Imperial College London

Final Year Project Report

Community Detection in Networks

Author:
Hesam Ipakchi

Supervisor:
Dr. Moez Draief
Second marker:
Dr. Jeremy Pitt

This report is submitted in fulfilment of the requirements
for the degree of *MEng Electronic and Information Engineering*
in the
Department of Electrical and Electronic Engineering
Imperial College London

May 2014

Abstract

Write Abstract here...

Acknowledgements

Write Acknowledgements here...

Contents

Abstract ii

Acknowledgements iii

Notation viii

1 Introduction 1

2 Background 3
- 2.1 Graph Theory Preliminaries . . . 3
- 2.2 Community Structure in Networks . . . 6
  - 2.2.1 Planted Partition Model . . . 8
  - 2.2.2 Hidden Clique Model . . . 11
- 2.3 Financial Networks . . . 15
  - 2.3.1 Prices and Returns of Financial Assets . . . 15
  - 2.3.2 Mean-Variance Portfolio Theory . . . 17
  - 2.3.3 Constructing Financial Networks . . . 19

3 Community Detection Algorithms 22
- 3.1 Spectral Clustering . . . 22
- 3.2 Modularity-based Optimisation . . . 23
  - 3.2.1 Greedy Algorithm . . . 25
  - 3.2.2 Simulated Annealing . . . 25
  - 3.2.3 Extremal Optimisation . . . 27
  - 3.2.4 Spectral Algorithm . . . 27
- 3.3 Belief Propagation Algorithm . . . 29
- 3.4 NLPI and AMP Algorithms . . . 29
  - 3.4.1 Non-linear Power Iteration . . . 29
  - 3.4.2 Approximate Message Passing . . . 30

4 Experiments on Synthetic Data 31
- 4.1 Spectral Clustering and Modularity-Optimisation Algorithms . . . 31
- 4.2 Belief Propagation Algorithm . . . 36
- 4.3 NLPI and AMP Algorithms . . . 36

5 Community Detection in Financial Networks **40**
5.1 Constructing the Real-world Financial Network 40
5.1.1 Random Matrix Theory 41
5.2 Community Detection Algorithms 44
5.2.1 Modularity Optimisation Methods 44
5.2.2 Modified Modularity Optimisation Method 46
5.3 Synthetic Data Testing 47
5.4 Application to Real-world Financial Network 48

A List of Stocks **50**

Bibliography **53**

List of Figures

2.1 Visualisations of example undirected graphs 5
2.2 Plots of adjacency matrices of graph generated by planted partition model 9
2.3 Visualisation of a graph generated by the planted partition model 10
2.4 Plots of adjacency matrices of graph generated by hidden clique model . . 12
2.5 Visualisation of a graph generated by the hidden clique model 12
2.6 Plots illustrating spectral phase transition of Wigner Matrices 14
2.7 Example plot for price and logarithmic return 17
2.8 Example plot for a correlation matrix 21

3.1 Visualisation of spectral clustering embedding 24
3.2 Illustration of greedy algorithm for modularity optimisation 26

4.1 Plot of overlap for spectral clustering and modularity methods in the dense regime. 33
4.2 Plot of overlap for spectral clustering and modularity methods in the sparse regime. 34
4.3 Visualisation of spectral clustering embedding in sparse and dense regimes 35
4.4 Plot of accuracy for NLPI algorithm 38
4.5 Plot of accuracy for AMP algorithm 39

5.1 Plot of expected return against volatility for 80 FTSE 100 stocks 41
5.2 Plots of empirical and RMT predicted eigenvalue spectrum 43
5.3 Plots of synthetically generated benchmark correlation matrices 48
5.4 Communities of the FTSE 100 data generated using the greedy algorithm. 49
5.5 Communities of the FTSE 100 data generated using the spectral relaxation method. 49
5.6 Communities of the FTSE 100 data generated using the spectral clustering algorithm based on the modified modularity matrix. 49

List of Tables

A.1 List of FTSE 100 Stocks studied 50

Notation

$\lvert\mathcal{S}\rvert$	Cardinality of the set $\mathcal{S}$
$\mathbb{1}_{\mathcal{S}}$	Indicator variable over the set $\mathcal{S}$
y	Scalar y
$\lvert y\rvert$	Absolute value of y
$\boldsymbol{v}$	Vector $\boldsymbol{v}$
$\lVert\boldsymbol{v}\rVert$	Euclidean norm of $\boldsymbol{v}$
$\mathbf{M}$	Matrix $\mathbf{M}$
M_{ij}	The element of the matrix $\mathbf{M}$ at row i and column j
$\mathbf{M}^T$	Transpose of the matrix $\mathbf{M}$
$\lvert\mathbf{M}\rvert$	Determinant of the matrix $\mathbf{M}$
$Tr(\mathbf{M})$	Trace of the matrix $\mathbf{M}$
$\mathbb{E}(X)$	Expected value of X
$Var(X)$	Variance of X
$\exp(x)$	Exponential function
$\log(x)$	*natural logarithm* of x (logarithm to the base e)

Chapter 1

Introduction

Networks have been studied extensively to model many interesting complex systems, including the Internet, social networks, financial networks and biological networks [14,42,45]. Any network consists of *nodes* which represent items of interest, and *edges* which represent the connectivity between pairs of nodes. For example, considering social networks, nodes are the users and the edges correspond to interactions between the users. An interesting feature many networks exhibit is *community structure*, which involves the natural dividing of nodes into groups, called *communities*, where there are denser connections within a group, and sparser connections between different groups [14, 15, 22, 42]. This particular type of community structure is also known as *assortative* [42]. For instance, social networks contain communities corresponding to real-life communities consisting of the members, such as friendship or family circles. The problem of detecting communities within networks is known as *community detection*, and algorithms are developed as a solution.

In order to provide a theoretical setting to test and compare different community detection algorithms, generative models of random graphs are very useful, and one such commonly used model is the *stochastic block model* [35, 42]. We will investigate several community detection algorithms, and will use different generative models in order to analyse and reason about them.

The underlying ingredients of the community detection algorithms have other interesting applications also, including the analysis of time series data within the context of financial networks. The aim involves seeking groups of correlated financial assets that can be used in *mean variance portfolio optimisation*. We consider the application of community

detection algorithms to real-world financial networks, in order find groups of correlated stocks found on the FTSE 100 exchange.

This project aims are twofold; firstly, we investigate and study different community detection algorithms, and secondly, use these to motivate modified techniques in order to detect communities within the financial networks setting using real-world data.

The rest of the report is organised as follows. In chapter 2, we provide technical background in graph theory, community structure within networks and financial networks required to understand the concepts investigated throughout the report. In chapter 3, several community detection algorithms, available in the literature, are introduced and explained. Chapter 4 details our experiments of several community detection algorithms on synthetic data and provides a summary of conclusions drawn in comparing the algorithms. In chapter 5, we motivate modified community detection algorithms for financial networks, and apply them to real-world data.

Chapter 2

Background

In this chapter we will describe all the technical background required to understand and detail the different settings we investigate. Initially, we will highlight some basic results in graph theory. Then, we will outline the problem of community detection and present three models used to generate random graphs with community structure to be used as a testing playground for algorithms. Following this, we will discuss basic concepts within finance required to understand the behaviour of financial assets that will provide the motivation for applying community detection algorithms to financial networks.

2.1 Graph Theory Preliminaries

We assume the reader is familiar with some basic concepts in linear algebra such as matrix multiplication, eigenvectors and eigenvalues of matrices. Rather, we will cover some basic tools within spectral graph theory using definitions from [14,17,22,34]. Spectral graph theory is the study of graphs through the eigenvalues and eigenvectors of matrices associated with the graphs [34]. We begin by defining some basic notions about graphs.

Definition 2.1. A *graph* $\mathcal{G}$ is a pair of sets (V,E), where V is a set of vertices or nodes and $E \subset V^2$, the set of unordered pairs of elements of V. The elements of E are called edges or links.

Definition 2.2. A graph $\mathcal{G} = (V, E)$ is called *undirected* if for all $v, w \in V$: $(v, w) \in E \iff (w, v) \in E$. Otherwise, $\mathcal{G}$ is called *directed*.

Definition 2.3. A *weighted* graph is a graph where a number (weight) is assigned to each edge.

We will assume, without loss of generality, that $V = \{1, \ldots, n\}$. See figure 2.1a for an example of an undirected graph with seven vertices and eleven edges.

Definition 2.4. A graph $\mathcal{G}' = (V', E')$ is a *subgraph* of $\mathcal{G} = (V, E)$ if $V' \subset V$ and $E' \subset E$. If $\mathcal{G}'$ contains all edges of $\mathcal{G}$ that join vertices of V', one says that the subgraph $\mathcal{G}'$ is induced or spanned by V'.

Definition 2.5. A partition of the vertex set V in two subsets S and $V - S$ is called a *cut*. The cut size is the number of edges of $\mathcal{G}$ joining vertices of S with vertices of $V - S$.

Definition 2.6. Two vertices are *adjacent* or *neighbours* if they are connected by an edge. The set of neighbours of a vertex v is called *neighbourhood*, and denoted by $\Gamma(v)$.

Definition 2.7. The *degree* d_v of a vertex v is the number of its neighbours, $|\Gamma(v)|$.

We will be interested in using certain graphs is the models, such as bipartite graphs.

Definition 2.8. A *bipartite* graph, is a graph whose vertices can be decomposed into two disjoint sets such that no two vertices within the same set are adjacent.

Definition 2.9. A *clique* of an undirected graph is a subset of its vertices such that every pair of vertices in the subset are adjacent in the graph.

An example of an undirected bipartite graph with nine vertices and eight edges is shown in figure 2.1b, whilst an example of a clique within an undirected graph is shown in figure 2.1c.

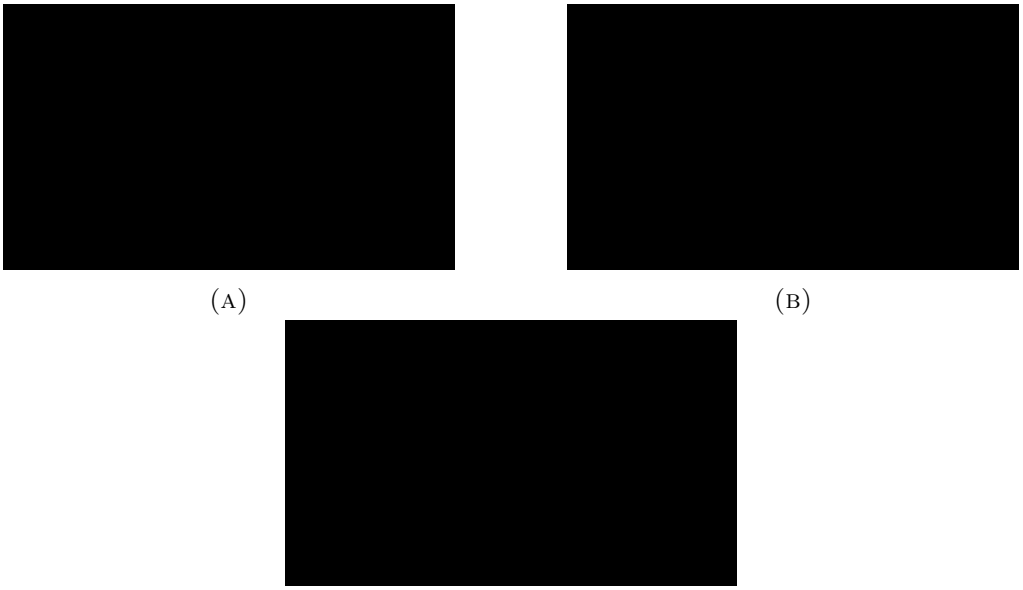
There is a very close connection between graphs and matrices, since the whole information about the topology of a graph can be entailed in matrix form, called the *adjacency matrix*.

Definition 2.10. The *adjacency matrix*, $\mathbf{A} \in \{0, 1\}^{n \times n}$ of a graph $\mathcal{G} = (V, E)$, is a $n \times n$ matrix whose element A_{ij} equals 1 if there exists an edge joining vertices i and j in $\mathcal{G}$, and zero otherwise.

From definition 2.10, it follows that elements of the adjacency matrix, $\mathbf{A}$, can be written as

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Note that the sum of elements of the i-th row of the adjacency matrix yields the degree of node i of the graph, $d_i = \sum_j A_{ij}$. Also, the adjacency matrix is symmetric if the graph is undirected.



(A) (B)

(C)

FIGURE 2.1: A set of visualisations of undirected graphs. In (a), the graph has seven nodes and eleven edges. In (b), a bipartite graph, with nine nodes (elements of disjoint sets are coloured black and red denoting membership) and eight edges, is shown. In (c), an undirected graph, with six nodes and six edges is shown, where the nodes coloured red form a clique within the graph.

Definition 2.11. The *weighted adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{n \times n}$ of a weighted graph $\mathcal{G} = (V, E)$, is a $n \times n$ matrix whose element A_{ij} equals the weight of the edge connecting nodes i and j, if it exists, and zero otherwise.

There are other matrices that have also been studied extensively in spectral graph theory, including the Laplacian which is applied in topics such as graph partitioning, synchronisation and graph connectivity [22].

Definition 2.12. The *degree matrix*, $\mathbf{D}$, of a graph $\mathcal{G} = (V, E)$, is a $n \times n$ diagonal matrix whose element D_{ii} equals the degree of vertex i.

From definition 2.12, it follows that elements of the degree matrix, $\mathbf{D}$, can be written as

$$D_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.13. The matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is called the *unnormalised Laplacian matrix*.

From definition 2.13, it follows that elements of the unnormalised Laplacian matrix of a graph $\mathcal{G} = (V, E)$, $\mathbf{L}$, can be written as

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.14. The matrix $\widetilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is called the *normalised Laplacian matrix*, where $\mathbf{I}$ is the $n \times n$ identity matrix.

Note that from definitions 2.13 and 2.14, the normalised Laplacian matrix can also be written as $\widetilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$.

An important property of adjacency and Laplacian matrices is their spectra, which we will use, later in the report, to motivate and develop a spectral clustering algorithm for community detection.

Definition 2.15. The *spectrum* of a graph $\mathcal{G}$ is the set of eigenvalues of its adjacency matrix, $\{\lambda_1, \ldots, \lambda_n\}$.

Definition 2.16. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of a matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$. The *spectral radius* is defined as $\rho(\mathbf{M}) = \max_i |\lambda_i|$.

2.2 Community Structure in Networks

An intuitive notion of communities within graphs involves the assignment of nodes to communities such that there are denser connections between nodes belonging to the same community, and sparser connections between nodes belonging to different communities. If a graph exhibits this property, it is said to contain assortative community structure [14, 15, 22, 42]. For instance, within social networks where nodes are users and edges between nodes represent interactions between the users, community structure within the graph corresponds to real-life communities consisting of the users. The aim of community detection algorithms is to estimate or recover the node assignments. The algorithms need to be efficient due to the large size of graphs in real-world networks, so we require the computationally complexity to not be worse than nearly linear in the number of edges in the graph (approximately $O(n^2 \log n)$ where n represents the number of nodes in the graph).

Within the literature, the terms *groups* and *clusters* are synonymous with communities, and as such we will use all three terms interchangeably through the report; so the reader should note all these terms refer to the same notion of communities in graphs.

In order to help provide a setting where different algorithms may be compared, we wish to study particular models which generate random graphs. One popular model is called the stochastic block model. Many special cases of this model have been studied, but we consider two versions, available in the literature. Firstly, there is a model considered by Decellle et al. [42] and Nadakuditi et al. [35], also known as the *planted partition model*. Secondly, there is a model used by Montanari [46, 48], which we will refer to as the *hidden clique model*. We emphasise that we do not exclusively focus on detecting cliques for the latter model, but the name is simply convenient for reference in this report.

Let us define the stochastic block model following Decellle et al. [42]. The stochastic block model has parameters: k, n_a, $\mathbf{P}$. k represents the number of communities (or groups), n_a refers to the expected fraction of nodes within each group a, for $1 \leq a \leq k$, and a $k \times k$ matrix $\mathbf{P}$ whose element P_{ab} equals the probability of an edge occurring between nodes belonging to groups a and b. It is known as the *affinity matrix*. We proceed to generate a random directed graph, $\mathcal{G}$, consisting of n nodes. Firstly, though, assign, to each node i of the graph, $\sigma_i \in \{1, \ldots, k\}$, a label indicating which community the node belongs to. These labels are chose independently, where, for each node i, $\Pr(\sigma_i = a) = n_a$. Let $\boldsymbol{u} = [\sigma_1, \ldots, \sigma_n]^T$ be the *ground-truth node assignments* of the graph. The random graph is generated to have an adjacency matrix, $\mathbf{A}$, whose elements are defined by

$$A_{ij} = \begin{cases} 0 & \text{if } i = j \\ X & \text{otherwise} \end{cases} \tag{2.1}$$

where $X \sim Be(P_{\sigma_i, \sigma_j})$.

This formulation matches the intuition of community structure, that the connectivity between two nodes depends solely on the community memberships of the two nodes. Note, also, that we do not allow self-loops.

The framework for testing community detection algorithms, which we will use, can now be summarised. Firstly, we generate synthetic datasets, by creating random graphs from the models described in sections 2.2.1 and 2.2.2, with varying parameters and known underlying ground-truth node assignments. Then we use the synthetically-generated graphs as an input to various algorithms (an appropriate model is chosen for each algorithm), which provides, as output, an estimate to the community assignments. Finally, for each

algorithm, we compare the estimated community assignments to the ground-truth values. This provides a notion of performance and accuracy to compare between the algorithms.

We now describe two models; one is a special case of the stochastic block model, created by imposing specific properties on the parameters, and the other a slightly modified version.

We also stress to the reader that we are only considering *non-overlapping communities*, where each node may only belong to one particular community.

2.2.1 Planted Partition Model

We will consider the formulation of the planted partition model as given by Decellle et al. [42]. To construct the planted partition model, we consider the stochastic block model, with $n_a = 1/k$, and the affinity matrix defined by

$$P_{ab} = \begin{cases} p_{in} & \text{if } a = b \\ p_{out} & \text{otherwise} \end{cases} \tag{2.2}$$

These properties essentially result approximately equal number of nodes and edges within each community. From this definition, p_{in} represents the probability of an edge occurring between two nodes belonging to the same community and p_{out} represents the probability of an edge occurring between two nodes belonging to the different communities. We assume assortative structure so that $p_{in} > p_{out}$. This just matches the intuition of edges more likely to appear between nodes belonging to the same community than between nodes belonging to different communities.

An example of a random graph generated by the planted partition model is shown in figure 2.2a. We labelled nodes using $\sigma_i = 1 + (i \bmod k)$ for $i = 1, \ldots, n$ and generated the graph with $n = 300$, $k = 3$, $p_{in} = 0.7$, $p_{out} = 0.3$. The adjacency matrix of this graph is plotted with a pixel shaded red if the element in the adjacency matrix, corresponding to the location of the pixel, equals 1; while a pixel is shaded white if the element equals 0. Since we know the ground truth labelling of nodes, we can, without loss of generality, reorder the rows and columns of the adjacency matrix, such that it consists of blocks of nodes associated with the node community memberships. This is plotted in figure 2.2b. Note that since $k = 3$, there are $3 \times 3 = 9$ blocks, where the blocks are denser along the main diagonal since these correspond to edges between nodes belonging to the same community and $p_{in} > p_{out}$.

(A) (B)

FIGURE 2.2: A set of plots for unlabelled, (a), and labelled, (b), adjacency matrices for random graph generated by planted partition model. The graphs were generated with $n = 300$, $k = 3$, $p_{in} = 0.7$ and $p_{out} = 0.3$.

FIGURE 2.3: A visualisation of an instance of a random graph generated by the planted partition model with $n = 240$, $k = 3$, $p_{in} = 0.2$ and $p_{out} = 0.01$.

In figure 2.3, we show a visualisation of an instance of a random graph generated by the planted partition model with parameters with $n = 240$, $k = 3$, $p_{in} = 0.2$, $p_{out} = 0.01$.

Decelle et al. [29] conjectured a phase transition for sparse graphs generated from the planted partition model, using non-rigorous ideas from statistical physics [37]. Nadakuditi et al. [35] used methods from random matrix theory to present an asymptotic analysis of spectra of random graphs to also demonstrate the presence of a phase transition. Essentially, we can distinguish between a *detectable* phase where it is possible to learn node assignments in a way that is correlated with the ground-truth node assignments of the graph, and an *undetectable* phase, where learning is impossible.

Let us define, for convenience, the variables $c_{in} = np_{in}$ and $c_{out} = np_{out}$. Consider a graph, generated by the planted partition model, and following the argument of [35], which we will not explain, one finds a transition occurring at the point

$$c_{in} - c_{out} = \sqrt{k(c_{in} + (k-1)c_{out})}. \tag{2.3}$$

In particular, let us consider the case where $k = 2$, so we find a transition at

$$c_{in} - c_{out} = \sqrt{2(c_{in} + c_{out})}. \tag{2.4}$$

Mossel et al. [37] proved the undetectable phase region of the conjecture given by equation equation (2.4) (that is to say, it is impossible to meaningfully recover the node assignments when $c_{in} - c_{out} < \sqrt{2(c_{in} + c_{out})}$). Massoulié [43] and then, independently using a different proof, Mossel et al. [44] proved the detectable phase region of the conjecture, meaning it is possible to recover node assignments positively correlated with the ground-truth when $c_{in} - c_{out} > \sqrt{2(c_{in} + c_{out})}$. The techniques used to prove these results are beyond the scope of this report, however these results provide a very important limit on the ability of algorithms to detect communities. This motivates the development of algorithms which can efficiently (in nearly linear time) detect communities, in the sparse regime, up to the limit.

2.2.2 Hidden Clique Model

We will consider the following formulation as explained by Montanari [46,48]. To construct the hidden clique model, we consider the stochastic block model with some modifications. We proceed to generate a graph with n nodes and k communities but with the affinity matrix, $\mathbf{P}$, becoming a $(k+1) \times (k+1)$ matrix defined by

$$P_{ab} = \begin{cases} p_{in} & \text{if } a = b,\ a \leq k,\ b \leq k \\ p_{out} & \text{otherwise} \end{cases} \tag{2.5}$$

Another tweak is that we now consider the variable n_a to represent the number of nodes within community a (rather than the expected fraction of nodes). Note that the number of nodes within each community does not necessarily sum to n, since we consider them to be 'hidden' within the graph. Also, we are interested in the regime where the size of these communities is small relative to the size of the graph. Once more, we assume assortative structure within the hidden communities so that $p_{in} > p_{out}$.

An example of a random graph generated by the hidden clique model is shown in figure 2.4a. We generated the graph with $n = 300$, $k = 3$, $p_{in} = 0.8$, $p_{out} = 0.2$, $n_1 = 50$, $n_2 = 40$, $n_3 = 20$. The adjacency matrix of this graph is plotted with a pixel shaded red if the element in the adjacency matrix, corresponding to the location of the pixel, equals 1; while a pixel is shaded white if the element equals 0. Since we know the ground truth

(A) (B)

FIGURE 2.4: A set of plots for unlabelled, (a), and labelled, (b), adjacency matrices for random graph generated by hidden clique model. The graphs were generated with $n = 300$, $k = 3$, $p_{in} = 0.8$, $p_{out} = 0.2$, $n_1 = 50$, $n_2 = 40$, $n_3 = 20$.

FIGURE 2.5: A visualisation of an instance of a random graph generated by the hidden clique model with $n = 150$, $k = 1$, $n_1 = 30$, $p_{in} = 1.0$, $p_{out} = 0.1$. Nodes belonging to the hidden community, which in this case is in fact a clique, are coloured red whilst other nodes are coloured blue.

labelling of nodes, we can, without loss of generality, reorder the rows and columns of the adjacency matrix, such that it consists of blocks of nodes associated with the node community memberships. This is plotted in figure 2.4b. We can see three dense blocks of size 50, 40 and 20 respectively, corresponding to the three hidden communities.

In figure 2.5, we show a visualisation of an instance of a random graph generated by the hidden clique model with parameters with $n = 150$, $k = 1$, $n_1 = 30$, $p_{in} = 0.3$, $p_{out} = 0.05$.

An interesting phase transition can also be derived for these models also. We follow the argument of Montanari [46, 48] to show this. We consider the simplest case of the model with only one hidden community (i.e. $k = 1$). We begin by generating a graph from the hidden clique model with n nodes and one community. Assume the n_1 nodes belonging to the hidden community make up a hidden community set, $\mathcal{S}$ (so that $|\mathcal{S}| = n_1$). Define $\mathbb{1}_n \in \{1\}^n$ as the n-dimensional vector with every element equal to 1. Also let $\mathbb{1}_{\mathcal{S}}$ be the indicator variable for nodes belonging to the hidden community. Denote the adjacency

matrix of the graph by $\mathbf{A}$, where each element is defined by

$$A_{ij} \sim Be(p_{ij}) \tag{2.6}$$

where

$$p_{ij} = \begin{cases} p_{in} & \text{if } i \in \mathcal{S},\ j \in \mathcal{S} \\ p_{out} & \text{otherwise} \end{cases} \tag{2.7}$$

Then, simply, we get the following

$$\mathbb{E}(\mathbf{A}) = (p_{in} - p_{out})\mathbb{1}_{\mathcal{S}}\mathbb{1}_{\mathcal{S}}^T + p_{out}\mathbb{1}_n\mathbb{1}_n^T \tag{2.8}$$

and

$$Var(A_{ij}) = p_{out}(1 - p_{out}) \text{ if } \{i, j\} \not\subseteq \mathcal{S} \tag{2.9}$$

Denote $\widetilde{\mathbf{A}}$ as the *normalised adjacency matrix* of the graph, defined by

$$\widetilde{\mathbf{A}} \equiv \frac{1}{\sqrt{np_{out}(1 - p_{out})}}(\mathbf{A} - p_{out}\mathbb{1}_n\mathbb{1}_n^T) \tag{2.10}$$

By taking the expectation and using equation (2.8), we get the following

$$\mathbb{E}(\widetilde{\mathbf{A}}) = \frac{1}{\sqrt{np_{out}(1 - p_{out})}}(p_{in} - p_{out})\mathbb{1}_{\mathcal{S}}\mathbb{1}_{\mathcal{S}}^T \tag{2.11}$$

Let us write $\widetilde{\mathbf{A}} = \mathbb{E}(\widetilde{\mathbf{A}}) + (\widetilde{\mathbf{A}} - \mathbb{E}(\widetilde{\mathbf{A}}))$. Now using equation (2.11), we get

$$\widetilde{\mathbf{A}} = \frac{1}{\sqrt{np_{out}(1 - p_{out})}}(p_{in} - p_{out})\mathbb{1}_{\mathcal{S}}\mathbb{1}_{\mathcal{S}}^T + (\widetilde{\mathbf{A}} - \mathbb{E}(\widetilde{\mathbf{A}})) \tag{2.12}$$

Let us now define the following

$$\lambda \equiv \frac{p_{in} - p_{out}}{\sqrt{np_{out}(1 - p_{out})}} \tag{2.13}$$

$$\boldsymbol{u} \equiv \mathbb{1}_{\mathcal{S}} \tag{2.14}$$

$$\mathbf{Z} \equiv \widetilde{\mathbf{A}} - \mathbb{E}(\widetilde{\mathbf{A}}) \tag{2.15}$$

Then we can re-write equation (2.12) as

$$\widetilde{\mathbf{A}} = \lambda \boldsymbol{u}\boldsymbol{u}^T + \mathbf{Z} \tag{2.16}$$

One can interpret λ as a signal-to-noise ratio, $\boldsymbol{u}$ as a signal (i.e. the ground-truth node

(A) (B)

FIGURE 2.6: We plot the limiting spectral density of the normalised adjacency matrix under two regimes. We illustrate the case where $\lambda < 1$, in (a) and the case where $\lambda > 1$, in (b). The blue dot in (b) represents the eigenvalue $(\lambda + \lambda^{-1})$ associated with he eigenvector that pops-out of the main semicircle lobe. Both figures obtained from [48].

assignments we which to infer) and $\mathbf{Z}$ as zero-mean noise with i.i.d. entries. We have essentially represented the problem of inferring the hidden community from the graph by a problem of estimating a rank-1 matrix in noise. Notice that for the generalisation with k communities, we would get a rank-k matrix plus noise for the normalised adjacency matrix.

Assume we generate a network associated with a normalised adjacency matrix, $\widetilde{\mathbf{A}}$, defined in equation (2.16) and are interested in reconstructing the vector of node assignments, $\boldsymbol{u}$. This problem has been investigated in many application under the guise of 'Low-rank deformation of Wigner matrices' [48]. Moreover much is known about the eigenvalue spectrum of such matrices. There is a very important spectral phase transition that exists [48]; if $\lambda < 1$, the top eigenvector of the adjacency matrix, $\mathbf{A}$, is orthogonal to the vector we wish to reconstruct (i.e. $\langle \boldsymbol{v}_1(\mathbf{A}), \boldsymbol{u} \rangle \approx 0$), whereas, if $\lambda > 1$, the top eigenvector of $\mathbf{A}$ is correlated with the vector we wish to reconstruct and $\langle \boldsymbol{v}_1(\mathbf{A}), \boldsymbol{u} \rangle \approx (1 - \lambda^{-2})$. In the latter regime, one eigenvector pops out of the semicircle lobe, as illustrated in figure 2.6. This particular eigenvector is associated with the eigenvalue $\lambda + \lambda^{-1}$ [48].

This result is key since it specifically describes a threshold where traditional spectral methods such as standard Principal Component Analysis (PCA) will not work (i.e. when $\lambda < 1$) and when it will produce a reconstructed vector correlated with the ground-truth (i.e. when $\lambda > 1$). Moreover we now have sufficient motivation to investigate methods where we can do better; more specifically we wish to study algorithms where we can essentially 'beat' this spectral threshold by producing a reconstructed vector correlated with the ground-truth in the regime where $\lambda < 1$. There is hope of achieving the improvement over standard PCA since we observe the structure of the principal eigenvector of the matrix has two special properties, it is *non-negative* (since the elements are node

assignments or indicator variables and are thus either zero or one) and *sparse* (since we are interested in the regime where the size of the hidden community or clique is small relative to the size of the graph).

2.3 Financial Networks

2.3.1 Prices and Returns of Financial Assets

Financial assets are instruments claiming to have monetary value that can be bought and sold. Financial assets can be separated into broad classes, with examples including stocks, bonds or real estate [32, 49]. The values of these assets is reflected in their price, which varies with time. Investors may wish to decide between which asset classes to invest in at any time, a process known as *asset allocation* [49]. Also, within a particular asset class, an investor wishes to allocate money to specific assets, a process known as *portfolio selection* [49]. For this report, we will focus on portfolio selection of stocks in our application of community detection algorithms, due to available data constraints; however a very similar scheme may be used to tackle asset allocation also.

Investors tend not to consider the prices of assets they have invested in, but rather the *return* generated. Let us consider a financial asset whose price at time t is $p(t)$. One popular measure of return is called the *rate of return* [33, 49] at a time t, denoted by $r(t)$, which is defined as

$$r(t_0) = \frac{p(t_1) - p(t_0)}{p(t_0)} \tag{2.17}$$

where we can interpret t_1 as the time when the investor sold the asset, and t_0 as the time when the investor bought the asset.

Critically, the rate of return is sensitive to large changes for longer time horizons [6]. In particular, we can consider a different measure of return, which is equivalent to a return with a constant interest rate [6]. We can generalise the concept interest rates with the simple example of an investor placing money (investing) in a bank account, as explained in [3, 31]. The amount of money initially invested is referred to as the *principal*. We then assume money grows by a multiplicative factor, where the gain is paid into the account by the bank. This process is often called *compounding*. The time at which the interest is compounded, is called the compounding period. *Compound interest* involves interest being paid on both the principal and the accumulated interest up to the present [6]. Typically, we are interested in the number of compounding periods in one year (i.e. the

number of times the interest on our principal is compounded each year) [31]. Denote the principal by w_0, the amount in the account time t by w_t, the interest rate by y and the number of compounding periods in a year by m. Then the amount within the account holdings after 1 year is given by

$$w_1 = w_0(1 + (y/m))^m \tag{2.18}$$

We can imagine diving a year into infinitesimally small compounding periods, and then determine the effect of this continuous compounding by taking the limit of ordinary compounding [3]. Notice the total number of compounding periods in a length of t years is given by mt. Thus the effect of continuous compounding is

$$w_t = \lim_{m\to\infty} w_0(1 + (y/m))^{mt} = w_0 \exp(yt) \tag{2.19}$$

If we divide equation (2.19) by the initial investment w_0, and take the natural logarithm, we get a representation for the return, $r = yt$. This indicates that taking the natural logarithm results in a constant interest rate. We have arrived at another measure for return, called the *logarithmic return*, that is defined as

$$r(t_0) = \log(p(t_1)) - \log(p(t_0)) \tag{2.20}$$

where, once more, we can interpret t_1 as the time when the investor sold the asset, and t_0 as the time when the investor bought the asset.

There are several advantages to using logarithmic returns, as explained in [52], which we will briefly summarise. Firstly, if we assume asset prices have a *log normal* distribution, then the logarithmic returns are conveniently normally distributed [52]. The reasons why assuming a log normal distribution may be appropriate for dynamic pricing of assets is beyond the scope of this report. Secondly, for small rates of return, the logarithmic return is approximately equal to the rate of return [52]. To see this, consider the following approximation combined with equation (2.17) and equation (2.20)

$$\log(1 + x) \approx x, \text{ for } x \ll 1 \tag{2.21}$$

Thirdly, we benefit from numerical stability since the addition of small numbers is numerically stable, whilst multiplying small numbers is subject to *arithmetic underflow* [52].

FIGURE 2.7: Plots of prices and logarithmic returns for Anglo American plc (AAL) between 2004 and 2014

However, there are disadvantages to using the logarithmic return, including the issue that the derivation is only correct if the interest rate is constant [6, 52]. Nonetheless, the logarithmic return is widely used in the literature (e.g. see [6–8, 25, 30, 45]), and hence we shall use it for the rest of the report, and the reader should note we shall use the terms 'return' and 'logarithmic return' interchangeably from now on. An example plot of price and logarithmic return for a stock is shown in figure 2.7.

2.3.2 Mean-Variance Portfolio Theory

The term *portfolio* relates to a investing in a combination of different assets. We can characterise a portfolio by *portfolio weights*, where the weight of an asset within the portfolio is given by the ratio of the value of the position in the asset divided by the total value of the portfolio. We are particularly interested in the return of the portfolio, which is related to the mean of the returns of the individual assets that make up the portfolio, as we will see, and the risk of the portfolio, which is related to the variance of the returns of the individual assets. This is the source of the term mean-variance portfolio theory. The reader should realise the inherit trade-off between risk and return; if the investor wishes to realise a larger return, he must bear a higher risk. A more detailed explanation of this relationship is given in [3, 33]. Rather we are simply interested in finding the most efficient, or *minimum-variance portfolio* for a given requested portfolio return (that is to say, the investor requests a specific expected return, and wishes to form a portfolio that has the lowest variance of all possible portfolio that can deliver the specified expected return). We can formalise the mean-variance portfolio setting in the following way, which summarises the explanations from [3, 6, 33].

Let P be a portfolio comprising of n assets, where the return of the portfolio is denoted by r_P and the variance of the portfolio is denoted by σ_P^2. Denote the return of asset i by r_i, the variance of the asset's return by $\sigma_i^2 \equiv Var(r_i)$, and the weight of asset i in the

portfolio by w_i. Also let X_0 denote the total amount invested in the portfolio (i.e. initial investment in the portfolio by the investor) and X_{0i} represent the the amount invested in asset i. We select the amounts in the assets forming the portfolio such that

$$\sum_{i=1}^{n} X_{0i} = X_0 \tag{2.22}$$

We define the portfolio weights using

$$w_i = \frac{X_{0i}}{X_0},\ i = 1, \ldots, n \tag{2.23}$$

and clearly

$$\sum_{i=1}^{n} w_i = 1 \tag{2.24}$$

Notice that a negative weight indicates a *short position* in that asset, and that the returns of the individual assets and portfolio are random variables.

In particular we can represent the return of the portfolio by

$$r_P = \sum_{i=1}^{n} w_i r_i \tag{2.25}$$

By using equation (2.25), we obtain the expected return of the portfolio

$$\mathbb{E}(r_P) = \sum_{i=1}^{n} w_i \mathbb{E}(r_i) \tag{2.26}$$

and the variance of the portfolio return

$$Var(r_P) \equiv \sigma_P^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \rho_{ij} \sigma_i \sigma_j \tag{2.27}$$

where ρ_{ij} is the correlation coefficient of the returns of assets i and j, defined as

$$\rho_{ij} = \frac{Cov(r_i, r_j)}{\sigma_i \sigma_j} \tag{2.28}$$

Note that the standard deviation of the returns of the portfolio (or any asset) is often called its *volatility*.

This formulation serves a key question, given estimates of each assets returns, variances and covariances (which one can obtain from historical data), how does one pick a selection

of these assets, for any given time period, in order to form the best portfolio for the investor? We see in equation (2.27), that by simply investing in assets which have a lower correlation with one another (i.e. a lower value of ρ_{ij}), we can reduce the variance, and thus risk, of the portfolio. This process is called *diversification*. Thus, for any given time period (and possibly dynamically), finding groups of assets, where the returns have higher correlation within groups and lower correlation between groups would help by presenting 'baskets' of assets that the investor can pick from knowing selecting from a range of baskets would be beneficial (of course which assets to select from inside the basket relates to the risk-return trade off). This serves as the main motivation for the application of community detection algorithms within financial networks, as we formalise in section 2.3.3.

2.3.3 Constructing Financial Networks

From section 2.3.2 we understand one way to help minimise risk in constructing portfolios involves analysing the correlation coefficients of returns between two assets. In order to study all possible correlations between all available assets, we construct a weighted, undirected and fully-connected network of assets, which we call the *financial network*. The following model has been considered by [5, 7, 8, 25, 45].

Let us consider the situation where the investor is faced with n financial assets, and has access to historical price data for all these assets for T time steps. The time steps may be trading days, or weeks, for instance, and the appropriate choice will depend on the type of assets available.

We proceed to construct a graph with n nodes, where each nodes represents an asset, and assign to the i-th node a single time series, denoted by X_i, which is defined as

$$X_i = \{x_i(1), \ldots, x_i(T)\} \tag{2.29}$$

where $x_i(t)$ describes the logarithmic return of asset i at time t, defined by equation (2.20). This time series describes the evolution of the logarithmic return of the asset over T time steps. We then model the weight of an edge connecting nodes i and j of the graph by the cross-correlation between the time series corresponding to assets i and j. We form a cross-correlation matrix, denoted by $\mathbf{C}$, whose elements are defined by

$$C_{ij} = \frac{\langle X_i X_j \rangle - \langle X_i \rangle \langle X_j \rangle}{\sqrt{[\langle X_i^2 \rangle - \langle X_i \rangle^2][\langle X_j^2 \rangle - \langle X_j \rangle^2]}} \tag{2.30}$$

FIGURE 2.8: Example of a correlation matrix. Evaluated from an ensemble of 80 stocks listed on the FTSE 100 (see appendix A for a list) using data between 01/01/2011 and 01/01/2013.

where the $\langle \cdots \rangle$ notation denotes a time average, so that

$$\langle X_i \rangle = \frac{1}{T} \sum_{t=1}^{T} x_i(t) \tag{2.31}$$

$$\langle X_i^2 \rangle = \frac{1}{T} \sum_{t=1}^{T} x_i^2(t) \tag{2.32}$$

$$\langle X_i X_j \rangle = \frac{1}{T} \sum_{t=1}^{T} x_i(t) x_j(t) \tag{2.33}$$

We also assume each time series X_i has been standardised (before assigning to the node) by using

$$X_i \coloneqq \frac{X_i - \langle X_i \rangle}{\sqrt{\langle X_i^2 \rangle - \langle X_i \rangle^2}} \tag{2.34}$$

so that

$$\langle X_i \rangle = 0 \tag{2.35}$$

$$\langle X_i^2 \rangle - \langle X_i \rangle^2 = 1 \tag{2.36}$$

Note that the cross correlation values is just a sample estimate for the correlation coefficient, ρ_{ij}, used in section 2.3.3, calculated from the historical data.

We can then characterise the financial network by the correlation matrix, which we also refer to as the networks weighted adjacency matrix.

In figure 2.8, we have plotted a correlation matrix using data of 80 stocks listed on the FTSE 100 (see appendix A for a list) between 2011 and 2013. Notice that the main diagonal has all elements equal to one, as you would expect, and that there are very few negative elements (i.e. very few assets that are anti-correlated with one another).

The problem statement can now summarised. Given a financial network, how can we group nodes into communities where correlations are higher within the communities and lower between the communities? Contrary to graphs with community structure described in section 2.2, the weights of the edges rather than the topology of the network are crucial in determining community memberships. In other words, we focus solely on the weighted adjacency matrix of the graph. Also the reader should note that the correlations between asset returns will vary over time, and thus representing this relationship dynamically (rather than over a one long period of time) is very important since investors wish to change their positions in order to react to the dynamics of market conditions.

Chapter 3

Community Detection Algorithms

In this chapter we introduce several community detection algorithms present in the literature that can be used to detect communities based upon different approaches. We introduce spectral clustering, modularity-based optimisation, non-linear power iteration method and message-passing algorithms, with specific reference to the application on generative block models.

3.1 Spectral Clustering

The basis of all spectral clustering algorithms is the transformation of a set of variables into the set of points in space whose coordinates are elements of eigenvectors of a matrix, and then the clustering of these points using well-known clustering algorithms [16, 22]. We consider the spectral clustering algorithm described in [16, 22], but we do not aim to justify the intuition behind the algorithm since it requires results spectral graph theory that is beyond the scope of this report. Nevertheless, the reader may refer to [16, 17, 22] for the details.

Firstly we compute the Laplacian matrix of the network, using definition 2.13 where we assume n nodes in the graph. Then we compute the k eigenvectors of the Laplacian matrix associated with the k largest eigenvalues. Denote the eigenvectors by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ and the eigenvalues by $\lambda_1, \ldots, \lambda_k$. The eigenvectors represent a k-tuple of real numbers associated with each vertex in the graph. We think of this association as a mapping from the vertices into a k-dimensional space. This embedding is characterised by $F : V \to \mathbb{R}^k$ where $F(i) = (\boldsymbol{u}_1^{(i)}, \ldots, \boldsymbol{u}_k^{(i)})$ and $\boldsymbol{u}_j^{(i)}$ denotes the i-th element of the j-th eigenvector. We

FIGURE 3.1: A visualisation of the embedding for the spectral clustering algorithm. The ground truth graph was generated using the planted partition model with $n = 150$, $k = 3$, $p_{in} = 0.8$ and $p_{out} = 0.2$. We have chosen 3 distinct communities since we can easily display the embedding in a 2-dimensional space. The coordinates for each point are the corresponding entries in the 2 eigenvectors of the Laplacian matrix considered. We label the ground-truth node assignments by colour (i.e. points with the same colour represent nodes belonging to the same community in the graph), and we can see that a k-means clustering algorithm would be applied to return cluster memberships that match the true community memberships.

have essentially represented node i of the graph as a point in a k-dimensional space where the coordinates are the i-th elements of all the top k eigenvectors of the Laplacian matrix. Finally, we apply the embedding as input to the popular k-means clustering algorithm. The clusters memberships of the n data points are the node assignments for the initial network. Notice, from the definition of the Laplacian matrix, that the vector of all ones is the principal eigenvector, meaning the elements of this remains the same for all nodes. Therefore, knowledge of this eigenvector does not help discriminate between different vertices, and hence the information is not useful. Hence we only apply the embedding to $k-1$ eigenvectors (i.e. the top k eigenvectors excluding the all-ones vector), and can thus represent the mapping in a $k-1$ dimensional space.

To visualise what such an embedding looks like, refer to figure 3.1, where we generated an example graph using the planted partition model and 3 communities (this is selected so the reader can easily see the embedding in a 2-dimensional space). The coordinates for each point are the corresponding entries in the 2 eigenvectors of the Laplacian matrix considered. We labelled the ground-truth node assignments by colour in the figure (i.e. points with the same colour represent nodes belonging to the same community in the graph), and we can see that a k-means clustering algorithm would be applied to return cluster memberships that match the ground-truth.

Note that the algorithm does require to compute k eigenvectors of a matrix, and this can be achieved using the power method. Also, the accuracy of the algorithm will largely depend on the k-means algorithm which has been shown to converge to local minima in

a cost measure (rather than global), but despite this, it seems to work well in practical applications [16, 22].

3.2 Modularity-based Optimisation

We introduce these algorithms, first considered by [14, 15], by explaining the intuition behind 'good' community partitions. In particular, the key ingredient involves determining partitions of the network where there are fewer edges between nodes belonging to different communities than expected. For instance, if the number of links between nodes associated between different communities is approximately the same as what one would expect to find given random placement of links within the network, then it is unlikely this provides a notion of meaningful community structure [15]. Equivalently, we can consider partitions where there are more edges between nodes belonging to the same community than expected.

Definition 3.1. The *null model* with respect to a network, whose adjacency matrix is given by $\mathbf{A}$, is the random graph denoted by $\mathcal{G}$, where each edge has a probability of $\frac{d_i d_j}{2m}$ of occurring, d_i is the degree of node i and $2m \equiv \sum_{ij} A_{ij}$.

The null model defined above is proposed as a baseline distribution if edges were randomly placed within the network.

Definition 3.2. Given a partition, $\boldsymbol{\sigma}$, of a network, the *modularity* is defined as: $Q(\boldsymbol{\sigma}) = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \delta(\sigma_i, \sigma_j)$.

The modularity is therefore considered a cost function for a partition of the network where larger modularity values indicate stronger community structure [14].

Definition 3.3. The *modularity matrix* is denoted by $\mathbf{B}$, whose elements B_{ij} are defined by $B_{ij} = A_{ij} - \frac{d_i d_j}{2m}$.

The aim of modularity optimisation algorithms is to find a partition of the network with the maximum value of modularity associated. Since searching over all possible partitions is exponential in the number of nodes of the network, the problem is NP-hard computationally [15]. Thus, we seek approximate methods that provide near-optimal solutions.

In the literature, there exists a variety of approximation algorithms for accurate and fast modularity optimisation, such as greedy algorithms, simulated annealing, spectral algorithms and extremal optimisation [22]. Within this report, we shall describe all these

FIGURE 3.2: Illustration of the two phases of the greedy method of [19]. The first phase involves optimisation of modularity using local changes, and the second phase aggregates the nodes into communities. The two phases are repeated until no further improvement of modularity is possible. For this example, only two passes are required until termination. This figure is reprinted from the Blondel et al. reference [19].

algorithms but only implement and test the greedy agglomerative method, since we consider it to be a faster version whilst maintaining similar accuracy to others.

3.2.1 Greedy Algorithm

The greedy algorithm of Clauset et al. [10] starts with all nodes as single groups and successively mergers two groups to form a larger community such that the modularity of the new partition increases after the joining [10, 22]. Moreover, the algorithm keeps permanently, the merger with the largest increase in modularity (hence at each step we compute ΔQ, the change in modularity, using definition 3.2). This is iterated until no further increase in modularity is possible [10]. Note that for a network with n nodes and m edges, the algorithm has complexity $O((m+n)n)$, or $O(n^2)$ on a sparse graph [22].

A different greedy algorithm has been proposed by Blondel et al. [19], that is also applicable for weighted networks [22]. We initialise each node to belong to an individual community, and then repeat the following two phases until there is no further increase in modularity possible. In the first phase, we sequentially consider each node, and given node i, we compute the increase in modularity, ΔQ, that results from moving node i into a neighbour community, and then permanently select the transition that yields the greatest increase in modularity [19, 22]. In the second phase, two communities are connected if an edge exists between any node belonging to the communities [19, 22]. Figure 3.2 illustrates the two phases of the algorithm on an example network. This figure is taken from the Blondel et al. reference [19].

3.2.2 Simulated Annealing

The simulated annealing algorithm of [2] is an iterative procedure that explores a space of possible states looking for the global optimum of modularity, which we denote by Q [2, 22]. Updates from one state to another are accepted with probability 1 if the transition results in an increase in the modularity. Otherwise, the update is only accepted with a small probability $\exp(-\beta \Delta Q)$, where ΔQ is the change in modularity (i.e. value of modularity after the transition minus the value before) and β represents the inverse-temperature of the system [2, 22]. The idea behind accepting a transition that results in a decrease in modularity with a small probability is to increase the chance of finding the global maximum (i.e. decrease the chance of converging towards local maxima) [22]. The algorithm converges to a stable state at some point, depending on the number of states explored and how β is varied, but it can be a good approximation for Q.

A more recent implementation by [11] consists of iterations that involve both individual and collective steps. Within the individual step, an individual node is moved to community at random, whilst the collective step consists of merging two communities or splitting one community [11]. Typically, each iteration involves n^2 individual steps and n collective steps, where n represents the number of nodes in the network [22]. This method can approximate the true maximum of modularity very accurately, and note that due to the variation of parameter selection (such as initial temperature and inverse-temperature chosen), an exact complexity cannot be estimated, but it is typically very slow and can only be used for small graphs [22].

3.2.3 Extremal Optimisation

The extremal optimisation algorithm of Duch and Arenas [13] is a heuristic search method that involves recursively bi-partitioning the network [13, 22]. It begins with a random partition and uses the contribution of each node to the modularity as a fitness measure with the movement of nodes with the lowest fitness value. The fitness function value of a node i is given by

$$q_i = \kappa_{\sigma(i)} - d_i e_{\sigma(i)} \tag{3.1}$$

where d_i is the degree of node i, $\kappa_{\sigma(i)}$ is the number of neighbours node i has in the community it belongs to, and $e_{\sigma(i)}$ is the fraction of edges in the network that connects at least one node which belongs to node i's community. Using this notation, one can re-write the modularity by $Q = \frac{1}{2m} \sum_i q_i$. We also normalise the variables q_i by dividing by d_i to

obtain

$$\rho_i = \frac{\kappa_{\sigma(i)}}{d_i} - e_{\sigma(i)} \tag{3.2}$$

so that $-1 \leq \rho_i \leq 1$ for all i. Therefore we have expressed the global cost function in terms of a sum over all vertices (through the local variables ρ_i), and, hence, we can optimise the global variable, Q, by optimising over the local variables [22]. At each iteration of the algorithm, we calculate ρ_i for every node i and move the node with the lowest value moved to the other community. Note that this transition alters the overall partition so that the values of the fitness need to be re-evaluated [13,22]. We repeat this process until no further improvement in the modularity is possible [13,22]. Extremal optimisation has empirically shown to achieve similar accuracy to simulated annealing but with a fast run time.

3.2.4 Spectral Algorithm

We shall describe the spectral method of [14] using the derivation explained in [14,15,22]. Recall the notation used for modularity and the modularity matrix in definitions 3.2 and 3.3, and let $\boldsymbol{\sigma}$ represent the vector of node assignments, where $\sigma_i = 1$ if node i belongs to class 1 and $\sigma_i = -1$ if it belongs to class 2. Then the modularity can be written as

$$\begin{aligned} Q &= \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{d_i d_j}{2m}\right)\delta(\sigma_i, \sigma_j) \\ &= \frac{1}{4m}\sum_{i,j}\left(A_{ij} - \frac{d_i d_j}{2m}\right)(\sigma_i\sigma_j + 1) \\ &= \frac{1}{4m}\sum_{i,j} B_{ij}\sigma_i\sigma_j \\ &= \frac{1}{4m}\boldsymbol{\sigma}^T\mathbf{B}\boldsymbol{\sigma} \end{aligned} \tag{3.3}$$

We can rewrite $\boldsymbol{\sigma}$ as a linear combination of the eigenvectors of $\mathbf{B}$, which we denote by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ (where we label in decreasing order corresponding to the associated eigenvalue), so that

$$\boldsymbol{\sigma} = \sum_{i=1}^{n}(\boldsymbol{u}_i^T\boldsymbol{\sigma})\boldsymbol{u}_i \tag{3.4}$$

Using this result in equation (3.3) yields

$$Q = \sum_i(\boldsymbol{u}_i^T\boldsymbol{\sigma})\boldsymbol{u}_i^T\mathbf{B}\sum_j(\boldsymbol{u}_j^T\boldsymbol{\sigma})\boldsymbol{u}_j = \sum_{i=1}^{n}(\boldsymbol{u}_i^T\boldsymbol{\sigma})^2\lambda_i \tag{3.5}$$

where λ_i is the eigenvalue of $\mathbf{B}$ associated with the eigenvector $\boldsymbol{u}_i$. We aim to maximise the modularity by choosing the elements of $\boldsymbol{\sigma}$. From equation (3.5), we see this can be achieved by aiming to increase the weights of the largest (i.e. most positive) eigenvalues. We cannot just set $\boldsymbol{\sigma}$ to be proportional to the largest eigenvector, $\boldsymbol{u}_1$, as we imposed each element to be either +1 or -1. Instead, we seek an approximate method, where we proceed to set the values of σ_i based upon the sign of the i-th component of $\boldsymbol{u}_1$. Essentially, the algorithm involves computing the leading eigenvector of the modularity matrix and then partitioning the nodes of the network into two groups according to the signs of the corresponding elements in the eigenvector. We repeat this procedure for dividing any one community into two communities until no further sub-division increases the value of modularity, when the algorithm shall terminate.

The spectral method for optimising modularity is quite fast, since computing the leading eigenvector of the modularity matrix can be computed using the well-known power method. Due to the special structure of the modularity matrix, the computation of the leading eigenvector takes $O(m + n)$ time, so that one partition of the network takes $O(n(m + n))$ time or $O(n^2)$ for a sparse graph [22]. As we need to repeatedly partition the network in order to optimise the modularity, the overall complexity is $O(dn(m + n))$ where d represents the depth of the hierarchical division. Typically, in practice $d \approx \log n$, so that for sparse graphs, the total time taken for this spectral algorithm is approximately $O(n^2 \log n)$. The spectral method is faster than simulated annealing and extremal optimisation, although not as fast empirically as the greedy algorithm [22]. An added benefit is the extensibility of the spectral method with the application to weighted networks.

3.3 Belief Propagation Algorithm

3.4 NLPI and AMP Algorithms

The following algorithms aim to partition networks based upon networks generated from the hidden clique model described in section 2.2.2.

3.4.1 Non-linear Power Iteration

The intuition behind the non-linear power iteration (NLPI) method is fairly straightforward. Recall we wish to reconstruct the node assignment vector denoted by $\boldsymbol{u}$. Equation (2.16) shows $\boldsymbol{u}$ is the principal eigenvector of a rank-1 matrix in noise, called the normalised adjacency matrix, and denoted by $\widetilde{\mathbf{A}}$. We use the standard power iteration algorithm with one extra step; we additionally apply a separable non-linear function that acts component-wise. We choose the non-linear function to force the reconstructed vector to adhere to one of the properties desired. In particular we can apply 'positive-part thresholding' [48], where we keep only the positive elements of the vector (and set the negative elements to zero) at each iteration. The following is a recursive definition of one iteration of the general approach, where t indexes the iteration

$$\begin{aligned} \boldsymbol{z}^{t+1} &= \widetilde{\mathbf{A}}\widehat{\boldsymbol{u}}^{t} \\ \widehat{\boldsymbol{u}}^{t} &= f_t(\boldsymbol{z}^{t}) \end{aligned} \tag{3.6}$$

where

$$\begin{aligned} \widehat{\boldsymbol{u}}^{0} &= [1, \ldots, 1]^T \\ \boldsymbol{z} &= [z_1, \ldots, z_n]^T \\ f_t(\boldsymbol{z}) &= [f_t(z_1), \ldots, f_t(z_n)]^T \end{aligned} \tag{3.7}$$

since we consider positive-part thresholding, for all $i = 1, \ldots, n$

$$f_t(z_i) = \begin{cases} z_i & \text{if } z_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

We will test this algorithm empirically for synthetically generated networks for varying SNR and sizes of the hidden community in section 4.3, but it is also important to analyse the algorithm theoretically, for instance by quantifying the (possible) improvement over spectral methods. Unfortunately, analysing this algorithm in terms of precise asymptotics is very difficult since there are dependencies existent after any number of iterations [48].

3.4.2 Approximate Message Passing

Although the NLPI method works well in practice (and we shall show this empirically later), we still seek an algorithm that can also be analysed theoretically. The approximate

message passing (AMP) algorithm involves one modification to the NLPI, where a memory term is subtracted. The following is a recursive definition of one iteration of the general approach, where t again indexes the iteration

$$\begin{aligned} \boldsymbol{z}^{t+1} &= \widetilde{\mathbf{A}}\widehat{\boldsymbol{u}}^{t} - b_t\widehat{\boldsymbol{u}}^{t-1} \\ \widehat{\boldsymbol{u}}^{t} &= f_t(\boldsymbol{z}^{t}) \end{aligned} \tag{3.9}$$

where we define

$$\begin{aligned} \widehat{\boldsymbol{u}}^{-1} &= [0,\ldots,0]^T \\ \widehat{\boldsymbol{u}}^{0} &= [1,\ldots,1]^T \\ \boldsymbol{z} &= [z_1,\ldots,z_n]^T \\ f_t(\boldsymbol{z}) &= [f_t(z_1),\ldots,f_t(z_n)]^T \\ b_t &\equiv \frac{1}{n}\sum_{i=1}^{n} f_t^{'}(z_i) \end{aligned} \tag{3.10}$$

and we again consider positive-part thresholding. We remark that the explicit formula for b_t is chosen since it cancels the statistical bias (i.e. decorrelates) on $\widehat{\boldsymbol{u}}_i^{t+1}$ due to $\widehat{\boldsymbol{u}}_i^{\leq t}$. The explanation for this result is beyond the scope of this report, but we refer the reader to [20, 21, 27, 28, 47] for details.

Chapter 4

Experiments on Synthetic Data

In this chapter we aim to experiment with community detection algorithms on synthetically generated data. We shall consider each algorithm described in the previous chapter, and experiment with data created from the appropriate generative model. The goal of the experiments on synthetic data is to understand how the underlying network structure, and the variation of parameters therein, affects the performance of different algorithms [39]. In general the experiments will proceed as follows. We generate a network with the appropriate block model and specified parameters, with an underlying ground-truth node assignments. We then measure the accuracy of the specific algorithm investigated as we vary model parameters. This then allows us to draw conclusions regarding the relative performance of community detection algorithms in controlled conditions given by networks with common properties. We conclude by comparing all the algorithms investigated and provide recommendations for their use in certain circumstances.

4.1 Spectral Clustering and Modularity-Optimisation Algorithms

We shall test the spectral clustering algorithm of section 3.1 and the greedy modularity optimisation method of section 3.2.1 using identical synthetic data generated using the planted partition model described in section 2.2.1. The goals are to, firstly, understand the quality of the partitions (i.e. accuracy) generated from the two algorithms as we vary both the sparsity of the network and the edge occurrence probabilities. Secondly, we wish to empirically test how close we can reliably detect communities up to the phase

FIGURE 4.1: A plot for the overlap of reconstructed node assignments of the Laplacian spectral clustering (blue) and greedy modularity optimisation methods (red) as the relative edge occurrence probabilities (c_{out}/c_{in}) are varied. 100 networks were generated with common parameters, $n = 200$, $k = 2$, $p_{in} = 0.9$. Both algorithms perform very similarly in this regime and we can clearly identify the detectable and undetectable phases since the there is a very sharp drop in the overlap values for both algorithms.

transition point. We assume assortative community structure throughout also. We apply each of generated networks as input to the Laplacian spectral clustering and greedy modularity method, and measure the accuracy of reconstruction of the node assignments by computing the *overlap* defined below [29]. Denote the overlap by T, and let the estimated node assignments given by $\{s_i\}$ with ground-truth node assignments given by $\{\sigma_i\}$, then we define T by

$$T = \max_{\pi} \frac{\frac{1}{n}\sum_i \delta(\sigma_i, \pi(s_i)) - \frac{1}{k}}{1 - \frac{1}{k}} \tag{4.1}$$

where π ranges over the permutations on s elements and k represents the number of clusters. This definition means that $0 \leq T \leq 1$ with a higher value implying improved reconstruction and more accurate results, and an overlap of 0 meaning the algorithm is, on average, no better than random uniform guessing of node assignments.

The traditional spectral clustering algorithm has been implemented in MATLAB based on the derivation in section 3.1, whilst the MATLAB implementation for the greedy modularity algorithm has been obtained from [53].

We begin by considering the dense regime and generating 100 networks from the planted partition model with common parameters $n = 200$, $k = 2$, $p_{in} = 0.9$. However, we vary p_{out} for each of the 100 networks to get a different value for the ratio c_{out}/c_{in}, so this signifies the variation in the edge occurrence probabilities. The results are shown in figure 4.1, where the overlap is plotted in the vertical axis and the horizontal axis is the value for the ratio c_{out}/c_{in}.

We see very similar performance for both algorithms in this regime and can also identify the phase transition region described in section 2.2.1, since there is a very sharp

FIGURE 4.2: A plot for the overlap of reconstructed node assignments of the Laplacian spectral clustering (blue) and greedy modularity optimisation methods (red) as the relative edge occurrence probabilities (c_{out}/c_{in}) are varied. 100 networks were generated with common parameters, $n = 200$, $k = 2$, $p_{in} = 0.6$. Both algorithms perform similarly for $c_{out}/c_{in} < 0.5$ but the modularity method seems to perform better until we get the sharp drop in overlap values due to the phase transition.

drop in overlap for a value of $c_{out}/c_{in} \approx 0.85$. Overlap values for both algorithms for $c_{out}/c_{in} < 0.85$ are approximately 1 implying perfect reconstruction of the ground-truth node assignments, whilst for $c_{out}/c_{in} > 0.85$, the overlap drops very sharply towards 0.

Let us now consider a sparser regime, where we again generate 100 networks from the planted partition model but now with common parameters $n = 200$, $k = 2$, $p_{in} = 0.6$. The results are shown in figure 4.2

Notice that both algorithms perform similarly for $c_{out}/c_{in} < 0.5$ but the modularity method seems to perform better until we get the sharp drop in overlap values. This is a very important observation to note, that the sharp drop in overlap values, for both algorithms, occurs at $c_{out}/c_{in} \approx 0.75$ which is lower than that for the case where $p_{in} = 0.9$. The theoretical phase transition point is the same for both cases, therefore this suggests that, as the network gets sparser, both algorithms performance declines with respect to performing perfect reconstruction within the detectable phase. These observations should not be a surprise given the derivation of the algorithms, and we will now explain known issues with both these algorithms that cause the decline in accuracy.

Recall the spectral clustering algorithm requires applying a k-means algorithm to the embedded vectors in a $k - 1$ dimensional plane. For the case with dense graphs, the eigenvalue spectrum of the Laplacian is separated and the embedded vectors can be seen to be arbitrarily well separated also. However, for the case where the graph is sparse, the eigenvector elements (i.e. coordinates in the plane) are very similar, and the k-means algorithm performance will be very poor, results in much lower accuracy for the spectral clustering algorithm. This observation is summarised as an illustration in figure 4.3.

(A) (B)

FIGURE 4.3: A visualisation of the embedding for the spectral clustering algorithm in the dense, (a), and sparse, (b), regimes. The ground truth graph for the dense regime was generated using the planted partition model with $n = 150$, $k = 3$, $p_{in} = 0.8$ and $p_{out} = 0.2$, whilst the graph for the sparse regime was generated using the same values of n and k but with $p_{in} = 0.08$ and $p_{out} = 0.02$. We label the ground-truth node assignments by colour (i.e. points with the same colour represent nodes belonging to the same community in the graph). Notice that, for the dense regime, all data points with different colours are well separated so the k-means algorithm can accurately recover the cluster memberships. However, for the sparse regime, every data (red green and blue) are concentrated in the same region in the 2-dimensional embedding space, so the k-means algorithm cannot recover the cluster memberships accurately at all. This illustrates an issue with the Laplacian spectral clustering method.

For the dense regime, we generate a network from the planted partition model with $n = 150$, $k = 3$, $p_{in} = 0.8$ and $p_{out} = 0.2$, whilst for the sparse regime, we generate a network with the same values for n and k, but $p_{in} = 0.08$ and $p_{out} = 0.02$. Note the important regulation that, for this illustration, the ratio c_{out}/c_{in} is identical for both regimes, and is below the theoretical phase transition (i.e. we are in the detectable region). We see that, for the dense regime, all data points belonging to different communities are well separated, and points belonging to the same community are clustered together, so the k-means algorithm can accurately recover the cluster memberships. However, for the sparse regime, all data points are clustered together, so the k-means algorithm cannot recover the cluster memberships accurately, and the overlap for the spectral clustering method will be very low indeed.

We now consider an issue associated with modularity optimisation algorithms (including the greedy method we have used), known as the *resolution limit* [22, 26]. Communities that are small when compared to the whole graph may not be distinguished even though they are well defined communities or even cliques, and therefore this problem has an impact on practical applications [22]. As [22, 26] explains, if the change in modularity, ΔQ, arising by merging two communities is positive, then the two groups will be clustered together. Let e_i be the number of edges within community i and e_{ij} be the number of edge between communities i and j. The expression for the change in modularity is then

given by

$$\Delta Q_{ij} = \frac{e_{ij}}{m} - 2\left(\frac{d_i}{2m}\right)\left(\frac{d_j}{2m}\right) \tag{4.2}$$

where d_i is the sum of the degrees of the vertices that belong to community i. Notice that $\Delta Q_{ij} > 0$ (and therefore the two communities are merged) if and only if $e_{ij} > \frac{d_i d_j}{2m}$ [26]. This is problematic because of the null model considered, the modularity tends to expect a value of $e_{ij} < 1$, which makes just a single edge between the communities (i.e. $e_{ij} = 1$) unexpected, so that two communities will be merged even though there is just one edge between them [26].

Another problem with modularity optimisation is known as *extreme near-degeneracy* [22,26]. Good et al. [26] explain that both the number of partitions with near-optimal modularity (i.e close with respect to the global maximum modularity value) grows exponentially with the number of nodes in the network [22]. This results in extreme degeneracies in the modularity which causes problems in determining the partition which maximises modularity as well as finding high modularity partitions, since partitions that have similar high modularity values associated are not necessarily similar to one another [22,26]. This means that, the more modular a network is (i.e. the more communities it contains), it is actually becomes more difficult to determine the optimum partition among the suboptimal ones, which is a counter-intuitive result [26].

To summarise both spectral clustering and modularity optimisation algorithms can detect communities up to the phase transition in the dense regime. However, we have seen how the performance declines as the ground-truth network gets sparser for both algorithms. We have explained some of the issues observed that can account for the disappointing results although there is no deep understanding regarding the modularity optimisation methods [22]. Describing the behaviour of the modularity methods is important, though, and there have been many situations where the algorithm works well in practical applications [14, 15, 22, 26]. This should not be startling given how the synthetic data we have used for our experimental tests are, in general, not well representative of real-world networks considered in practice. Since the greedy algorithm is very fast and can be applied to larger scale networks, we would recommend using this algorithm instead of other modularity optimisation methods as well as traditional spectral clustering as it achieves similar or better accuracy with faster run time. For the case of smaller size networks, the pragmatic option seems most suitable, where these different algorithms may be run and then compared in training cases, with the best one evaluated used for further investigation in test applications. We notify the reader, we shall revisit modularity optimisation algorithms as they are useful for a particular application of financial networks, further

studied in chapter 5. For the moment, nevertheless, we wish to find a class of algorithms that, empirically, seem to reliably detect communities up to the phase transition in the sparse regime also.

4.2 Belief Propagation Algorithm

4.3 NLPI and AMP Algorithms

We shall use the procedure outlined in [48] to test the NLPI and AMP algorithms, where the synthetic data used is simply an instance of a normalised adjacency matrix. Recall equation (2.16), where we decomposed the matrix into signal (weighted by a signal-to-noise ratio term) plus noise. We will consider the case of a network generated by the hidden clique model with n nodes and one hidden community consisting of k nodes. We may choose the community memberships arbitrarily. Using previous notation, let us denote $\boldsymbol{u}$ as the indicator variable for nodes belonging to the hidden community (i.e. so that precisely k elements of $\boldsymbol{u}$ are equal to one and the rest equal zero), *lambda* as the SNR and $\mathbf{Z}$ as the noise term. We set the elements of $\mathbf{Z}$ to be $Z_{ij} \sim \mathcal{N}(0, 1/n)$ i.i.d. entries. Therefore given a value of λ and our ground-truth community assignments, we can construct the normalised adjacency matrix, denoted by $\widetilde{\mathbf{A}}$, using equation (2.16). This matrix will serve as input to the NLPI and AMP algorithms, which produce an output representing their reconstructed node assignments, which we shall denote by $\widehat{\boldsymbol{u}}$. We shall represent the accuracy of the algorithms by the inner product of the ground-truth and reconstructed node assignments, $\langle \boldsymbol{u}, \widehat{\boldsymbol{u}} \rangle$. Notice this value lies between 0 and 1, and a larger number indicating improved reconstruction and better accuracy.

Now, with the framework set up, we can construct our own tests, seeking the accuracy as both the SNR and the size of the hidden community are varied. Define a new variable $\varepsilon \equiv k/n$ representing the proportion of all nodes that belong to the hidden community, then we can construct a grid of points representing different values of λ and ε. We, finally, plot the values of $\langle \boldsymbol{u}, \widehat{\boldsymbol{u}} \rangle$ for each point on the grid to analyse the behaviour.

Utilising this approach with $n = 500$ nodes, varying λ between 0 and 1.2, choosing a grid resolution of 100 and running 50 iterations of the NLPI algorithm, we show in figure 4.4, the accuracy of the NLPI algorithm plotted against λ and ε.

The motivation for the NLPI algorithm is to detect the node assignments more accurately than standard PCA for small-sized hidden communities. The results can be analysed by

FIGURE 4.4: A plot of $\langle \boldsymbol{u}, \widehat{\boldsymbol{u}} \rangle$ for the NLPI algorithm for different values of λ and ε. The synthetic data is generated from a network with $n = 500$ nodes and a grid with resolution 100, whilst the algorithm is run for 50 iterations.

FIGURE 4.5: A plot of $\langle \boldsymbol{u}, \widehat{\boldsymbol{u}} \rangle$ for the AMP algorithm for different values of λ and ε. The synthetic data is generated from the same network with $n = 500$ nodes as used in figure 4.4, and a grid with resolution 100, whilst the algorithm is run for 50 iterations.

focusing on the accuracy for small values of ε and all ranges of λ. We have previously discussed that the transition for standard PCA algorithms for small-size hidden communities is $\lambda = 1$, where, for values above this threshold, reconstruction is possible, and below, it is impossible. Figure 4.4 illustrates that for small values of ε (e.g. between 0 and 0.2), the reconstructed vector is correlated with ground-truth (i.e. $\langle \boldsymbol{u}, \widehat{\boldsymbol{u}} \rangle > 0$). In particular, we can get good reconstruction for values of λ all the way down to 0.8. Thus we have shown to beat the spectral threshold!

Although the NLPI algorithm can be shown to beat the spectral threshold empirically by using the synthetic data strategy of [48] that we used above, as we have already noted, analysing its asymptotic behaviour mathematically is not trivial. This served as the motivation for the AMP algorithm where precise asymptotics could be derived theoretically (however this is beyond the scope of this report). We shall now analyse the performance of the AMP algorithm using identical synthetic data (i.e. the same normalised adjacency matrix input for every value pair (λ, ε) in the grid) as we tested the NLPI algorithm with. We also chose to run the AMP algorithms for 50 iterations. We have plotted the accuracy for the AMP algorithm in figure 4.5.

Figure 4.5 illustrates that for small values of ε (e.g. between 0 and 0.2), the reconstructed

vector is correlated with ground-truth. In particular, we get good reconstruction for values of λ all the way down to 0.8. Thus we have shown to beat the spectral threshold once more! We also note that the striking similarity between figure 4.4 and figure 4.5 is to be expected given the formulations of these algorithms.

We have seen how NLPI and AMP algorithms can be used to detect hidden communities in networks generated by a hidden clique model. In particular we have seen the empirical improvements over standard spectral methods such as PCA. Moreover, the NLPI approach of applying a suitable non-linear function as an extra step to traditional power-iteration algorithm may be used for any problem where an eigenvector with special properties (e.g. sparsity or non-negativity) needs to be found. Although the application is beyond the scope of this report, the AMP algorithm has had much success when applied to *compressed sensing*; we refer the reader to [20, 21, 27, 28, 47] for more details.

Chapter 5

Community Detection in Financial Networks

In this chapter we aim to apply algorithms to detect communities within real-world financial networks. Firstly, we explain the process of gathering the data to create the financial networks. Then we investigate and apply different community detection algorithms and compare their performance on both synthetically generated and the real-world data.

5.1 Constructing the Real-world Financial Network

The dataset we use consists of daily closing prices of 80 stocks in the FTSE 100 index, which we obtained from [51]. The time period considered is between the beginning of 2004 to the end of 2013, a total of 2501 prices. his is the data we obtained after the removal of a few data points due to incomplete data across different stocks.The complete list of stocks is given in appendix A. We then calculated, for each stock and for each time period, the logarithmic return. We generated a time series of these returns and associated each stock with a single time series. By using the method described in section 2.3.3, we proceeded to construct financial network represented by a fully-connected, undirected and weighted graph. There are 80 nodes in this network (each one representing one of the stocks), and the weights on the edges connecting any two nodes is the cross-correlation between the time series of returns associated with the stocks represented by the two nodes. We stress, at this point, the data of the whole period (01/01/2004 - 01/01/2013) is currently

FIGURE 5.1: A plot of expected return against volatility of the 80 stocks in the FTSE 100 index considered. Data points obtained from price data during the period 01/01/2004 - 01/01/2013.

represented by one single network. Figure 5.1 shows a plot of the expected return against the volatility for each stock considered during this period.

5.1.1 Random Matrix Theory

The correlation matrix is representing the weighted adjacency matrix of the network, and in order to better understand the weights in the network, we wish to refer to an important result from Random Matrix Theory (RMT) that has been outlined in [4, 5, 12, 45]. In particular, we wish to distinguish between random and non-random properties of empirical correlation matrices.

A correlation matrix created from n random time series of length T, in the limits $n \to +\infty$ and $T \to +\infty$ with $1 < T/n < +\infty$, has a specific distribution of eigenvalues known as the *Sengupta-Mitra distribution* [4, 12, 30, 45]. This distribution is defined by

$$\rho(\lambda) = \begin{cases} \frac{T}{n} \frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{2\pi\lambda} & \text{if } \lambda_- \leq \lambda \leq \lambda_+ \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where the maximum and minimum eigenvalues (λ_+ and λ_- respectively) are given by

$$\lambda_+ = \left(1 + \sqrt{\frac{n}{T}}\right)^2 \tag{5.2}$$

and

$$\lambda_- = \left(1 - \sqrt{\frac{n}{T}}\right)^2 \tag{5.3}$$

Therefore the set of eigenvalues of an empirical correlation matrix that lies within this distribution is considered to occur purely as a result of random noise [12, 30, 45]. Moreover,

(A) (B)

FIGURE 5.2: Plots of eigenvalue spectra for empirical correlation matrix and RMT prediction, (a), in addition to a zoomed-in version, (b). The empirical correlation matrix was constructed from the daily log-returns of the FTSE 100 data set, and its eigenvalue spectrum is plotted in red. The RMT prediction is the Sengupta-Mitra distribution with appropriate parameters ($n = 80$, $T = 2501$), and is plotted in blue. The zoomed-in graph identifies the existence of eigenvalues outside of the region predicted by RMT, whilst the zoomed-out graph clearly shows the maximum eigenvalue (i.e the market mode eigenvalue) with a value of about 28.

we may regard any eigenvalue larger than λ_+ to represent important structure within the data [12, 30, 45].

Analysing the deviation of the eigenvalue spectrum of empirical correlation matrices constructed from real-world financial data from the RMT distribution constitutes an effective method to filter noise out from the data. For example, we constructed the correlation matrix from the FTSE 100 data set (described in section 5.1) and plotted the eigenvalue spectrum for this matrix alongside the corresponding Sengupta-Mistra distribution (i.e. RMT prediction with $n = 80$ and $T = 2501$) in figure 5.2. We observe two interesting regions of the eigenvalue spectrum outside the RMT prediction. Firstly, the largest eigenvalue of the correlation matrix, which we shall denote by λ_m, is much larger than all other eigenvalues. Also, the eigenvector associated with the largest eigenvalue, denoted by v_m, has all elements positive. This has been observed in many previous studies of empirical correlation matrices, and this eigenvalue is also called the *market mode* [30, 45], meaning this component acts as a common factor influencing all assets within the market [45]. Secondly, we observe a few eigenvalues just outside the RMT predicted region (i.e. eigenvalues just larger than λ_+ and much smaller than λ_m. We believe these components reflect a mesoscopic level of groups of stocks within the market (i.e. neither at the level of individual stocks in the form of noise, nor at the level of the entire market in the form of the market mode eigenvalue) [45], hence we expect members of these groups of stocks to demonstrate similar underlying properties, such as related sector classifications.

We proceed to utilise the eigenvalue spectrum observed for the data set and the RMT

prediction to filter out the empirical correlation matrix to reflect a mesoscopic structure, as achieved by [45]. Recall the correlation matrix for our FTSE 100 data set is a 80×80 matrix denoted by $\mathbf{C}$ and that we denote λ_i as the $i-th$ eigenvalue of $\mathbf{C}$ and $\boldsymbol{v}_i$ represents the eigenvector associated with λ_i. We are able to decompose this matrix as the sum of three matrices

$$\mathbf{C} = \mathbf{C}^{(r)} + \mathbf{C}^{(g)} + \mathbf{C}^{(m)} \tag{5.4}$$

where $\mathbf{C}^{(r)}$ represents the correlation matrix corresponding to the random components, defined by

$$\mathbf{C}^{(r)} \equiv \sum_{i:\lambda_i \leq \lambda_+} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T \tag{5.5}$$

$\mathbf{C}^{(m)}$ represents the correlation matrix corresponding to the market mode component, defined by

$$\mathbf{C}^{(m)} \equiv \lambda_m \boldsymbol{v}_m \boldsymbol{v}_m^T \tag{5.6}$$

and $\mathbf{C}^{(g)}$ represents the remaining correlations

$$\mathbf{C}^{(g)} \equiv \sum_{i:\lambda_+ < \lambda_i < \lambda_m} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T \tag{5.7}$$

Therefore, we now have a representation of a filtered empirical correlation matrix, $\mathbf{C}^{(g)}$, which represents the mesoscopic (group level) correlations of the stocks, which we shall use, crucially, as the input to several community detection algorithms (it can be thought of as a weighted adjacency matrix of a new filtered network).

5.2 Community Detection Algorithms

So far we have been able to construct a financial network based on the correlations of daily log returns of stocks and, using RMT, a filtered correlation matrix that represents a new financial network with links (hopefully) representing group-level correlation. Although, the question still remains, given either the initial or filtered correlation matrix, how does one produce a set of groups of stocks with greater correlations within a group than between groups? From previous sections, we understand the notion of community detection within graphs, which we shall also refer to as *binary networks*, and have analysed several algorithms that tackle this problem. However, in these problems we analysed adjacency matrices that contained binary elements (i.e. a '1' if an edge exists in the graph and a'0' otherwise), whereas, in this problem, we study a weighted adjacency matrix with elements

as real numbers. The reader should also note the adjacency matrix studied in previous sections is directly related to the structure of the network in question, whereas in this case, it is related to the weights of links between nodes. This suggests the possibility of having to modify previously studied algorithms for this scenario.

Given the conclusions of our analysis in chapter 3, and that the real-world data in this case is not well represented by generative random models, we shall use the modularity optimisation as a basis of some of our algorithms.

5.2.1 Modularity Optimisation Methods

Recall in section 3.2, we introduced the notion of modularity optimisation as a method for community detection within networks. We shall once more consider algorithms for modularity optimisation, but for the case of financial networks.

Let us denote a partition of n nodes, in the financial network, into communities/groups by the vector $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_n]^T$, where σ_i denotes the group to which node i belongs to. This is essentially the group to which stock i belongs. We then define the modularity for this partition, $Q(\boldsymbol{\sigma})$, by

$$Q(\boldsymbol{\sigma}) = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(\sigma_i, \sigma_j) \tag{5.8}$$

where $\mathbf{A}$ is the weighted adjacency matrix of the network, $k_i \equiv \sum_j A_{ij}$ and $2m \equiv \sum_{ij} A_{ij}$. Note that in the case of binary networks, A_{ij} represented the presence or absence of an edge between nodes i and j, d_i represented the degree of node i and m represented the total number of edges in the networks. Since we are interested in financial networks, a naive approach would be to use the empirical correlation matrix, denoted by $\mathbf{C}$, as the networks weighted adjacency matrix. Note that we are essentially ignoring the results of section 5.1.1, and we will see in section 5.2.2 why this is not a good idea, both intuitively and mathematically.

For now, though, let us use the following relationship

$$A_{ij} = \frac{1}{2}\big(C_{ij} + 1\big) - \delta(i, j) \tag{5.9}$$

where $\mathbf{C}$ denotes the correlation matrix, and $\delta(i, j)$ removes self edges. From this definition we simply note that $A_{ij} \in [0, 1]$.

We now focus on finding the partition, $\boldsymbol{\sigma}$, that maximises the modularity. We notice that a larger value of A_{ij} implies larger correlation between the stocks i and j, whilst a smaller value implies a lower correlation. Recalling how, for the case with binary networks, we sought after denser connections within groups and sparser connections between groups, we realise the modularity maximisation algorithms used on binary networks should have the same effect on financial networks. This is intuitive since, in both types of networks, we aim to find the partition which maximises the sum of correlations or number of edges between nodes within same community and minimises the sum of correlations or number of edges between nodes belonging to different communities.

We shall select two familiar approaches, widely known in the literature, as algorithms for the modularity maximisation. Firstly we consider a greedy agglomerative method discussed earlier in the report, and secondly a method that uses spectral relaxation.

We consider the same greedy agglomerative method described in section 3.2.1, which we noted is also applicable in this case of weighted networks. The implementation code for this algorithm is in MATLAB and has been taken from [53].

The approach using the spectral relaxation can be summarised using the argument from [54]. Let $\mathbf{B}$ denote the modularity matrix, defined by

$$B_{ij} = A_{ij} - \frac{d_i d_j}{2m} \tag{5.10}$$

where A_{ij}, d_i and m are defined as before. Also denote the set of nodes belonging to group a by $\mathcal{S}_a \equiv \{i : \sigma_i = a\}$. The algorithm iterates by attempting to split the node members of a single group in an optimal fashion by using modularity. Assume at one iteration there are q groups so the partition is indexed by $[q] \equiv \{1, 2, \ldots, q\}$. For some $a \in [q]$, let $\mathbf{B}_a$ denote the submatrix restricted to nodes in $\mathcal{S}_a$. Let $\boldsymbol{v} \in \mathbb{R}^{|\mathcal{S}_a|}$ denote the sign vector given the algorithm operating on $\mathbf{B}_a$. Then the change in modularity is given by

$$\begin{aligned} \Delta Q &= \frac{1}{2m}\left(\sum_{i,j\in\mathcal{S}_a} B_{ij}\frac{(1+v_i v_j)}{2} - \sum_{i,j\in\mathcal{S}_a} B_{ij}\right) \\ &= \frac{1}{4m}\left(\sum_{i,j\in\mathcal{S}_a} B_{ij} v_i v_j - \sum_{i,j\in\mathcal{S}_a} B_{ij}\right) \\ &= \frac{1}{4m}\boldsymbol{v}^T \widetilde{\mathbf{B}}_a \boldsymbol{v} \end{aligned} \tag{5.11}$$

where $\widetilde{\mathbf{B}}_a = \mathbf{B}_a - diag\left(\sum_{j\in\mathcal{S}_a} B_{ij}\right)$.

The algorithm accepts the splitting of the group which maximises the modularity difference, and terminates when reaches a threshold regarding the size of the groups and the possible improvement in modularity at a given iteration. We have implemented this algorithm of [54] using MATLAB, stressing the input is defined by equation (5.9).

5.2.2 Modified Modularity Optimisation Method

We move on to understand the potential issues with the naive approach hinted in the previous section. Intuitively, the term $d_i d_j/2m$ reflects the null hypothesis that the observed network structure is wholly based on the degrees of the nodes. This idea is sound when applied to binary networks, however, these terms do not have a specific meaning when applied to financial networks since we are interested in the correlation matrix rather than the structure of the underlying graph. The quantities $d_i \equiv \sum_j C_{ij}$ and $2m \equiv \sum_{ij} C_{ij}$ do not make up a clear notion of a null model when combined. Moreover, [45] have shown the naive approach is mathematically incorrect and leads to biased results. In order to detect communities within the correlation matrix that leads to correct results, we seek improvement in the construction of the null model for the network. We have already discussed, in section 5.1.1, how to construct a filtered correlation matrix that takes into account the random noise at the level of individual stocks and a market-wide component. Therefore, this filtered matrix, which we have denoted by $\mathbf{C}^{(g)}$ and defined in equation (5.7), can be used as a modularity matrix, since it reflects a suitable null model (i.e. random noise plus market wide correlation) subtracted from the observed correlation matrix.

Our aim, here, is to use $\mathbf{C}^{(g)}$ as input to an algorithm that can detect communities by maximising modularity. We hope this provides better results than the naive modularity methods discussed previously. The construction of an algorithm is not quite as straightforward, though, since we take into account the modification of the modularity matrix.

We must first, though, introduce a new formulation of modularity given a partition, as used by [45], which we denote by $Q_n(\boldsymbol{\sigma})$

$$Q_n(\boldsymbol{\sigma}) = \frac{1}{C_{norm}} \sum_{ij} C_{ij}^{(g)} \delta\left(\sigma_i, \sigma_j\right). \tag{5.12}$$

where C_{norm} is a normalisation term defined by

$$C_{norm} \equiv \sum_{ij} |C_{ij}| \tag{5.13}$$

which just ensures the value of the newly-defined modularity lies within the interval $[-1, +1]$. The new formulation of modularity is specifically aimed to detect mesoscopic-level communities.

The 'modified' modualirty method is a spectral clustering algorithm based on the exact same technique considered for binary networks considered in section 3.1, but instead we shall use the filtered correlation matrix, $\mathbf{C}^{(g)}$, as input. We simply find the eigenvectors of $\mathbf{C}^{(g)}$, use them to construct the embedded vectors, which are then clustered using the familiar k-means clustering algorithm (same procedure as with binary networks). We use different (and appropriate) values for the number of groups, then run the algorithm for each one and choose the partition with the best value for the re-defined modularity (equation (5.12)).

5.3 Synthetic Data Testing

Before applying the four algorithms discussed previously to our financial network based on the FTSE 100 data set, we must run tests to confirm we can correctly detect correlated sets of time series in synthetically generated benchmark cases, as also outlined by [45]. We consider a benchmark data set of correlation matrices with 100 time series (i.e. we consider $n = 100$ stocks in the data set) divided into 10 communities of 10 correlated time series. Note the length of the time series is chosen to be $T = 2500$ to reflect similar conditions to the real data set and is also prescribed by RMT (i.e. $T > n$). The set consists of correlation matrices generated with different levels of correlations between the groups and within a group reflecting different signal-to-noise ratios (SNR). We set the ground-truth partition, which we denote by $\boldsymbol{\sigma}^*$, to be the same across all correlation matrices. We considered 6 such benchmarks within the correlation matrix set with varying SNR, and have illustrated 2 such examples in figure 5.3. We confirmed that, in all the benchmarks, all four methods succeeded in identifying $\boldsymbol{\sigma}^*$.

(A) (B)

FIGURE 5.3: Plots of synthetically generated benchmark correlation matrices, one with low SNR (a value of 0.5), (a), in addition to one with higher SNR (a value of 1), (b). These are two examples from a set created from 100 time series divided into 10 communities of 10 correlation time series. The blocks along the diagonal represent cross-correlations between members of the same group, and are thus high (i.e. close to one), whereas off-diagonal blocks represent cross-correlations between time series belonging to different communities reflecting noise (system-wide or additional inter-community correlations).

FIGURE 5.4: Communities of the FTSE 100 generated using the greedy algorithm. The name of each label represents the associated stock's ticker (see appendix A) whilst the colour of the label represents community memberships. For this algorithm, two distinct communities were generated.

5.4 Application to Real-world Financial Network

We now apply the three modularity optimisation methods discussed in section 5.2 to the FTSE 100 data collected. The result of the greedy algorithm application is shown in figure 5.4.

FIGURE 5.5: Communities of the FTSE 100 generated using the spectral relaxation method. The name of each label represents the associated stock's ticker (see appendix A) whilst the colour of the label represents community memberships. For this method, two distinct communities were generated.

FIGURE 5.6: Communities of the FTSE 100 data generated using the spectral clustering algorithm based on the modified modularity matrix. The name of each label represents the associated stock's ticker (see appendix A) whilst the colour of the label represents community memberships. For this method, four distinct communities were generated.

Appendix A

List of Stocks

TABLE A.1: The ticker symbol, name and industry of the 80 FTSE 100 companies studied. Classifications were obtained from [50, 51].

Ticker	Name	Industry
AAL	Anglo American	Basic Materials
ABF	Associated British Foods	Consumer Goods
ADN	Aberdeen Asset Management	Financials
AGK	Aggreko	Industrials
AHT	Ashtead Group	Industrials
AMEC	AMEC	Basic Materials
ANTO	Antofagasta	Basic Materials
ARM	ARM Holdings	Technology
AV	Aviva	Financials
AZN	AstraZeneca	Health Care
BAB	Babcock International Group	Industrials
BARC	Barclays	Financials
BATS	British American Tobacco	Consumer Goods
BG	BG Group	Basic Materials
BLND	British Land Company	Technology
BLT	BHP Billiton	Basic Materials
BNZL	Bunzl	Industrials
BP	BP	Basic Materials
BRBY	Burberry Group	Consumer Goods
BSY	British Sky Broadcasting Group	Consumer Services
BT-A	BT Group	Technology

CCL	Carnival	Consumer Goods
CNA	Centrica	Utilities
CPI	Capita	Industrials
DGE	Diageo	Consumer Goods
EZJ	easyJet	Consumer Services
GFS	G4S	Industrials
GKN	GKN	Consumer Goods
GSK	GlaxoSmithKline	Health Care
HMSO	Hammerson	Financials
HSBA	HSBC Holdings	Financials
IAG	International Consolidated Airlines Grp	Consumer Services
IHG	InterContinental Hotels Group	Consumer Services
IMT	Imperial Tobacco Group	Consumer Goods
ITRK	Intertek Group	Industrials
ITV	ITV	Consumer Services
JMAT	Johnson Matthey	Basic Materials
KGF	Kingfisher	Consumer Services
LAND	Land Securities Group	Financials
LGEN	Legal & General Group	Financials
LLOY	Lloyds Banking Group	Financials
LSE	London Stock Exchange Group	Financials
MGGT	Meggitt	Industrials
MKS	Marks and Spencer Group	Consumer Services
MRO	Marathon Oil Corporation	Oil & Gas
MRW	Wm. Morrison Supermarkets	Consumer Services
NG	National Grid	Utilities
NXT	NEXT	Consumer Services
OML	Old Mutual	Financials
PRU	Prudential	Financials
PSN	Persimmon	Industrials
PSON	Pearson	Consumer Services
RB	Reckitt Benckiser Group	Consumer Goods
RBS	Royal Bank of Scotland Group	Financials
RDSB	Royal Dutch Shell	Basic Materials
REL	Reed Elsevier	Consumer Services
REX	Rexam	Consumer Goods

RIO	Rio Tinto	Basic Materials
RR	Rolls-Royce Holding	Industrials
RRS	Randgold Resources Limited	Basic Materials
SBRY	J Sainsbury	Consumer Services
SDR	Schroders	Financials
SGE	The Sage Group	Technology
SHP	Shire	Health Care
SMIN	Smiths Group	Industrials
SN	Smith & Nephew	Health Care
SSE	SSE	Utilities
STAN	Standard Chartered	Financials
SVT	Severn Trent	Utilities
TATE	Tate & Lyle	Consumer Goods
TPK	Travis Perkins	Industrials
TSCO	Tesco	Consumer Services
ULVR	Unilever	Consumer Goods
UU	United Utilities Group	Utilities
VOD	Vodafone Group	Technology
WEIR	The Weir Group	Industrials
WMH	William Hill	Consumer Services
WOS	Wolseley	Industrials
WPP	WPP ORD 10P	Consumer Services
WTB	Whitbread	Consumer Services

Bibliography

[1] E. P. Wigner, "Characteristic vectors of bordered matrices with infinite dimensions," in *The Annals of Mathematics, 62*, pp. 548–564, 1955.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," 1983, `http://leonidzhukov.net/hse/2013/stochmod/papers/KirkpatrickGelattVecchi83.pdf`.

[3] D. G. Luenberger, "Investment science," 1998.

[4] A. M. Sengupta and P. P. Mitra, "Distributions of singular values for some random matrices," *Phys. Rev. E*, vol. 60, pp. 3389–3392, 1999.

[5] V. Plerou, P. Gopikrishnan, B. Rosenow, L. A. N. Amaral, and H. E. Stanley, "Universal and non-universal properties of cross-correlations in financial time series," 1999, arXiv:cond-mat/9902283v1.

[6] J.-P. Onnela, "Taxonomy of Financial Assets." `http://jponnela.com/`, 2002.

[7] J.-P. Onnela, A. Chakraborti, K. Kaski, and J. Kertesz, "Dynamic asset trees and portfolio analysis," 2002, arXiv:cond-mat/0208131v1.

[8] J.-P. Onnela, K. Kaski, and J. Kertesz, "Clustering and information in correlation based financial networks," 2003, arXiv:cond-mat/0312682v1.

[9] M. E. J. Newman, "The structure and function of complex networks," 2003, arXiv:cond-mat/0303516v1.

[10] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," 2004, `http://www.ece.unm.edu/ifis/papers/community-moore.pdf`.

[11] R. Guimera, M. Sales-Pardo, and L. A. N. Amaral, "Modularity from fluctuations in random graphs and complex networks," 2004, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2441765/pdf/nihms-34161.pdf.

[12] J. Duch and A. Arenas, "Financial Applications of Random Matrix Theory: Old Laces and New Pieces," 2005, arXiv:cond-mat/0501368v1.

[13] M. Potters, J. Bouchaud, and L. Laloux, "Community detection in complex networks using Extremal Optimization," 2005, arXiv:physics/0507111v1.

[14] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," 2006, arXiv:physics/0605087v3.

[15] M. E. J. Newman, "Modularity and community structure in networks," 2006, arXiv:physics/0602124v1.

[16] U. von Luxburg, "A tutorial on spectral clustering," 2006, http://www.stanford.edu/class/ee378b/papers/luxburg.pdf.

[17] D. Spielman, "Spectral graph theory and its applications," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pp. 29–38, 2007.

[18] L. K. Chan, J. Lakonishok, and B. Swaminathan, "Industry Classifications and Return Comovement," in *Financial Analysts Journal, Vol. 63, No. 6*, 2007.

[19] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," 2008, arXiv:0803.0476v2.

[20] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," 2009, http://www.stanford.edu/~montanar/RESEARCH/FILEPAP/mpacs.pdf.

[21] A. Maleki, D. L. Donoho, and A. Montanari, "Analysis of Approximate Message Passing Algorithm," 2010, http://www.ece.rice.edu/~mam15/CISS2010.pdf.

[22] S. Fortunato, "Community detection in graphs," 2010, arXiv:0906.0612v2.

[23] J. Leskovec, K. J. Lang, and M. W. Mahoney, "Empirical Comparison of Algorithms for Network Community Detection," 2010, arXiv:1004.3539v1.

[24] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," 2010, arXiv:1008.3926v1.

[25] D. J. Fenn, M. A. Porter, P. J. Mucha, M. McDonald, S. Williams, N. F. Johnson, and N. S. Jones, "Dynamical Clustering of Exchange Rates," 2010, arXiv:0905.4912v2.

[26] B. H. Good, Y.-A. de Montjoye, and A. Clauset, "The performance of modularity maximization in practical contexts," 2010, arXiv:0910.0165v2.

[27] A. Montanari, "Graphical Models Concepts in Compressed Sensing," 2011, arXiv:1011.4328.

[28] M. Bayati and A. Montanari, "The dynamics of message passing on dense graphs, with applications to compressed sensing," 2011, arXiv:1001.3448v4.

[29] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborova, "Phase transition in the detection of modules in sparse networks," 2011, arXiv:1102.1182v1.

[30] D. J. Fenn, M. A. Porter, S. Williams, M. McDonald, N. F. Johnson, and N. S. Jones, "Temporal Evolution of Financial Market Correlations," 2011, arXiv:1011.3225v2.

[31] D. Kuhn, "The basic theory of interest," *Computational Finance: 422 Lecture Notes*, 2012.

[32] D. Kuhn, "Fixed-income securities," *Computational Finance: 422 Lecture Notes*, 2012.

[33] D. Kuhn, "Mean variance portfolio theory," *Computational Finance: 422 Lecture Notes*, 2012.

[34] D. Spielman, "Spectral graph theory and its applications," in *Combinatorial Scientific Computing. Chapman and Hall / CRC Press*, 2012.

[35] R. R. Nadakuditi and M. E. J. Newman, "Graph spectra and the detectability of community structure in networks," 2012, arXiv:1205.1813v1.

[36] S. Ertekin, H. Hirsh, and C. Rudin, "Learning to Predict the Wisdom of Crowds," 2012, arXiv:1204.3611v1.

[37] E. Mossel, J. Neeman, and A. Sly, "Stochastic Block Models and Reconstruction," 2012, arXiv:1202.1499v4.

[38] J. Yang and J. Leskovec, "Defining and Evaluating Network Communities based on Ground-truth," 2012, arXiv:1205.6233v3.

[39] M. Gomez-Rodriguez, J. Leskovec, and A. Krause, " Inferring Networks of Diffusion and Influence," 2012, arXiv:1006.0234v3.

[40] E. Mossel, J. Neeman, and A. Sly, "Belief Propagation, Robust Reconstruction, and Optimal Recovery of Block Models," 2013, arXiv:1309.1380v1.

[41] F. Krzakala, C. Moore, E. Mossel, J. Neeman, A. Sly, L. Zdeborova, and P. Zhang, "Spectral redemption: clustering sparse networks," 2013, arXiv:1306.5550v2.

[42] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborova, "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications," 2013, arXiv:1109.3041v2.

[43] L. Massoulie, "Community detection thresholds and the weak Ramanujan property," 2013, arXiv:1311.3085v1.

[44] E. Mossel, J. Neeman, and A. Sly, "A Proof Of The Block Model Threshold Conjecture," 2013, arXiv:1311.4115v1.

[45] M. MacMahon and D. Garlaschelli, "Unbiased community detection for correlation matrices," 2013, arXiv:1311.1924v1.

[46] Y. Deshpande and A. Montanari, "Finding Hidden Cliques of Size sqrt(N/e) in Nearly Linear Time," 2013, arXiv:1304.7047v1.

[47] J. Barbier, F. Krzakala, and C. Schulke, "Compressed sensing and Approximate Message Passing with spatially-coupled Fourier and Hadamard operators," 2013, arXiv:1312.1740v2.

[48] "Talk at simons institute for the theory of computing: Finding small structures in large datasets." `http://simons.berkeley.edu/talks/andrea-montanari-2013-11-18`, cited in March 2014.

[49] Z. Bodie, A. Kane, and A. J. Marcus, "Essentials of investments ninth edition," 2013.

[50] "Wikipedia: Industry classifications benchmark." `http://en.wikipedia.org/wiki/Industry_Classification_Benchmark`, cited in March 2014.

[51] "Yahoo finance website." `http://uk.finance.yahoo.com/`, cited in March 2014.

[52] "Quantivity wordpress blog." `http://quantivity.wordpress.com/2011/02/21/why-log-returns/`, cited in March 2014.

[53] "Community detection greedy algorithm code." `http://www.elemartelot.org/index.php/programming/cd-code`, cited in April 2014.

[54] "Andrea montanari webpage article." http://www.stanford.edu/class/ee378b/hw4sol.pdf, cited in April 2014.

[55] Y. Deshpande and A. Montanari, "Information-theoretically Optimal Sparse PCA," 2014, arXiv:1402.2238v1.