes. The dense cluster on the right of the ring is not recognized. After the Lanczos algorithm in $\mathbb{Z}_N$, the k-means step assigns a vertically striped pattern of four clusters. The fifth cluster, indicated in yellow, is spread out over the ring. The dense cluster is distinguished quite well in $\mathbb{Z}_N$. This explains the higher accuracy of the spectral clustering algorithm after Lanczos in $\mathbb{Z}_N$.

The entries of the Lanczos matrix V grow to 130 bits and the entries of T are a maximum of 50 bits.

6.2.3 Yeast10 Dataset

A scaling factor $d = 6$ is sufficient. Figure 6.10 shows the convergence of the Ritz values. After 72 iterations, the algorithm breaks down because overflow occurs, i.e. entries of more than 1024 bits are encountered. The relative error of the first ten Ritz values after 72 iterations is given in Table 6.7.

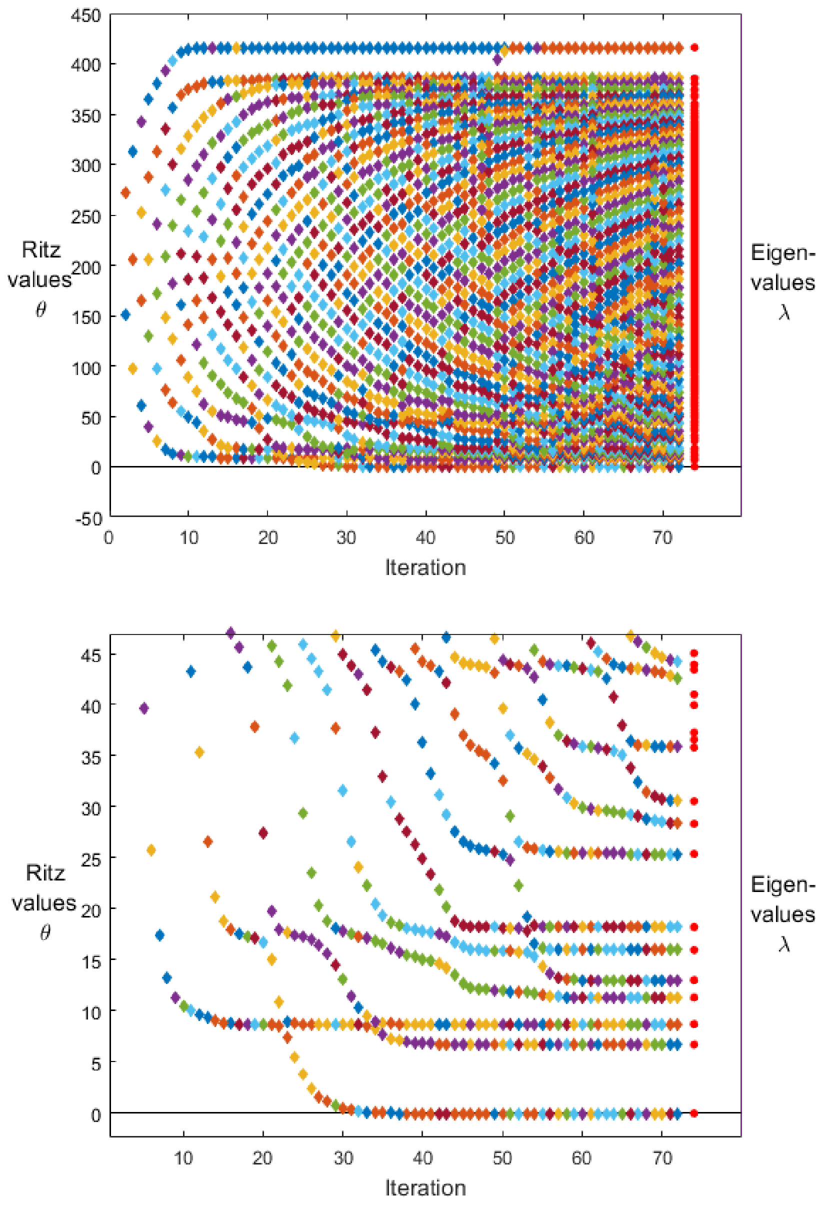


Figure 6.10: The Ritz values of T are plotted against the iteration number. The 1484 eigenvalues of L are plotted as red dots on the right side. A zoomed in plot of the convergence of the 20 smallest Ritz values is shown in the lower figure.

i	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lambda_i}$ $\mathbb{R}$	$\frac{\lvert\theta_i-\lambda_i\rvert}{\lambda_i}$ $\mathbb{Z}_N$
1	6.7998e-13	2.2589e-04
2	1.5898e-14	1.1925e-05
3	1.3129e-14	2.8286e-06
4	1.1559e-11	1.1538e-05
5	1.6783e-13	4.0184e-06
6	3.2107e-11	3.4317e-06
7	2.7197e-10	6.2687e-06
8	2.4903e-07	1.3887e-05
9	1.8286e-04	4.8851e-03
10	4.5403e-05	2.5330e-03

Table 6.7: The relative error of the smallest ten Ritz values for the Yeast10 dataset. Parameters: $d = 6$, $m = 72$.

Let us investigate the quality of the Ritz vectors, of which the accuracy is shown in Table 6.8. The small values show that the approximations are very poor: it seems that the approximations are actually perpendicular to the exact vectors. Further investigation shows that the approximated eigenvectors all point approximately in the same direction, i.e. copies of an approximated eigenvector are found. Orthogonality is lost completely. Re-orthogonalization was applied to try to improve the approximations, but this caused the algorithm to overflow before a sufficiently accurate approximation of the eigenvalues was achieved.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
1	1.3565e-07	1.9567e-03
2	9.6185e-07	1.0796e-04
3	8.0197e-06	1.6427e-04
4	2.5474e-04	2.6230e-02
5	2.2338e-05	5.9645e-03
6	1.7451e-04	2.8707e-03
7	3.4285e-04	3.1437e-03
8	2.5142e-03	5.9683e-02
9	3.7567e-02	6.5497e-01
10	9.1887e-03	2.5041e-01

Table 6.8: The absolute cosine of the angle between Ritz vectors $\tilde{u}_i$ and eigenvector u_i for the Yeast10 dataset.

The entries of V grow to 492 bits. The entries of T have a maximum bit size of 156.

6.3 The QR algorithm

The QR algorithm is applied to the matrix T that is obtained after the Lanczos algorithm in $\mathbb{R}$ or $\mathbb{Z}_N$. The performance of Algorithm 12 and the same algorithm in the real domain are compared. After p iterations of the QR algorithm, the Ritz values and the eigenvectors of T_p are computed with the **scipy.linalg.eig** function in Python. After the algorithm in $\mathbb{Z}_N$, the Ritz values are rescaled by 10^e. The eigenvectors of T_p are stored in a matrix W. The Ritz vectors are computed with $\tilde{U} = VQ_{\prod}W$.

6.3.1 Wisconsin Breast Cancer Dataset

A scaling factor $e = 20$ is required for the QR algorithm in $\mathbb{Z}_N$. Let us start by investigating the influence of using shifts on the convergence of the QR algorithm. Figure 6.11 shows the relative error of the smallest diagonal elements of T_p as approximations of the eigenvalues of L after every iteration of QR in $\mathbb{R}$ and $\mathbb{Z}_N$. A better accuracy is achieved when shifts are used. Figure 6.11 shows that the algorithms in $\mathbb{R}$ and in $\mathbb{Z}_N$ behave similarly in terms of convergence: the first eigenvalue converges steeply within 10 iterations. The second eigenvalue converges within 15 iterations. For the second eigenvalue to converge with better accuracy, the QR algorithm without shifts needs at least 200 iterations. Therefore, the QR algorithm with shifts is used to obtain subsequent results.

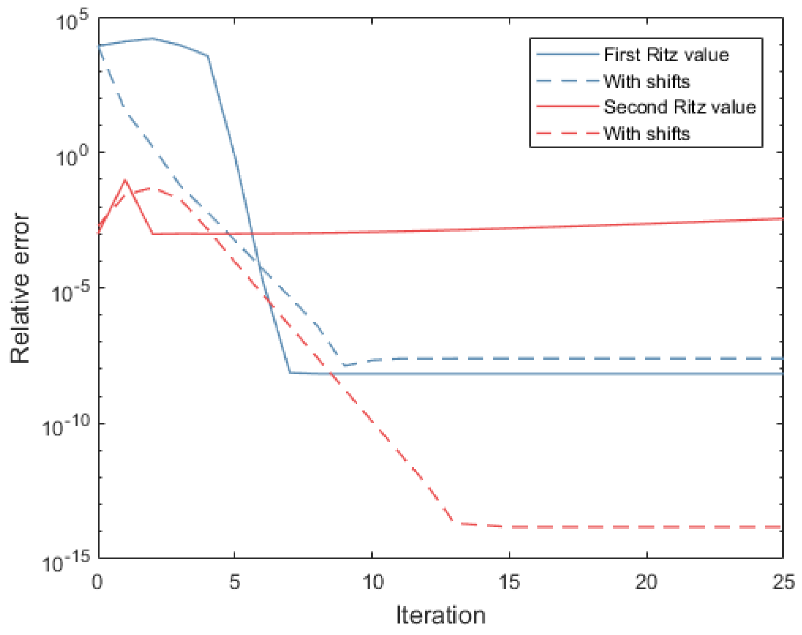


Figure 6.11: (a) The relative error of the smallest diagonal elements of T_p after every iteration of the QR algorithm in $\mathbb{R}$ with $g = 20$.

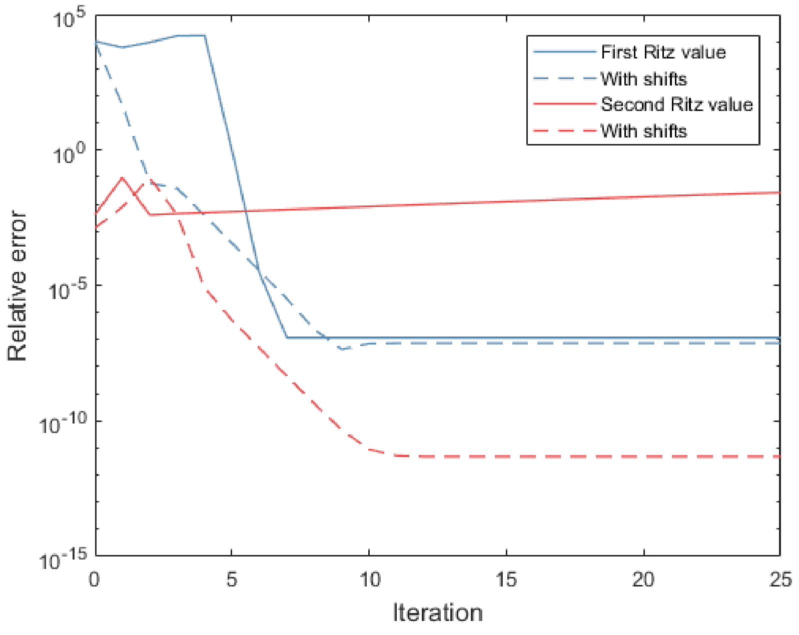


Figure 6.11: (b) The relative error of the smallest diagonal elements of T_p after every iteration of the QR algorithm in $\mathbb{Z}_N$ with $g = 20$.

The following results were obtained with 5 iterations of the QR algorithm. Table 6.9 shows the angle between the approximated eigenvectors and the exact eigenvectors. The approximation is highly accurate after 5 iterations of the QR algorithm.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
1	1.00000000	1.00000000
2	0.99999999	1.00000000

Table 6.9: The absolute cosine of the angle between Ritz vectors $\tilde{u}_i$ and eigenvector u_i for the Wisconsin Breast Cancer dataset.

Table 6.10 shows the cluster quality. The QR algorithms in $\mathbb{R}$ and in $\mathbb{Z}_N$ yield very accurate results.

	QR $\mathbb{R}$	QR $\mathbb{Z}_N$
Cluster accuracy	95.85%	95.85%
Silhouette value	0.9118	0.9118

Table 6.10: The QR algorithm performs equally well in $\mathbb{R}$ and in $\mathbb{Z}_N$ on the Wisconsin Breast Cancer dataset. Parameters: $e = 20$, $p = 5$.

Figure 6.12 shows the approximations of the first two eigenvectors of the Laplacian after the QR algorithm. The eigenvectors are approximated well enough to distinguish the two original clusters with great accuracy. Note that the pictures are rotated because the eigenvectors were approximated in the opposite direction. This does not have influence on the accuracy of the eigenvectors.

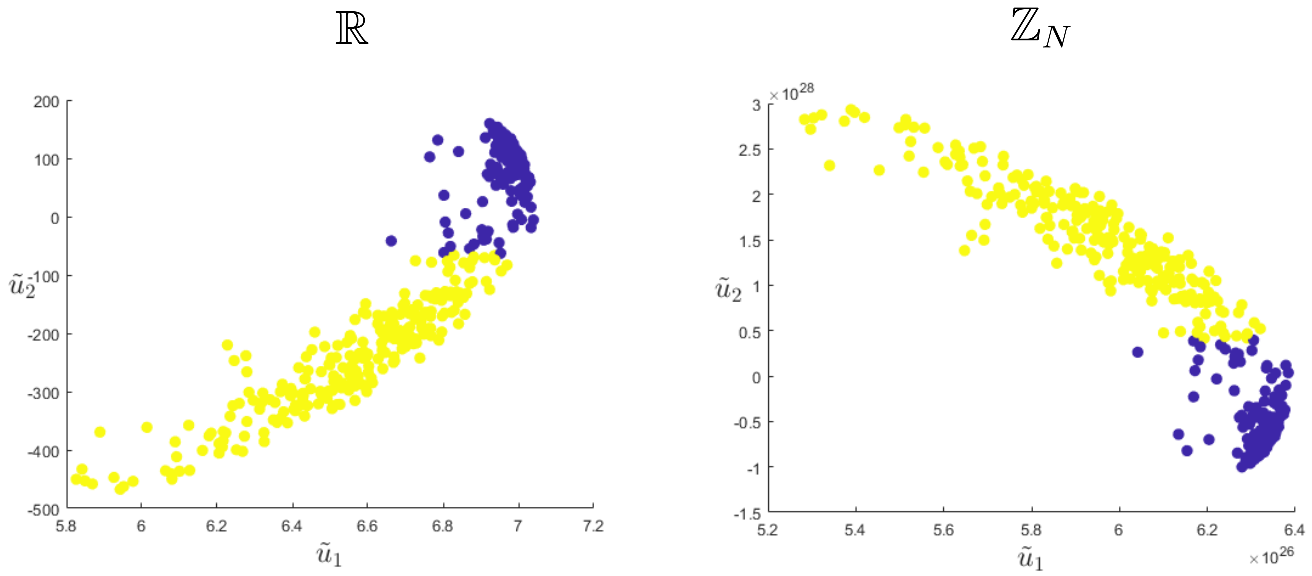
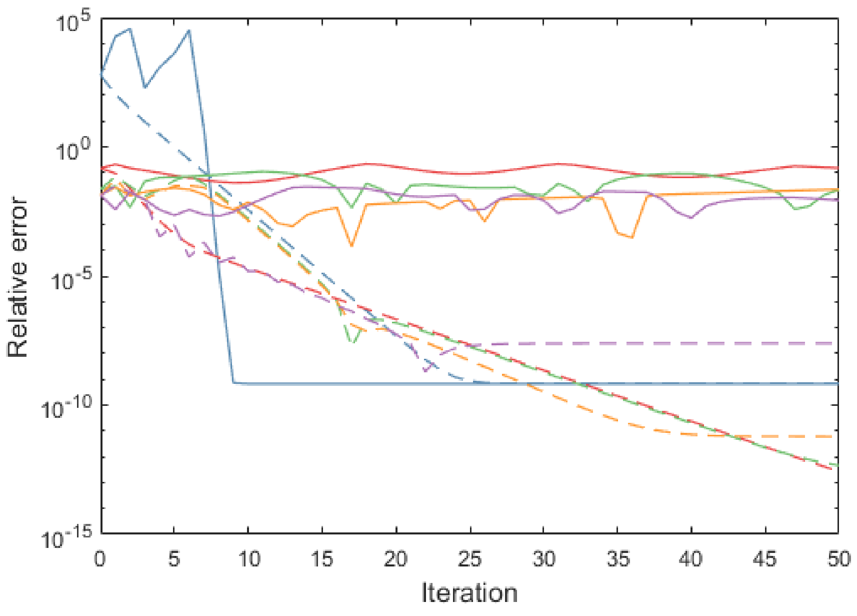


Figure 6.12: Comparison of first two eigenvectors of L for the Wisconsin Breast Cancer dataset as approximated by the QR algorithm with $p = 5$ and $e = 20$ in $\mathbb{R}$ (left) and in $\mathbb{Z}_N$ (right). The colours denote the clusters that are assigned by the k-means clustering algorithm.

The entries of $Q_\prod$ have a maximum bit length of 67. The entries of T_p become 98 bits long and the entries of R become 99 bits long.

6.3.2 Yeast5 Dataset

A scaling factor of $e = 40$ was required to reconstruct the eigenvectors of L. Figure 6.13 shows the relative error of the five smallest diagonal elements of T_p after every iteration of the QR algorithm. The eigenvalues are approximated with greater accuracy in $\mathbb{R}$. Strikingly, the first eigenvalue converges faster in the algorithm without shifts. The second to fifth eigenvalues converge to a greater accuracy within 50 iterations in the QR algorithm with shifts. The convergence of the fifth eigenvalue in the algorithm with shifts shows oscillatory behavior.



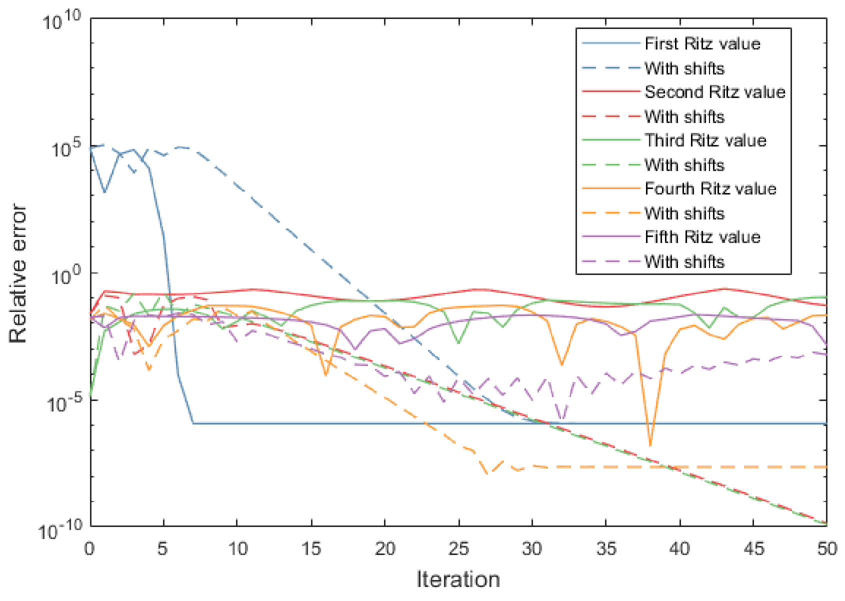


Figure 6.13: The relative error of the five smallest diagonal elements of T_m after every iteration of QR in $\mathbb{R}$ (upper figure) and in $\mathbb{Z}_N$ (lower figure) with $e = 40$. The continuous lines correspond to the QR algorithm without shifts, the dashed lines correspond to the algorithm with shifts.

Figure 6.14 shows the absolute cosine of the angle between the second to the fourth Ritz vectors and the eigenvectors that they approximate. The absolute cosine is given for the first 20 iterations of the QR algorithm in $\mathbb{Z}_N$. The first Ritz vector is not taken into account since the first eigenvector is known to be the constant eigenvector. After 15 iterations, the Ritz vectors have a constantly high accuracy. Therefore, 15 iterations of the QR algorithm were chosen to assess the cluster accuracy.

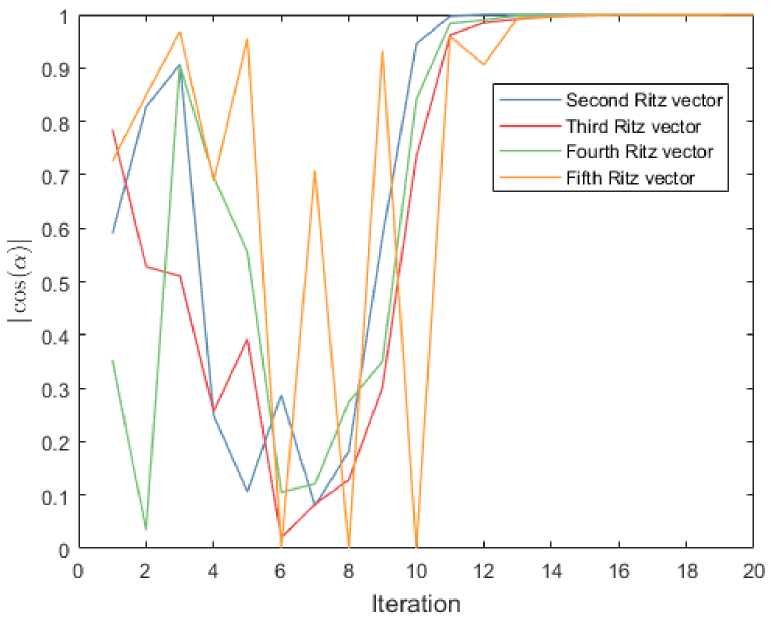


Figure 6.14: The cosine of the angle between the four Ritz vectors and the exact eigenvectors for the Yeast5 dataset for the first 20 iterations of the QR algorithm in $\mathbb{Z}_N$.

Table 6.11 shows the cluster quality for the Yeast5 dataset. The accuracy of the spectral clustering algorithm after the QR step in $\mathbb{R}$ is lower than in $\mathbb{Z}_N$. Strikingly, the clusters in $\mathbb{R}$ have a negative silhouette value. This indicates that no substantial cluster structure has been found by the k-means clustering algorithm.

	QR $\mathbb{R}$	QR $\mathbb{Z}_N$
Cluster accuracy	43.23%	55.73%
Silhouette value	-0.1687	0.3137

Table 6.11: Cluster quality after the QR algorithm in $\mathbb{R}$ and $\mathbb{Z}_N$ on the Yeast5 dataset. Parameters: $e = 40$, $m = 15$.

Figure 6.15 shows the cluster results on three of the approximated vectors. Just as the cluster results after the Lanczos algorithm, we see that the dense cluster (indicated in blue in the lower plots) is not picked up on by the k-means clustering step in $\mathbb{R}$. This cluster is better distinguished in $\mathbb{Z}_N$. This explains the higher accuracy of the results after the QR algorithm in $\mathbb{Z}_N$.

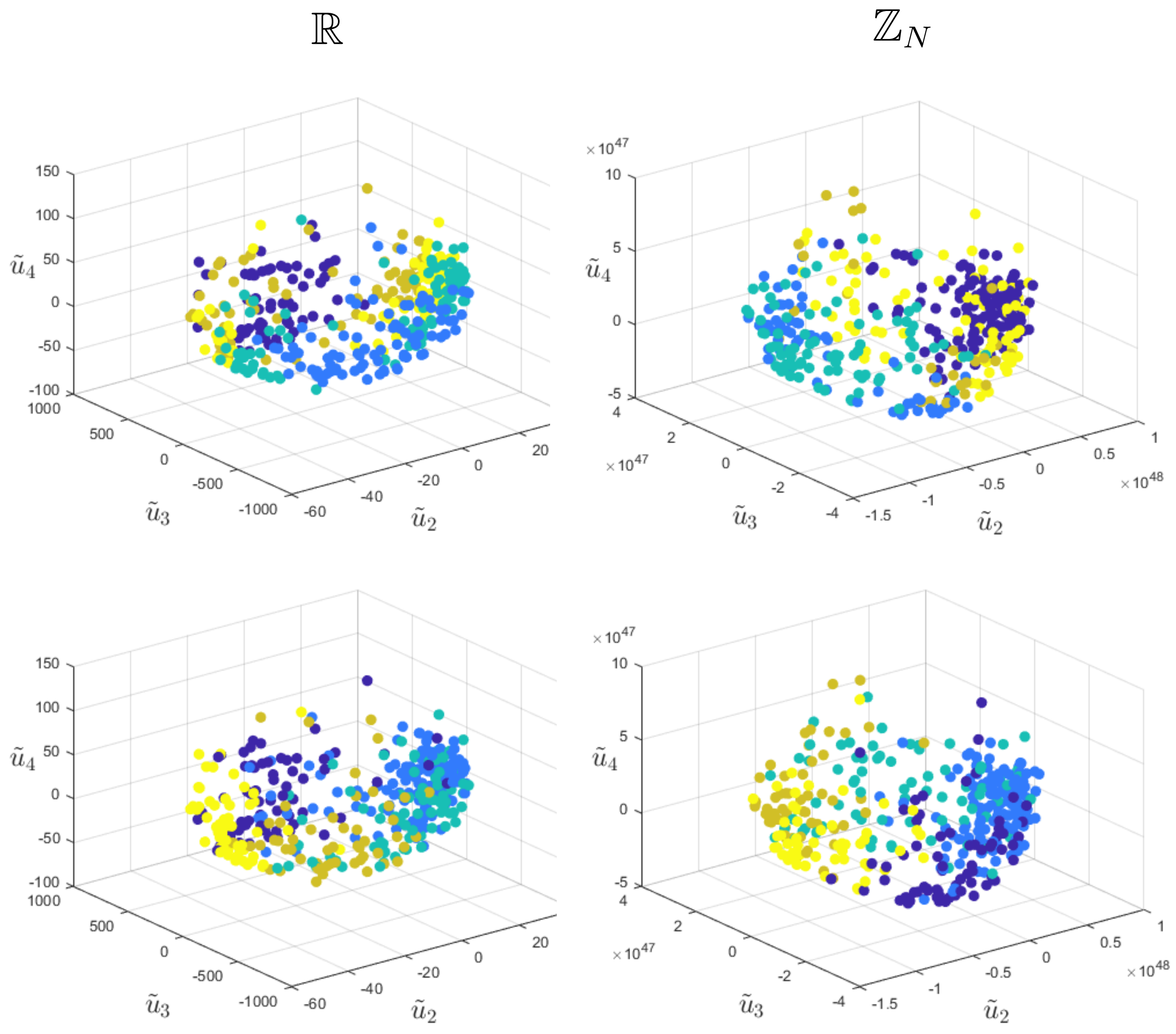


Figure 6.15: Cluster results after the QR algorithm in $\mathbb{R}$ (upper left) and $\mathbb{Z}_N$ (upper right) with $e = 40$ and $m = 15$. The actual clusters are indicated by the colours in the lower figures. Three of the four eigenvectors are shown.

The entries of Q and Q_{prod} become 101 bits large. The maximum bit length of both R and T_m is 130.

6.3.3 Yeast10 Dataset

Since it is not possible to compute Ritz vectors of the Yeast10 dataset, we will only investigate the accuracy of the diagonal elements of T_p as Ritz values. The relative error is plotted against the iterations of the QR algorithm in Figure 6.16. Strikingly, we see that the shifted QR algorithms do not converge at all in the first 200 iterations. Apparently, the shifts are not good approximations of the eigenvalues. Figure 6.17 shows the relative error of the Ritz values when the algorithm in $\mathbb{Z}_N$ is shifted after first running 15 iterations without shifts. This indeed shows that the algorithm with shifts now does converge. However, the algorithm with shifts does not perform better.

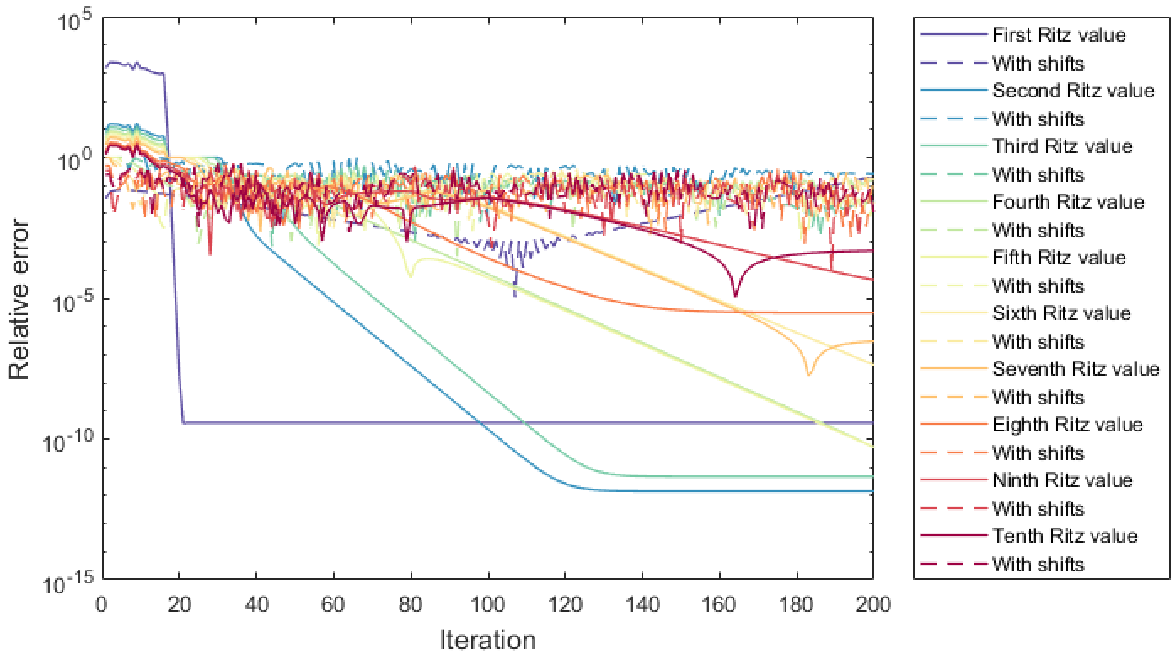


Figure 6.16: (a) The relative error of the diagonal elements of T_m after every iteration of QR in $\mathbb{R}$ with $g = 30$. The continuous lines correspond to the QR algorithm without shifts, the dashed lines correspond to the algorithm with shifts.

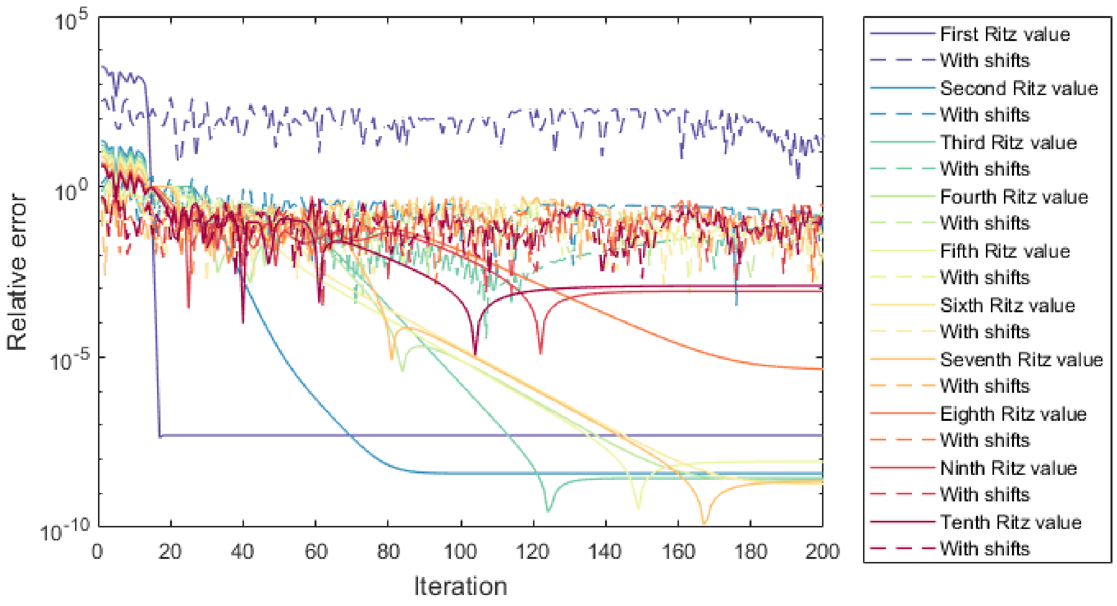


Figure 6.16: (b) The relative error of the diagonal elements of T_m after every iteration of QR in $\mathbb{Z}_N$ with $g = 30$. The continuous lines correspond to the QR algorithm without shifts, the dashed lines correspond to the algorithm with shifts.

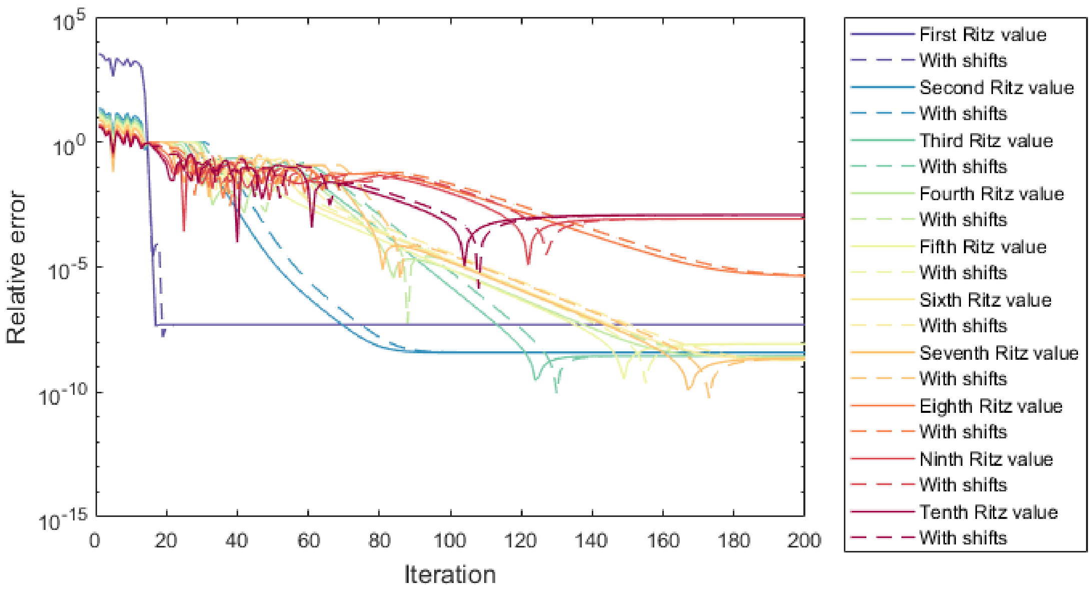


Figure 6.17: After 15 iterations without shifts, the QR algorithm in $\mathbb{Z}_N$ with shifts also works on the Yeast10 dataset.

6.4 Back substitution

The back substitution step is not tested on the Yeast10 dataset since we were not able to compute the eigenvectors of the Laplacian with the constructed Lanczos basis V. A linear system is solved for $k-1$ Ritz values, which are selected from the diagonal of T_p. The eigenvectors of T_P are stored in a matrix W. The Ritz vectors are computed with $\tilde{U} = VQ_{\prod}W$.

6.4.1 Wisconsin Breast Cancer Dataset

Figure 6.18 shows the entries of the second eigenvector of L as computed with the back substitution method in $\mathbb{R}$ and in $\mathbb{Z}_N$. The entries are plotted against their index. The colours of the data points indicate the actual clusters.

Table 6.12 shows the cluster quality in $\mathbb{R}$ and $\mathbb{Z}_N$.

	BS $\mathbb{R}$	BS $\mathbb{Z}_N$
Cluster accuracy	95.57 %	96.42%
Silhouette value	0.9122	0.9092

Table 6.12: Parameters: $d = 6$, $e = 20$, $g = 5$, $m = 6$, $p = 5$.

The absolute cosine of the angle between the Ritz vector and the second eigenvector is shown in Table 6.13.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
2	0.99987914	0.99913171

Table 6.13: The absolute cosine of the angle between $\tilde{u}_2$ and eigenvector u_2 for the Wisconsin Breast Cancer dataset.

The maximum bit length of entries of eigenvector w_2 of T_p is 10 bits. The maximum bit length of entries of the Ritz vector $\tilde{u}_2$ is 122.

6.4.2 Yeast5 Dataset

A scaling factor $g = 10$ is required for the back substitution step. Table 6.14 shows the angles between the approximations and the exact eigenvectors.

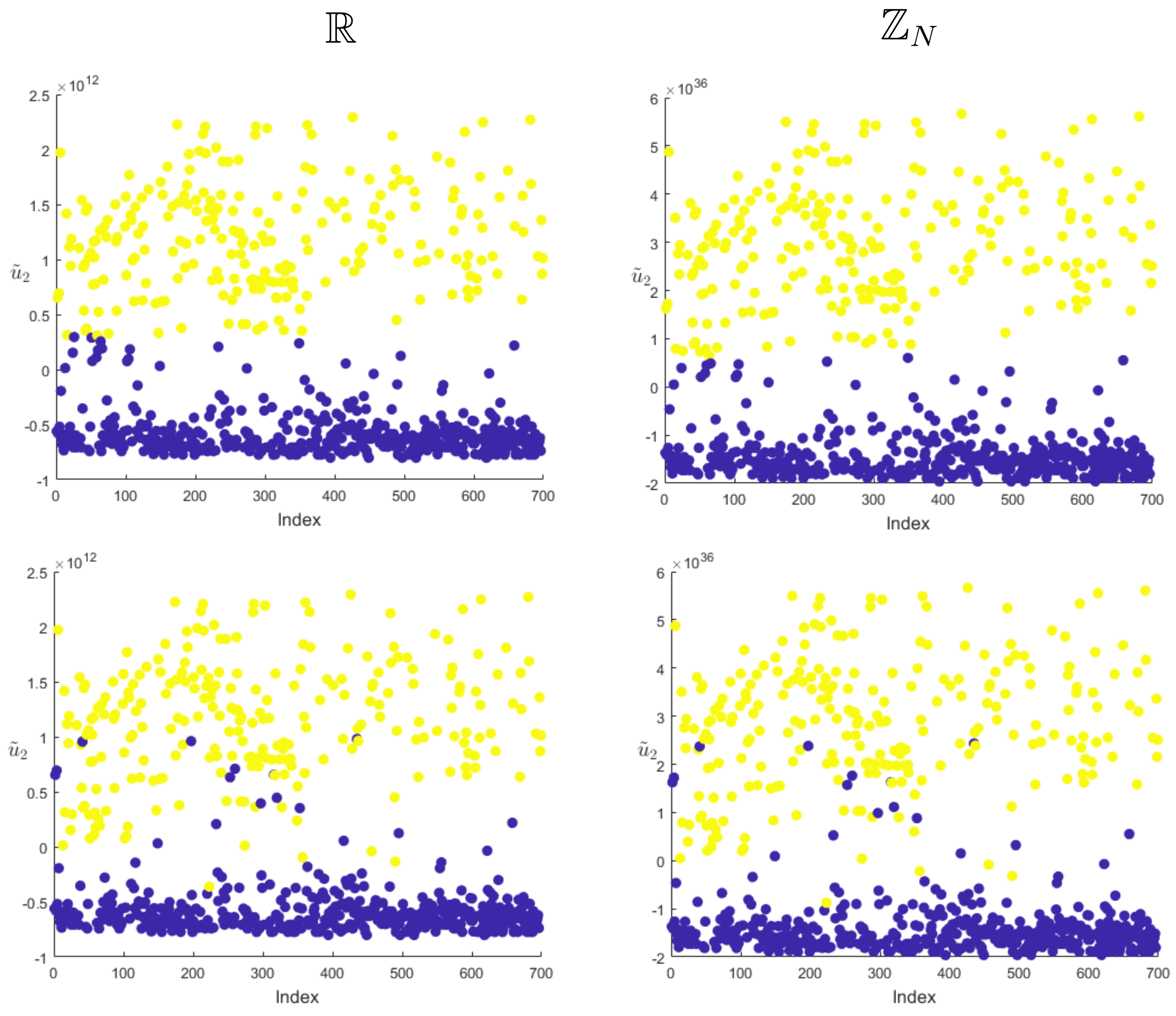


Figure 6.18: Cluster results after back substitution in $\mathbb{R}$ (upper left) and $\mathbb{Z}_N$ (upper right) with $d = 6$, $e = 20$, $g = 5$, $m = 6$, $p = 5$. The actual clusters are indicated by the colours in the lower figures. Three of the four eigenvectors are shown.

i	$\lvert\cos\alpha\rvert$ $\mathbb{R}$	$\lvert\cos\alpha\rvert$ $\mathbb{Z}_N$
2	0.99999999	0.99995171
3	1.00000000	0.99840385
4	0.99999968	0.99948435
5	0.99999964	0.99893584

Table 6.14: The absolute cosine of the angle between vectors $\tilde{u}_i$ and eigenvector u_i for the Yeast5 dataset.

Table 6.15 shows the accuracy and silhouette values after the eigenvectors are approximated entirely in $\mathbb{R}$ or $\mathbb{Z}_N$. The same scaling factor as in the QR algorithm can be used.

	BS $\mathbb{R}$	BS $\mathbb{Z}_N$
Cluster accuracy	35.94 %	35.42 %
Silhouette value	0.6820	0.6881

Table 6.15: Cluster quality of the back substitution phase on the Yeast dataset. Parameters: $d = 6$, $e = 40$, $g = 10$, $m = 10$, $p = 15$.

Figure 6.19 shows a representative example of a cluster result after the eigenvectors of the Laplacian are approximated in $\mathbb{R}$ or $\mathbb{Z}_N$. The actual clusters are also shown. This figure illustrates the complex behavior of the k-means clustering step. Although the eigenvectors are approximated well, the k-means algorithm finds a different pattern than one would expect. This determines the decline in accuracy when compared to the other cluster results of this dataset.

The entries of the eigenvectors w_i have a maximum bit length of 71 bits. The maximum bit length in the Ritz vectors is 231.

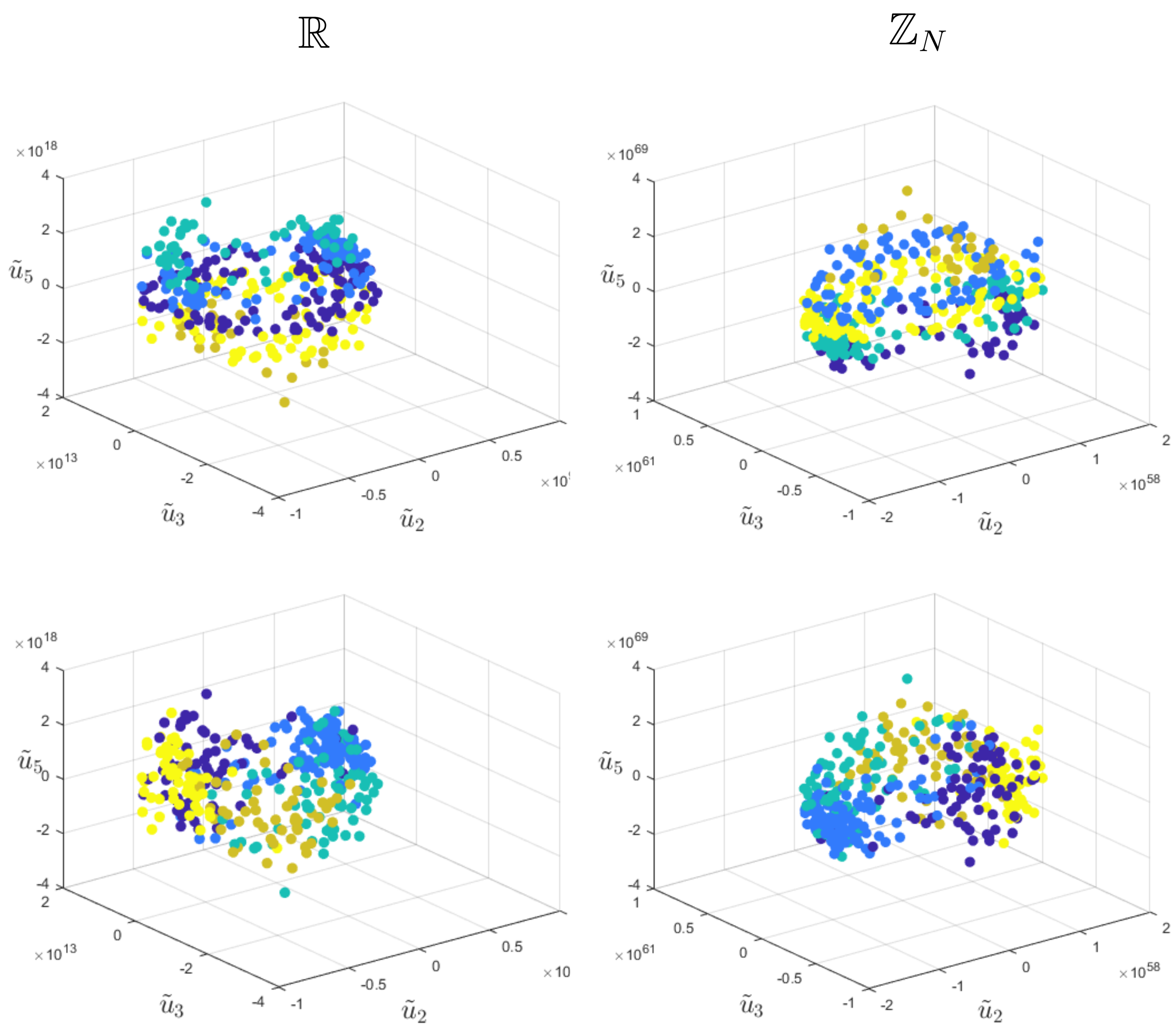


Figure 6.19: Cluster results after back substitution in $\mathbb{R}$ with 36% accuracy (upper left) and $\mathbb{Z}_N$ with 35% accuracy (upper right). The actual clusters are indicated by the colours in the lower figures. Three of the four eigenvectors are shown.

Chapter 7

Conclusion and discussion

7.1 Conclusion

The aim of this research was to answer the following two research questions:

1. How can the approximation of eigenvectors be performed in the integer domain?
2. How does this influence the performance of the spectral clustering algorithm?

To answer the first research question, three stages were distinguished in the approximation of the eigenvectors of a symmetric matrix. The Lanczos algorithm was used to transform a symmetric matrix into a tridiagonal matrix with similar eigenvalues. The QR algorithm was used to find these eigenvalues and to find an upper triangular matrix that can be used to approximate the eigenvectors. Finally, the back substitution step was performed to find the approximate eigenvectors. The Lanczos algorithm, the QR algorithm and the back substitution method were adapted to $\mathbb{Z}_N$ in such a way that they can be implemented more easily in a privacy preserving manner. The square roots were eliminated from the algorithms. It was investigated whether the number of divisions could be reduced, but overflow turned out to occur too fast with fewer divisions. A design for the privacy preserving approximation of eigenvectors was proposed.

The adapted algorithms were implemented and tested in order to answer the second research question. The influence of the approximation of eigenvectors in the integer domain on spectral clustering was investigated for three datasets with $k = 2$, $k = 5$ and $k = 10$ respectively, where k denotes the number of clusters. We can conclude from the results in Chapter 6 that the Lanczos algorithm performs well in the integer domain. Although the Ritz values are more accurate in $\mathbb{R}$, both in $\mathbb{R}$ and in $\mathbb{Z}_N$ the smallest k eigenvalues are approximated accurately enough before spurious eigenvalues occur. For $k = 2$ and $k = 5$, the eigenvectors were approximated with high accuracy. The cluster quality is higher in $\mathbb{Z}_N$ for $k = 5$, because the k-means algorithm was able to recognize clusters with various densities. For $k = 10$, orthogonality is lost and we were not able to approximate the eigenvectors. Re-orthogonalization was applied in an attempt to solve this issue, but this caused overflow to occur before a sufficiently accurate approximation of the eigenvalues was achieved.

Of the three steps in $\mathbb{Z}_N$, the QR algorithm needs the highest scaling factor. For $k = 2$ and $k = 5$, applying shifts in the QR algorithm improves the convergence speed of the eigenvalues. However, whereas we would expect quadratic convergence in the QR algorithm with shifts, linear convergence was observed instead. Again, a higher cluster quality was obtained in $\mathbb{Z}_N$. For $k = 10$, the QR algorithm performs worse when shifts are applied. The back substitution step performs as good in $\mathbb{R}$ as in $\mathbb{Z}_N$ for $k = 2$. This is also the case for $k = 5$, but both in $\mathbb{R}$ and in $\mathbb{Z}_N$ the cluster accuracy is lower than in the other algorithms, while the eigenvectors are approximated with high accuracy. We were not able to test the back substitution step for $k = 10$.

We can conclude that the eigenvalues of the Laplacian are approximated well in the integer domain. A small number of corresponding eigenvectors can be approximated with high accuracy. For a small number of clusters, a good performance of the spectral clustering algorithm was achieved. The k-means clustering step has a strong impact on the cluster quality, even when the eigenvectors are approximated well. As a higher number of clusters requires more iterations of the Lanczos algorithm, the loss of orthogonality affects the accuracy of the spectral clustering algorithm. We will now evaluate the conclusions and suggest directions for future research.

7.2 Discussion

In this section, we will first discuss the influence of the Laplacian and the k-means clustering step on the results. Next, we zoom in on some of the striking results that were encountered in Chapter 6. Moreover, improvements for the designed algorithms will be proposed. We will finish the thesis with suggestions for future research.

7.2.1 Influence of other steps

The topic of this research was the influence of the approximation of eigenvectors in the integer domain on the spectral clustering results. However, two other phases in the spectral clustering algorithm had an influence on the results: the construction of the Laplacian and the k-means clustering step. Many different similarity functions and graphs can be used in the construction of the Laplacian. Moreover, either an unnormalized or a normalized Laplacian can be used. The Laplacian had to be constructed in such a way that the multiplicity of eigenvalue 0 was 1, and that the entries were integer values. The optimal Laplacian depends strongly on the context [51]. The k-means clustering step also has a strong influence on the performance of the spectral clustering algorithm. This was shown in the plots of the cluster results: even although convex clusters would sometimes be visible to the bare eye, they would not always be detected in the k-means clustering step. A drawback of the k-means clustering algorithm is that clusters of different densities are hard to detect [19]. The algorithm is also sensitive to outliers. Moreover, the clusters should be of similar sizes.

7.2.2 Evaluation of results

The Lanczos algorithm failed on the dataset with $k = 10$. When many iterations of the Lanczos algorithm are performed, the orthogonality between the basis vectors in V is lost. While this problem is also inherent to the standard Lanczos algorithm, we hypothesize that loss of orthogonality is aggravated in the unnormalized Lanczos algorithm. Due to the lack of normalization, the entries of the columns of V grow with every iteration. Therefore, small errors also grow and many large entries are introduced outside of the diagonal of V^TV. This hypothesis could be tested by applying an alternative normalization step in the Lanczos algorithm, for example with the 1-norm.

In the QR algorithm with shifts, we expected a quadratic convergence rate. A linear convergence rate was observed instead. This may be caused by the fact that the Q are now not orthonormal but only orthogonal. Taking the lower right entry $T_{p(m,m)}$ of matrix T_p as eigenvalue approximation is based on the equality

$$\mu^{(k)} = \frac{q_m^T T_p q_m}{q_m^T q_m} = T_{p(m,m)}. \tag{7.1}$$

However, since $q_m^T q_m = 1$ now does not hold, the applied shift may not be a good enough approximation of an eigenvalue to induce quadratic convergence.

Furthermore, the QR algorithm with shifts has a grave disadvantage: the eigenvalues do no longer necessarily converge in order of magnitude on the diagonal of T_p. While the smallest eigenvalues will show up in the upper left corner, the more interior part of the spectrum will be more mixed on the diagonal. In the QR algorithm without shifts, the eigenvalues show up in a descending order on the diagonal from top to bottom. In a privacy preserving version of the QR algorithm, it is extremely convenient to know in which matrix position a certain eigenvalue will converge. Therefore, the QR algorithm without shifts may be preferable. However, we have seen that more iterations will be required.

7.2.3 Suggested improvements

Several improvements can be made to the algorithms. First of all, it was now manually checked whether spurious eigenvalues occurred in the Lanczos algorithm. The removal or detection of spurious eigenvalues by implementing a reorthogonalization method is a desirable improvement. Reorthogonalization methods were discussed in Section 3.1.4. For a privacy preserving Lanczos algorithm, full reorthogonalization as shown in (3.29) may be the most straightforward technique. Full reorthogonalization can be implemented with matrix vector products and subtraction operations. Partial reorthogonalization would require a secure comparison protocol and the intermediate computation of the Ritz vectors. A second improvement would be to implement stopping criteria. It was discussed in Section 3.1.3 that the stopping criterion of the Lanczos algorithm is generally defined by the accuracy of the Ritz vectors. Therefore, we propose to run the Lanczos algorithm for several iterations and subsequently to compute the Ritz vectors. The required number of Lanczos iterations is dependent on the desired number of eigenvectors. A stopping criterion for the QR algorithm could be implemented by computing the difference between two consecutive T_p. In the privacy preserving domain, this would require a secure comparison protocol. Third, we were not able to approximate eigenvalues with a multiplicity larger than 1 with the Lanczos algorithm. Since the multiplicity of eigenvalue 0 of the Laplacian is equal to the number of connected components in the data graph, this limits the set of Laplacians that can be analyzed. The block Lanczos algorithm could be implemented in order to solve this problem. Finally, the number of clusters k now had to be known in advance. A truly privacy preserving spectral clustering algorithm does not publish the number of clusters either. In order to find the optimal k, the algorithm could be run for different values of k. The silhouette values for the different k could be used to determine the optimal k. Another method is the *elbow method*, which finds the optimal k by looking at the percentage of variance that is explained [19]. However, this would require to run the entire spectral clustering algorithm multiple times and would therefore be extremely computationally complex.

In the privacy preserving algorithms, there is a trade-off between security and efficiency. The unnormalized Lanczos algorithm requires two divisions to be performed in every iterations. It was discussed in Section 4.5 that a privacy preserving division protocol with an encrypted divisor is computationally heavy. Therefore, a variation of the privacy preserving Lanczos algorithm could use a public inner product $v_j \cdot v_j$. This would mean that the privacy service provider learns the squared norm of the vectors v_j in every iteration. As a result, the super-diagonal entries

$$\gamma_j = (v_j \cdot v_j) \div (v_{j-1} \cdot v_{j-1}) \tag{7.2}$$

are known to the privacy service provider as well. So, only the diagonal of T is encrypted. An advantage of this approach in comparison with the entirely secure Lanczos procedure in Algorithm 10, is that a symmetric matrix T can be obtained. After all, we saw in (5.16) that the $\sqrt{\gamma_j}$ are sufficient to construct a symmetric T. When T is symmetric, the QR algorithm computes the eigenvalues and eigenvectors directly, so the back substitution step is not necessary. Furthermore, Ortega and Kaiser developed a version of the symmetric QR algorithm without square roots [34]. Further research is required to investigate whether the publication of the γ_j is a good trade-off between security and efficiency.

Finally, it was suggested in Section 5.3 that the normalization factor of (5.22) that is defined as the 1-norm without the absolute values may also be used in the QR algorithm. This would reduce the complexity of the algorithm in the privacy preserving domain. The QR algorithm in $\mathbb{Z}_N$ gave the same performance with this "semi-norm" as with the 1-norm. We would like to remark that normalizing vectors with the Euclidean norm is only necessary in two situations: the computation of the angle between vectors and the rotation of a vector.

7.2.4 Further research

If we look back at Figure 1.1, a natural suggestion for future research is to focus on the remaining phases of the design of a privacy preserving algorithm. Moreover, the second phase should be investigated more in-depth. The suggested protocols can be optimized for the specific setting. For example, the Laplacian is a symmetric matrix and therefore only half of the matrix needs to be stored.

In this research, the Paillier cryptosystem was chosen for its computational efficiency and relatively small ciphertext size. The application of other cryptosystem may be an interesting topic for future research. For example, the Damgård-Jurik cryptosystem "enables more efficient encryption of larger plaintexts than Paillier's cryptosystem" [30]. Furthermore, the possible application of data packing was mentioned in this thesis. Although the entries grow quite large during the algorithm, data packing may still provide an improvement in terms of efficiency.

Further research is required to investigate if and how data packing can be applied. Another coding technique that may be applicable is *batching*, which makes use of the Chinese Remainder Theorem [49]. A number of small integers of size m_i can be batched into one large integer of size M in the plaintext domain by computing

$$(x_1, \ldots, x_n)_B = \sum_{i=1}^{n} \mu_i \frac{M}{m_i} x_i. \tag{7.3}$$

A great advantage of batching is that it allows both the addition and multiplication with another batched integer. This may be convenient in the computation of the matrix products and inner products that are required for the algorithms. Packing only allows addition and multiplication with a constant. However, encoding a batch is computationally more complex than packing. Further research should investigate whether batching may provide an additional advantage over the packing technique.

Another possible future extension of this research is to implement a generalization of the spectral clustering method in a privacy preserving manner. The co-regularized multi-view spectral clustering method allows for multiple representations of a dataset, for example the translations of text documents in multiple languages [25]. Every view can be used to cluster the dataset, but a combination of the views can lead to more robust and accurate results. This method looks for clusters that are consistent across multiple views of the data. Complex data structures with multiple views are often encountered in the biomedical domain, in which the contribution of multi-view clustering is becoming increasingly apparent [45].

Finally, we would like to make a concluding remark with respect to further research. This thesis aimed to bridge a gap between numerical mathematics and cryptography. It was explained that some operations that are often applied in numerical mathematics, such as the square root operation, increase the complexity of a privacy preserving protocol disproportionately. Therefore, the successful application of the numerical algorithms with an alternative norm is a useful result for the field of privacy preserving data mining. More research in general is required with respect to the effect of the adaptation of numerical algorithms to the privacy preserving domain on the accuracy and convergence of the algorithms, such as was performed in [13] and [27].

Appendix A

The secure square root protocol

Liedel's secure square root protocol assumes an initial approximation that is up to 5.4 bits accurate [29]. The number of iterations of Goldschmidt's algorithm is fixed at $\theta = \lceil \log_2(\frac{k}{5.4}) \rceil$. This can ensure an accuracy of k bits. In the last iteration, the Newton-Raphson method is applied because it can eliminate accumulated errors. It should be noted that the cryptographic primitive underlying Protocol 6 is a linear secret sharing scheme, which is a different multiparty computation technique than Paillier encryption [43]. However, secret sharing also works on messages in the integer domain. In Protocol 6, a secret-shared number x is denoted by $[x]$. With the notation $\mathsf{Trunc}([x], k, f)$, the truncation of the f least significant bits of secret-shared number x with total bit length k is meant. The goal of Protocol 6 in this thesis is merely to give an indication of the subprotocols that a secure square root solution would require, so that the relative complexity of the square root operation becomes clear.

Protocol 6: $[\sqrt{x}] \leftarrow \mathsf{sqrt}([x], k, f)$ [29]

Input : Initial encrypted approximation $[y_0] = [\frac{1}{\sqrt{x_0}}]$.

Output: Encrypted approximation $[g]$ of $[\sqrt{x}]$.

```
1  $\theta \leftarrow \lceil \log_2(\frac{k}{5.4}) \rceil$
2  $[g_0] \leftarrow \mathsf{Mult}([y_0], [x])$
3  $[g_0] \leftarrow \mathsf{Trunc}([g_0], k, f)$
4  $[h_0] \leftarrow \mathsf{PubDiv}([g_0], 2)$
5  $[gh] \leftarrow \mathsf{Mult}([g_0], [h_0])$
6  $[gh] \leftarrow \mathsf{Trunc}([gh], k, f)$
7  for $i = 1, \ldots, \theta - 2$ do
8      $[r] \leftarrow [\frac{3}{2} \cdot 2^f] - [gh] = [\frac{3}{2} \cdot 2^f] \cdot [gh]^{-1}$
9      $[g] \leftarrow \mathsf{Mult}([g], [r])$
10     $[h] \leftarrow \mathsf{Mult}([h], [r])$
11     $[g] \leftarrow \mathsf{Trunc}([g], k, f)$
12     $[h] \leftarrow \mathsf{Trunc}([h], k, f)$
13     $[gh] \leftarrow \mathsf{Mult}([g], [h])$
14     $[gh] \leftarrow \mathsf{Trunc}([gh], k, f)$
15 end
16 $[r] \leftarrow [\frac{3}{2} \cdot 2^f] - [gh] = [\frac{3}{2} \cdot 2^f] \cdot [gh]^{-1}$
17 $[h] \leftarrow \mathsf{Mult}([h], [r])$
18 $[h] \leftarrow \mathsf{Trunc}([h], k, f)$
19 $[H] \leftarrow 4 \cdot [h]^2 = \mathsf{Mult}([h]^4, [h])$
20 $[H] \leftarrow \mathsf{Mult}([H], [x])$
21 $[H] \leftarrow [3 \cdot 2^{2f}] - [H] = [3 \cdot 2^{2f}] \cdot [H]^{-1}$
22 $[H] \leftarrow \mathsf{Mult}([h], [H])$
23 $[g] \leftarrow \mathsf{Mult}([H], [x])$
24 $[g] \leftarrow \mathsf{PubDiv}([g], 2)$
```

References

[1] M. Beye, Z. Erkin, and R. L. Lagendijk. Efficient privacy preserving k-means clustering in a three-party setting. In *IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2011.

[2] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 486–497. ACM, 2007.

[3] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[4] E. Carlen. Lecture notes on the non–symmetric eigenvalue problem. `http://www.math.wustl.edu/~wick/teaching/Math2605Notes/chap4.pdf`, 2003.

[5] P. Ciarlet and J. Lions. *Handbook of Numerical Analysis.* North-Holland, 1990.

[6] J. Cullum and R. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. 1: Theory.* Society for Industrial and Applied Mathematics, 2002.

[7] M. Dahl, C. Ning, and T. Toft. On secure two-party integer division. In *International Conference on Financial Cryptography and Data Security*, pages 164–178. Springer, 2012.

[8] C. Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. *SIAM Journal on Numerical Analysis*, 7(1):1–46, 1970.

[9] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Privacy-preserving user clustering in a social network. In *Information Forensics and Security, 2009. WIFS 2009. First IEEE International Workshop on*, pages 96–100. IEEE, 2009.

[10] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing.

IEEE Transactions on Information Forensics and Security, 7(3):1053–1066, 2012.

[11] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Privacy-preserving distributed clustering. *EURASIP Journal on Information Security*, 2013(1):1–15, 2013.

[12] P. Failla and M. Barni. Gram-schmidt orthogonalization on encrypted vectors. In *Trustworthy Internet*, pages 93–103. Springer, 2011.

[13] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. *Automata, Languages and Programming*, pages 927–938, 2001.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[15] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2009.

[16] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, 1996.

[17] K. Hafner. And if you liked the movie, a Netflix contest may reward you handsomely. `http://www.nytimes.com/2006/10/02/technology/02netflix.html`, October 2006.

[18] D. Hamad and P. Biela. Introduction to spectral clustering. `http://lagis-vi.univ-lille1.fr/~lm/classpec/reunion_28_02_08/Introduction_to_spectral_clustering.pdf`, 2008.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer New York, 2009.

[20] S. J. A. Hoogh, de. *Design of large scale applications of secure multiparty computation: secure linear programming.* PhD thesis, Eindhoven University of Technology.

[21] P. Horton and K. Nakai. A probabilistic classification system for predicting the cellular localization sites of proteins. In *ISMB*, volume 4, pages 109–115, 1996.

[22] A. Jeckmans, Q. Tang, and P. Hartel. Privacy-preserving collaborative filtering based on horizontally partitioned dataset. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 439–446. IEEE, 2012.

[23] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* CRC press, 2014.

[24] L. Kuang, L. Yang, J. Feng, and M. Dong. Secure tensor decomposition using fully homomorphic encryption scheme. *IEEE Transactions on Cloud Computing*, 2015.

[25] A. Kumar, P. Rai, and H. Daume. Co-regularized multi-view spectral clustering. In *Advances in neural information processing systems*, pages 1413–1421, 2011.

[26] R. J. Lambert. *Computational aspects of discrete logarithms.* PhD thesis, University of Waterloo, 1997.

[27] R. Lazzeretti, T. Pignata, and M. Barni. Piecewise function approximation with private data. *IEEE Transactions on Information Forensics and Security*, 11(3):642–657, 2016.

[28] M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

[29] M. Liedel. Secure distributed computation of the square root and applications. In *International Conference on Information Security Practice and Experience*, pages 277–288. Springer, 2012.

[30] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):1–39, 2009.

[31] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125. IEEE, 2008.

[32] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[33] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[34] J. M. Ortega and H. F. Kaiser. The LLT and QR methods for symmetric tridiagonal matrices. *The Computer Journal*, 6(1):99–103, 1963.

[35] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices.* PhD thesis, University of London, 1971.

[36] C. C. Paige. Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

[37] C. C. Paige. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear algebra and its applications*, 34:235–258, 1980.

[38] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[39] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516. ACM, 1999.

[40] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of computation*, 33(145):217–238, 1979.

[41] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[42] S. Sanghavi. Lecture notes on Large scale learning. `http://users.ece.utexas.edu/~sanghavi/courses/scribed_notes/`, 2013.

[43] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[44] K. Tasdemir, Y. Moazzen, and I. Yildirim. Geodesic based similarities for approximate spectral clustering. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 1360–1364. IEEE, 2014.

[45] E. Tsivtsivadze, H. Borgdorff, J. van de Wijgert, F. Schuren, R. Verhelst, and T. Heskes. Neighborhood co-regularized multi-view spectral clustering of microbiome data. In *IAPR International Workshop on Partially Supervised Learning*, pages 80–90. Springer, 2013.

[46] A. Ultsch. Clustering with som: U*c. pages 75–82, 2005.

[47] D. Verma and M. Meila. A comparison of spectral clustering algorithms. *University of Washington Tech Rep UWCSE030501*, 1:1–18, 2003.

[48] T. Veugen. Encrypted integer division. In *IEEE International Workshop on Information Forensics and Security*, pages 1–6. IEEE, 2010.

[49] T. Veugen. Efficient coding for computing with encrypted data. In *IEEE Transactions on Information Forensics and Security*. IEEE, 2017. Unpublished article.

[50] T. Volkhausen. The paillier cryptosystem: A mathematical introduction. `http://www2.cs.uni-paderborn.de/cs/ag-bloemer/lehre/proseminar_WS2005/material/Volkhausen_Ausarbeitung.pdf`, 2006.

[51] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[52] H. S. Warren. *Hacker's Delight*. Addison-Wesley, 2012.

[53] D. S. Watkins. The QR algorithm revisited. *SIAM review*, 50(1):133–145, 2008.

[54] Y. Yang, H. T. Shen, F. Nie, R. Ji, and X. Zhou. Nonnegative spectral clustering with discriminative regularization. In *AAAI*, pages 2–4, 2011.

[55] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[56] K. Y. Yeung. Model-based clustering and data transformations for gene expression data, 2006.