A Renormalizable Random Graph Model

THESIS

submitted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE
in
PHYSICS AND MATHEMATICS

Author :	Tyron Darnell Lardy
Student ID :	1520881
Supervisor :	Diego Garlaschelli
	Evgeny Verbitskiy
2nd corrector :	n.a.

Leiden, The Netherlands, June 29, 2018

A Renormalizable Random Graph Model

Tyron Darnell Lardy

Instituut-Lorentz, Leiden University
P.O. Box 9500, 2300 RA Leiden, The Netherlands

June 29, 2018

Abstract

The availability of information about complex networks is severely restrained by issues, such as confidentiality and privacy. This poses a problem when analysing properties of networks that are of relevance to the general public. One example is the study of the resilience of banking networks to financial distress. We discuss a random graph model that reconstructs such unavailable networks, based on information that is either node specific or specific to groups of nodes. The procedure is determined by enforcing renormalizability, i.e. consistency when modelling renormalizations of networks. Then we propose a weighted semi-renormalizable extension of this model. Both models are tested on an empirical trade network, by analysing how well they captures properties that are commonly used to characterize networks. Their performances are shown to closely resemble that of the (weighted) fitness-induced Configuration Model.

Contents

Chapter 1

Introduction

Complex networks can be used to represent a wide range of systems, such as the internet, ecosystems, social, and financial networks. Due to the high importance of many such systems in day-to-day life, there is a great desire to understand them. To this end, it is useful to consider complex networks as graphs, so that they can be studied from a mathematical point of view.

Definition 1. A *graph* is an ordered pair $G = (V, E)$ of a set V of vertices and a set $E \subset V^2$ of edges. An edge $(i, j) \in E$ denotes a connection between the vertices $i, j \in V$. The adjacency matrix of G is the $|V| \times |V|$ matrix $A = (a_{ij})$, where $a_{ij} = 1$ if there is a connection between the vertices i and j and $a_{ij} = 0$ otherwise.

Vertices serve as any hub in a system and edges represent some relation between these hubs. For example, vertices might stand for financial institutions and edges could denote a financial commitment between them, such as loans. The set of edges of a graph can either be ordered or unordered. In the latter case all relations are reciprocated and in the former, relations are directed from one vertex to another. In most networks, there is some sense of how strong these relations between vertices are. Such networks are represented by weighted graphs.

Definition 2. A *weighted graph* is a graph $G = (V, E)$, together with a function $w : E \to \mathbb{R}_{\geq 0}$, such that $w(e)$ is the weight on edge $e \in E$. The weighted adjacency matrix of G is the $|V| \times |V|$ matrix $W = (w_{ij})$, where w_{ij} denotes the weight on the connection between i and j. The relation between the adjacency matrix of a graph and its weighted counterpart is $a_{ij} = \mathbb{1}_{w_{ij}>0}$.

When studying networks, one finds that the topology of networks heavily affects the dynamics and properties of the represented systems. As an example, the topology of banking networks affects the stability of the system under financial distress, and the structure of social networks influences the propagation of diseases [1, 2]. However, limited availability of information often prevents us from knowing the exact topology of real world networks. In order to nevertheless be able to study their dynamics and properties, it is appropriate to generate similar networks as a model. This modelling is often done with random graphs.

Definition 3. A *random graph* is a graph G, in which the connections between vertices are determined by probability. That is, the entries a_{ij} of the adjacency matrix are bernoulli random variables: $a_{ij} \sim \text{Ber}(p_{ij})$. A *weighted random graph* is a graph G in which the weights on the edges are determined by probability. That is, the entries w_{ij} of the adjacency matrix are random variables that follow a non-negative probability distribution.

In this thesis, we first outline the properties that are often computed for the characterization of networks. Afterwards, the combination of two ways in which the availability of information about a network can be limited will be considered. The first is that there might be no knowledge whatsoever of the links of the network. Instead, for each node there might only be given a measure of how likely it is to create (in- and outgoing) connections, which we refer to as the node (in- and out) fitness. Secondly, information might be presented at multiple scales of a network. This means that the available information can be specific to groups of nodes, instead of individual nodes. We will discuss a random graph model that is able to consistently use this limited multi-scale information to reconstruct the unweighted topology of networks. Finally, we will extend the unweighted model to be able to reconstruct the weighted topology of networks. Both the unweighted and weighted version of the model are assessed by analysing how well they reproduce the network properties of an empirical trade network.

Chapter 2

Network properties

We start by recalling some important notions and concepts that are used to characterize networks. There are several widely used computations that provide numerical characterisation of networks. This chapter aims to provide an overview and explanation of the most commonly used ones.

2.1 Undirected properties

In the first part of this section, we will consider an unweighted (binary) and undirected graph $G = (V, E)$. First of all, we define a notion of how big G is: the *size* of G is the number of nodes $|V|$. Secondly, the *density* $D(G)$ is the fraction of possible edges that are actually present in the graph:

$$D(G) = \frac{|E|}{\binom{n}{2}} = \frac{2|E|}{n(n-1)}.$$

Note that $D(G)$ ranges from 0 to 1. The maximum of 1 corresponds to a fully connected graph and the minimum of 0 to a graph without edges. The closer the value is to 1, the more connected the nodes in the network are and the more *dense* the network is. Conversely, the closer the value is to 0, the less connected the nodes are and the more *sparse* the network is. The density of a network is a global measure of connectedness, since it is associated with the entire graph. We now define a local measure of connectedness, the degree, which is specific to a single node. Accordingly, let $i \in V$ be an arbitrary node.

Definition 4. The *degree* of i is the number of connections it has

$$k_i = \sum_j a_{ij},$$

where (and in the remainder of this chapter), the sum over j spans over values $i \neq j$.

The empirical distribution of the degrees of the nodes in G is called the *degree distribution*. Many real world networks show a degree distribution that asymptotically follows a power law. That is, for large values of k, the fraction $P(k)$ of nodes with a degree of k is

$$P(k) \sim k^{-\gamma},$$

where $\gamma \in \mathbb{R}_{>1}$. For real-life examples, commonly in the range $2 < \gamma < 3$. Networks with this property are called scale-free networks. In order to more easily discuss the connectedness of i, any node that i has a connection to, is called a neighbour of i, and all of its neighbours and edges between them together are referred to as its neighbourhood. Generally, the nodes in the neighbourhood of i are either more or less likely to have a similar degree as i. This is called assortative or disassortative mixing respectively. Now, for an explanation, imagine a friendship network. It is more likely for popular people to be connected to other popular people and for unpopular people to be connected to unpopular people. Friendship networks thus show assortative mixing. The average nearest neighbour degree is defined to quantify this concept:

Definition 5. The *average nearest neighbour degree* of i is:

$$k_i^{nn} = \frac{\sum_j a_{ij} k_j}{k_i}.$$

A positive correlation between k_i and k_i^{nn} then indicates assortative mixing and a negative correlation disassortative mixing.

Let us now return to the example of a friendship network. What we would most likely find when studying such a network is that any two friends of a person are generally themselves friends, since friends tend to form groups. We say that the nodes of the network are clustered together. The degree to which the neighbourhood of i is clustered can be measured by the fraction of pairs of i's neighbours that are connected. This is known as the clustering coefficient of i.

Definition 6. The *clustering coefficient* of i is:

$$C_i = \frac{2\sum_{j,k:j\neq k} a_{ij} a_{jk} a_{ik}}{k_i(k_i - 1)} = \frac{2\sum_{j,k:j\neq k} a_{ij} a_{jk} a_{ik}}{\sum_{j,k:j\neq k} a_{ji} a_{ki}}.$$

In other words, the clustering coefficient can be seen as the fraction of possible triangles between i and its neighbours that is present in the network. The denominator $k_i(k_i-1)$ gives the maximal possible number of triangles between i and its neighbours. The summation in the numerator counts the number of those present in the graph.

The measures discussed so far are used to describe the binary structure of graphs. We can generalise them for the characterisation of weighted graphs. Therefore, let G now be an undirected weighted graph with weighted adjacency matrix $W=(w_{ij})$. The idea of node degree, for one, can be generalized to the notion of node strength.

Definition 7. The *strength* of i is

$$s_i = \sum_j w_{ij}.$$

Similarly, the average nearest neighbour degree has a weighted counterpart called the average nearest neighbour strength.

Definition 8. The *average nearest neighbour strength* of i is

$$s_i^{nn} = \frac{\sum_j a_{ij} s_j}{k_i} = \frac{\sum_{j,k:j\neq k} a_{ij} w_{jk}}{k_i}.$$

For the extension of the clustering coefficient, weights on the edges in the neighbourhood of i must be used to assign a value to each triangle. There is, however, not just one way to do this. Suppose, for example, that there is a triangle between nodes $i,j,k\in V$. Then the product of all three weights in the triangle $w_{ij}w_{ik}w_{jk}$ might be used, or instead one could choose for example to use only the product of the two weights $w_{ij}w_{ik}$. We will consider here the adaptation that uses the geometric mean of all three weights introduced in [3].

Definition 9. The *weighted clustering coefficient* of i is

$$C_i^w = \frac{2\sum_{j,k:j\neq k}(\tilde{w}_{ij}\tilde{w}_{jk}\tilde{w}_{ik})^{1/3}}{k_i(k_i-1)} = \frac{2\sum_{j,k:j\neq k}(\tilde{w}_{ij}\tilde{w}_{jk}\tilde{w}_{ik})^{1/3}}{\sum_{j,k:j\neq k} a_{ji}a_{ki}},$$

where $\tilde{w}_{ij} = w_{ij}/\max_{i,j} w_{ij}$.

An interpretation is that it is the unweighted clustering coefficient renormalised by the average geometric mean of the "triangle weights" in the neighbourhood of i. It has the property that $C_i^w \to C_i$ as $w_{ij}\to c\in\mathbb{R}_{\geq 0}$ for all $i,j\in V$. This is desirable, because a network with the same weight on all links can be considered binary.

2.2 Directed properties

In the same way that there are multiple possible ways to extend the clustering coefficient to weighted networks, not all of the properties outlined in the previous section have just one counterpart for directed networks. The size and density of a network being the exceptions. Therefore, let $G = (V, E)$ be a directed graph. The size of G is again the number of nodes $|V|$; and the density $D(G)$ is again the fraction of possible edges that are actually present in the graph. Although the number of possible edges is now twice as large:

$$D(G) = \frac{|E|}{2\binom{n}{2}} = \frac{|E|}{n(n-1)}.$$

Since there are now incoming as well as outgoing connections, there is not just a single notion of how connected any node $i \in V$ is. Rather, there is a measure that counts the number of incoming connections a node has, its in-degree; and one that counts the number of outgoing connections a node has, its out-degree.

Definition 10. The *in-degree* of i is the number of incoming connections it has

$$k_i^{in} = \sum_j a_{ji}$$

Definition 11. The *out-degree* of i is the number of outgoing connections it has

$$k_i^{out} = \sum_j a_{ij}.$$

Nodes that i has a connection to are called its outward neighbours and nodes that have a connection to i are the inward neighbours. The in/out-degree distribution is the empirical distribution of the in/out-degrees of the nodes of G. Finally, the total degree of i is the sum of its in- and out-degree.

Definition 12. The *total degree* of i is the total number of connections, inward or outward

$$k_i^{tot} = k_i^{in} + k_i^{out}.$$

A useful measure counts how many of the connections i has are reciprocal.

Definition 13. The number of connections i has that are reciprocal is

$$k_i^{\leftrightarrow} = \sum_j a_{ij}a_{ji}.$$

Because of the dichotomy between in- and outgoing connections, there are numerous ways to extend the average nearest neighbour degree to the directed case. In general, one could look at the average of any of the degrees outlined above of either the inward, outward, or bilateral neighbours of i. Of all the twelve possibilities, we only consider two

Definition 14. The *average nearest neighbour in-degree* and *average nearest neighbour out-degree* of i are defined as

$$
\begin{aligned}
k_i^{nn,in} &= \frac{\sum_j a_{ji} k_j^{in}}{k_i^{in}} = \frac{\sum_{j,k:j\neq k} a_{ji} a_{kj}}{\sum_j a_{ji}} \\
k_i^{nn,out} &= \frac{\sum_j a_{ij} k_j^{out}}{k_i^{out}} = \frac{\sum_{j,k:j\neq k} a_{ij} a_{jk}}{\sum_j a_{ij}}.
\end{aligned}
$$

Recall that the clustering coefficient of i gives the number of possible triangles i could possibly form with its neighbours that are actually present in the graph. The idea of a triangle is, however, ambiguous for directed graphs. One could count two inward neighbours of i that have any connection between themselves as a triangle containing i. Since any inward neighbour of i can have a connection to any of $k_i - 1$ other inward neighbours of i, in this case the total possible triangles is given by $k_i^{in}(k_i^{in}-1)$. Alternatively, one might count as a triangle two outward neighbours of i that have any connection between themselves. Analogous to the last scenario, $k_i^{out}(k_i^{out}-1)$ denotes the total number of triangles in this case. The understanding we adapt here is the one where any two neighbours (both outward and inward) of i that are connected are counted as a triangle. In a sense, this is the least restrictive approach, since a triplet nodes that are in any way pairwise connected is considered to be a triangle. Naively, we might think that the total number of possible triangles is now again simply given by $k_i^{tot}(k_i^{tot}-1)$. However, this counts too many triangles. This is because for reciprocal neighbours, there are only $k_i^{tot}-2$ other neighbours they can form an edge to, since they are counted twice in k_i^{tot}. This occurs $2k_i^{\leftrightarrow}$ times (once for every reciprocal node counted by k_i^{tot}) and therefore the total number of possible triangles is given by $k_i^{tot}(k_i^{tot}-1)-2k_i^{\leftrightarrow}$. Thus, the number of these triangles that is realised is given by $1/2\sum_{j,k:j\neq k}(a_{ij}+a_{ji})(a_{jk}+a_{kj})(a_{ik}+a_{ki})$. The factor $1/2$ is used to compensate for the fact that for two connected neighbours $j,h \in V$ of i, the corresponding triangle is counted twice.

Definition 15. The *directed clustering coefficient* of node i is:

$$
C_i^d = \frac{1/2\sum_{j,k:j\neq k}(a_{ij}+a_{ji})(a_{jk}+a_{kj})(a_{ik}+a_{ki})}{k_i^{tot}(k_i^{tot}-1)-2k_i^{\leftrightarrow}}.
$$

This definition of the directed clustering coefficient has the nice property that if the adjacency matrix is symmetric, then we have $C_i^d = C_i$. That this is true can be seen by the fact that if A is symmetric, then $(a_{ij} + a_{ji}) = 2a_{ij}$. All the factors of 2 then cancel out, leaving us with the definition of the undirected clustering coefficient:

$$\begin{aligned} C_i^d &= \frac{1/2 \sum_{j,k:j\neq k}(a_{ij}+a_{ji})(a_{jk}+a_{kj})(a_{ik}+a_{ki})}{\sum_{j,k:j\neq k}(a_{ij}+a_{ji})(a_{jk}+a_{kj})} \\ &= \frac{1/2 \sum_{j,k:j\neq k} 8a_{ij}a_{jk}a_{ik}}{\sum_{j,k:j\neq k} 4a_{ij}a_{jk}} \\ &= \frac{\sum_{j,k:j\neq k} a_{ij}a_{jk}a_{ik}}{\sum_{j,k:j\neq k} a_{ij}a_{jk}} = C_i. \end{aligned}$$

Moreover, if G has a weighted adjacency matrix $W = (w_{ij})$, we can extend the in- and outdegree to the in- and outstrength.

Definition 16. The *in-strength* and *out-strength* of node i are respectively given by:

$$s_i^{in} = \sum_j w_{ij} \text{ and } s_i^{out} = \sum_j w_{ji}.$$

Similarly, we can extend the total degree to the total strength

Definition 17. The *total strength* of i is:

$$s_i^{tot} = s_i^{in} + s_i^{out}.$$

Even though it is possible, we will not introduce any weighted variants of the reciprocal degree. Such notions are not widely used in the literature. The weighted extensions of the average nearest neighbour indegree and the average nearest neighbour outdegree are the average nearest neighbour in-strength and the average nearest neighbour out-strength respectively.

$$\begin{aligned} s_i^{nn,in} &= \frac{\sum_j a_{ji}s_j^{in}}{k_i^{in}} = \frac{\sum_{j,k:j\neq k} a_{ji}w_{kj}}{\sum_j a_{ji}} \\ s_i^{nn,out} &= \frac{\sum_j a_{ij}s_j^{out}}{k_i^{out}} = \frac{\sum_{j\neq k} a_{ij}w_{jk}}{\sum_j a_{ij}}. \end{aligned}$$

For the weighted and directed version of the clustering coefficient, we simply combine our understanding of the value assigned to weighted triangles and that of directed triangles. This leads to the the definition of the weighted directed clustering coefficient.

Definition 18. The *weighted directed clustering coefficient* of node i is:

$$C_i^{w,d} = \frac{1/2 \sum_{j,k:j\neq k} (\tilde{w}_{ij} + \tilde{w}_{ji})(\tilde{w}_{jk} + \tilde{w}_{kj})(\tilde{w}_{ik} + \tilde{w}_{ki})}{k_i^{tot}(k_i^{tot} - 1) - k_i^{\leftrightarrow}},$$

where $\tilde{w}_{ij} = w_{ij} / \max_{i,j} w_{ij}$.

Just as for the weighted (undirected) clustering coefficient, we see that $C_i^{w,d} \to C_i^d$ as $w_{ij} \to c \in \mathbb{R}_{\geq 0}$ for all $i, j \in V$. Moreover, if W is symmetric, we have $C_i^{w,d} = C_i^w$, just as $C_i^d = C_i$ if A is symmetric.

Chapter 3

Unweighted renormalizable graph model

As mentioned earlier, computing the network properties introduced in the previous chapter on real life networks is often prohibited by a lack of available information regarding such networks. Often, we do not have any information regarding the exact links of the networks, but rather we only have some node specific information. An example would be a banking network of loans, where each bank is only legally required to disclose their total debt and loans, rather than exactly how much they loan from and to each other bank. On top of that, information can be given at different scales. This means that instead of having specific information about each individual node, we might only have information about groups of nodes. Returning to the example of the banking network, this could mean that instead of having the total debts and loans from each bank in some country, we only know the total debt and loan from all the banks in that country together. In this chapter, we will formalise the concept of groups of nodes and then discuss a random graph model that consistently uses limited and multi-scale information.

For this model, we assume that there is some directed network $G_0 = (V_0, E_0)$, about which limited information is available. That is, we assume that the number of nodes and the number of edges of the network are known, i.e. $|V_0| = N \in \mathbb{N}$ and $|E_0| = L \in \mathbb{N}$. On top of that, we suppose that for each node $i \in V_0$ we know a pair of parameters x_i and y_i, its in- and out-fitness. The in-/out-fitness of a node should be considered as a measure of how likely this node is to create in-/outgoing connections to other nodes. In principle, any proxy can be used for this measure. For simplicity, we will assume here that the real network underlying G_0 was weighted and that for each node we know the in- and out-strength. It is then common practice ([4–6]) to associate the fitnesses with the strengths:

$$x_i := s_i^{out} \text{ and } y_i := s_i^{in}.$$

3.1 Renormalization procedure

The lack of an inherent geometric scale makes talking about different scales of a graph ambiguous. The renormalization of G_0, that is the way we define different scales of the network, thus must be defined more rigorously. To that end, the nodes are grouped together in any arbitrary way and then the connections between these groups are considered (see Figure 3.1). This construction of the connections between groups is then considered a renormalization of G_0.

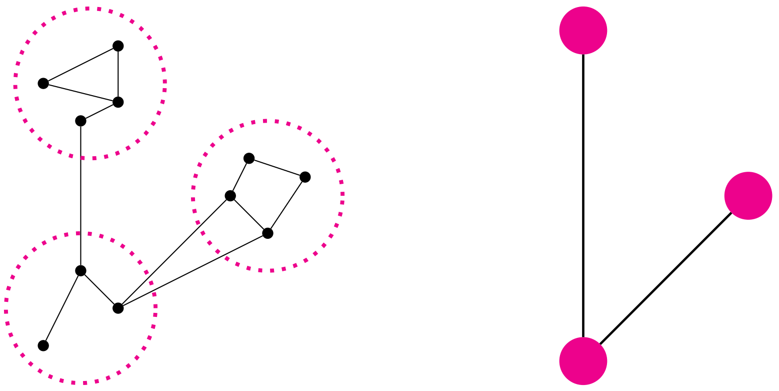
(a) *Original graph.* **(b)** *Renormalized graph.*

Figure 3.1: *Renormalization of an undirected graph.*

More formally, let $P = \{V_i\}_{i=1}^n$ be an arbitrary non-trivial partition of V_0. This partition represents the groups in which the nodes are put together.

Definition 19. The *quotient graph* of G_0 induced by P is the graph G_0/P with vertex set P and edge set $\{(V_i, V_j)|\exists u \in V_i, v \in V_j \text{ s.t } (u, v) \in E_0\}$ [7].

The quotient graph G_0/P is then called a renormalization of G_0. We refer to the nodes of G_0/P with the corresponding set names and thus do not make a clear distinction between the nodes of G/P and the sets $V_1, \ldots V_n$. The relation between the adjacency matrix of G and G/P then is

$$a_{V_iV_j} = 1 - \prod_{\substack{u\in V_i \\ v\in V_j}} (1 - a_{uv}), \quad \text{for } i, j = \{1, \ldots, n\}. \tag{3.1}$$

Indeed, the left hand side is equal to 1 if and only if at least one of the terms in the product vanishes. This corresponds to the fact that two groups are connected only if there exists at leasts one connection between the elements of the groups. The fitness of a group is defined as

$$x_{V_i} := \sum_{u \in V_i} x_u \quad \text{for} \quad i = \{1, \dots, n\}. \tag{3.2}$$

For weighted graphs, we define the the weights on the edges of a renormalization of the graph as

$$w_{V_i V_j} := \sum_{\substack{u \in V_i \\ v \in V_j}} w_{uv} \quad \text{for} \quad i, j = \{1, \dots, n\}. \tag{3.3}$$

3.2 Reconstruction of the binary topology

The model outlined in this section was introduced in [REF]. To specify the model, we make two assumptions.

1) The probability p_{ij} of creating a connection from node i to node j is a function of the out-fitness of node i and the in-fitness of node j. That is $p_{ij} = p(x_i, y_j)$ for some continuous function $p : \mathbb{R}^2 \to [0,1]$. For this connection probability, we do not distinguish between groups of nodes or nodes, i.e. this same function is used when modelling renormalizations of G_0. The fundamental idea underlying the latter part of this assumption is that the interaction between individual nodes in some way mimics the interaction between groups of nodes. In a sense, meaning that our model is self-similar.

2) The model is *renormalizable*. That is, for any partition $P = \{V_i\}_{i=1}^n$ of V_0, equation (3.1) holds. The entries $a_{V_i V_j}$ and a_{uv} here are random variables, which according to hypothesis 1 are equal to 1 with probability $p(x_{V_i}, y_{V_j})$ and $p(x_u, y_v)$ respectively. By equality of random variables, we mean equality in distribution. The first is that we assume the renormalized connections between groups of nodes are independent of the exact underlying configuration of nodes, but rather depend only on the combined fitnesses as in (3.2). Therefore, we assume that the probability of creating a connection between two groups is the same as creating a connection between any of the nodes of the groups

$$p_{V_i V_j} = 1 - \prod_{\substack{u \in V_i \\ v \in V_j}} (1 - p_{uv}).$$

From which equation (3.1) directly follows, since the Bernoulli distributions is fully determined by its expected value. This condition is called renormalizability, because it means that it does not matter whether we start with information about a renormalized version of the graph and then create a model or we create a model of the full graph and then renormalize.

The combination of these two assumptions fully determines our model:

Theorem 1. *The function p is given by:*

$$p(x, y) = 1 - e^{-\lambda xy},$$

for some $\lambda \in \mathbb{R}$.

Proof. Let $A, B \subset V_0$ be two disjoint set of nodes. Since they are disjoint, there is always a partition that contains both A and B, thus hypothesis 2 tells us that (3.1) holds:

$$1 - p_{AB} = \prod_{u \in A} \prod_{v \in B} (1 - p_{uv}).$$

From hypothesis 1 and equation (3.2) follows:

$$\begin{aligned} 1 - p(x_A, x_B) &= \prod_{u \in A} \prod_{v \in B} (1 - p(x_u, x_v)) \\ 1 - p(\sum_{u \in A} x_u, \sum_{v \in B} x_v) &= \prod_{u \in A} \prod_{v \in B} (1 - p(x_u, x_v)). \end{aligned}$$

Substituting $g = 1 - p$:

$$g(\sum_{u \in A} x_u, \sum_{v \in B} x_v) = \prod_{u \in A} \prod_{v \in B} g(x_u, x_v).$$

Finally, taking the natural logarithm on both sides and substituting $f = \ln(g)$:

$$\begin{aligned} \ln(g(\sum_{u \in A} x_u, \sum_{v \in B} x_v)) &= \ln\left(\prod_{u \in A} \prod_{v \in B} g(x_u, x_v)\right) \\ \ln(g(\sum_{u \in A} x_u, \sum_{v \in B} x_v)) &= \sum_{u \in A} \sum_{v \in B} \ln(g(x_u, x_v)) \\ f(\sum_{u \in A} x_u, \sum_{v \in B} x_v) &= \sum_{u \in A} \sum_{v \in B} f(x_u, x_v). \end{aligned}$$

Since the fitnesses can have any value, f is an additive function. The continuity of p implies that f, too, is continuous. By theorem 1 in appendix A, therefore f is linear and thus

$$f(x,y) = -\lambda xy$$

for some $\lambda \in \mathbb{R}$. $\square$

The reason we write $p_{ij} = 1 - e^{-\lambda x_i y_j}$ rather than $p_{ij} = 1 - e^{\lambda x_i y_j}$ is that we can now see λ as a direct measure of the density of our final graph. For any pair of nodes i and j, if we increase λ, p_{ij} increases as well and vice versa. This means that for a higher value of λ, more links will be realized in our model. The value of λ is estimated by imposing that the expected number of links of the random graph, L^*, should be equal to L:

$$\begin{aligned} L^* &= \mathbb{E}[\sum_{i,j} a_{ij}] \\ &= \sum_{i,j} p_{ij} \\ &= \sum_{i,j} 1 - e^{-\lambda x_i y_j} = L. \end{aligned}$$

3.2.1 Analysis on a trading network

We will now analyse how well the proposed random graph model captures the properties of a real world network. For this, we will look at the trade flow network of 96 commodities between 207 countries in the year 2008. Nodes represent the countries and edges between them represent whether or not two countries trade any of the 96 commodities. These trades are directed in a sense that export to another country is considered as an outgoing edge and import as an incoming edge. The weight on the edges stands for how much capital is involved in the export or import of these commodities. Even though this is a weighted network, the model can still be used to compute its unweighted properties. The model is tuned for this particular network by pretending that we only know the size of the network and the in- and out-strength of each node. Since all edges in our model are independent Bernoulli random variables, we can now calculate the properties of the model by computing their expected value. For example, for the average nearest neighbour in-degree, we see

$$\mathbb{E}[k^{in}_{nn,i}] = \mathbb{E}[\frac{\sum_j a_{ji} a_{kj}}{\sum_j a_{ji}}] = \frac{\sum_j \mathbb{E}[a_{ji}]\mathbb{E}[a_{kj}]}{\sum_j \mathbb{E}[a_{ji}]} = \frac{\sum_j p_{ji} p_{kj}}{\sum_j p_{ji}}.$$

In the same way, this can be done for all the directed binary properties outlined in Chapter 2. These expected values can be thought of as the average value of this property for our model, since actually creating a large number of models and averaging out their values for this property will give this expected value. This has also been done for the fitness induced Configuration Model (fiCM) as described in [5], where the connection probability is given by

$$p_{ij} = \frac{\delta x_i y_j}{1 + \delta x_i y_j}.$$

Here $\delta \in \mathbb{R}_{>0}$ is picked such that the expected number of edges, i.e. $\sum_{i,j} p_{ij}$, coincides with the observed number of links in the network. The Configuration Model is a widely known and accepted graph model and therefore a good baseline for comparison of our model. The calculations have been implemented in Matlab of which the code can be found in Appendix B.

In Figure 3.2 we see the plot of the calculated expected values of the degree sequences with respect to the models and the observed degree sequence of the network. The degrees of our model follow more or less the same trend as the observed degrees. However, it is evident that the model performs worse at capturing the trend of the out-degree sequence for nodes with low strength than for nodes with a high strength. This is even more evident in the reconstruction of the unweighted second order properties shown in Figure 3.3, as any mistake in the reconstruction of the first order properties is amplified in the reconstruction of the second order properties. We thus see that the average nearest neighbour out-degree is reconstructed a lot more poorly than the average nearest neighbour in-degree. The clustering coefficient sits somewhere in between, as it is a function of both in-connections and out-connections. However, even though the fiCM performs slightly better than our model, the mistakes made by both models are of very similar fashion.

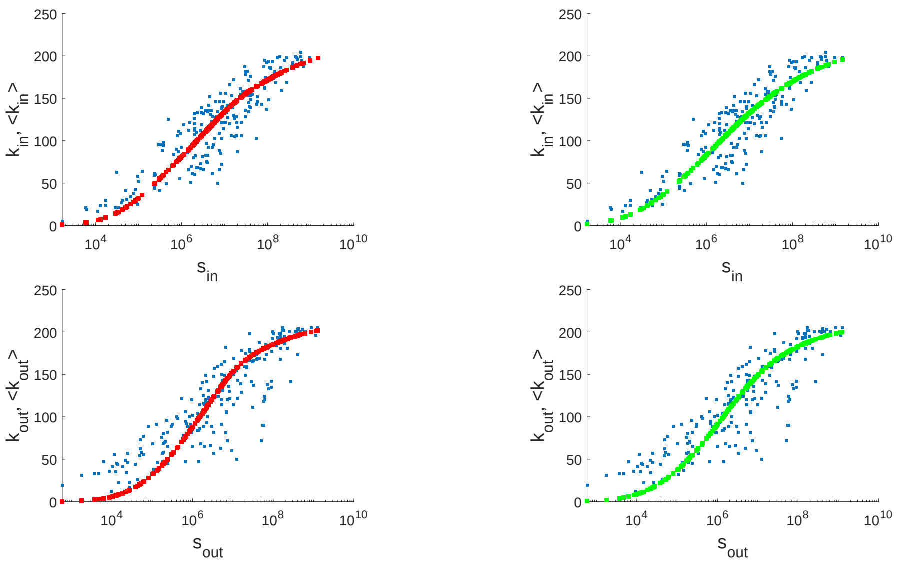

Figure 3.2: Reconstruction of the degree sequences of the trade network. *Scatter plots of node in-degrees k^{in} and out-degrees k^{out} observed in the trade network (blue), the average of our model (red) and the average of the fiCM (green) versus the observed node strengths s^{in} and s^{out} respectively.*

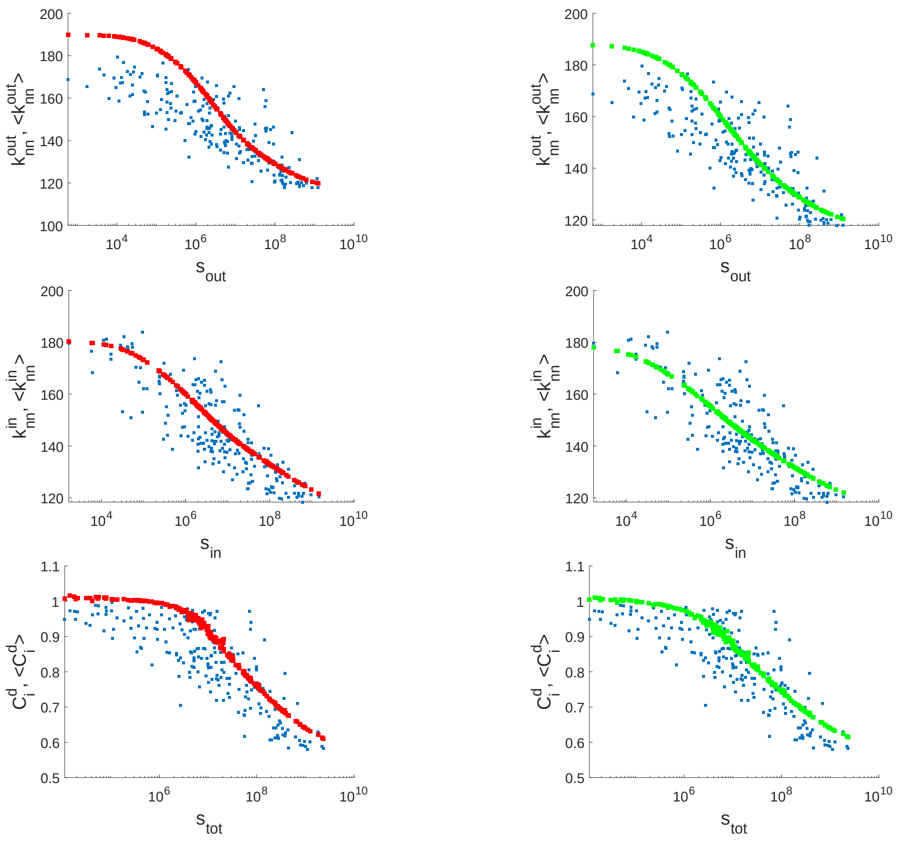

Figure 3.3: Reconstruction of the average nearest neighbour degrees and the clustering coefficients of the trade network. *Scatter plots of node average nearest node out-degree k_{nn}^{out}, average nearest node in-degree k_{nn}^{in}, and clustering coefficient c^d observed in the network (blue), the average of our random graph (red) and the average of the fiCM (green) versus the node out-strength, in -strength and total-strength respectively.*

Chapter 4

Extension to a weighted model

The model presented in the previous chapter only reconstructs the binary topology of networks. This means that it can't be used to analyse the weighted properties outlined in chapter two. Therefore, in this chapter we will extend the binary model to a weighted model.

We again assume that there is some directed real network $G_0 = (V_0, E_0)$, about which limited information is available. The available limited information is the same type of information that was known in the previous chapter. However, on top of an unknown adjacency matrix $A = (a_{ij})$, G_0 now has an unknown weighted adjacency matrix $W = (w_{ij})$. For the modelling of this weighted adjacency matrix, we have two assumptions. The ideas underlying these are the weighted equivalents of the ideas behind the assumptions made in the previous chapter.

1) The weight w_{ij} on the link from node i to node j is a random variable with non-negative probability distribution. This distribution has parameters depending on the out-fitness of i and the in-fitness of j: $w_{ij} \sim F(\cdot|x_i, y_j)$ for some probability distribution F. This holds for individual nodes as well as for groups of nodes.

2) The model is renormalizable. This means that the probability distribution of the weights between two groups must be the same as that of the sum of weights between the nodes of those two groups. Therefore, for any partition $P = \{V_i\}_{i=1}^{n}$ of V_0, equation (3.3) must hold. The weights $w_{V_iV_j}$ and w_{uv} are random variables with probability distribution $F_{V_iV_j}$ and F_{uv} respectively.

We will first show that the second assumption is stronger than the corresponding assumption in the previous chapter. For any partition $P = \{V_i\}_{i=1}^n$ of V_0, we must have, by equation (3.3)

$$\mathbb{P}(w_{V_iV_j} = 0) = \mathbb{P}(\sum_{\substack{u \in V_i \\ v \in V_j}} w_{uv} = 0) = \prod_{\substack{u \in V_i \\ v \in V_j}} \mathbb{P}(w_{uv} = 0).$$

The relation between the adjacency matrix and its weighted counterpart then gives

$$\begin{aligned} \mathbb{P}(a_{V_iV_j} = 0) &= \prod_{\substack{u \in V_i \\ v \in V_j}} \mathbb{P}(a_{uv} = 0) \\ 1 - \mathbb{P}(a_{V_iV_j} = 1) &= \prod_{\substack{u \in V_i \\ v \in V_j}} (1 - \mathbb{P}(a_{uv} = 0) \\ 1 - a_{V_iV_j} &= \prod_{\substack{u \in V_i \\ v \in V_j}} (1 - a_{uv}). \end{aligned}$$

And therefore, equation (3.1) still holds. On top of that, since the distribution of w_{ij} depends on x_i and y_j we must also have

$$\mathbb{P}(w_{ij} = 0) = p(x_i, y_j)$$

for some function $p : \mathbb{R}^2 \to [0, 1]$ for all pairs of nodes $i, j \in V_0$. We can then follow the same proof as that of theorem 1 to find the following proposition:

Proposition 1. *For two nodes $i, j \in V$, we have*

$$\mathbb{P}(w_{ij} = 0) = e^{-\lambda x_i y_j}.$$

The binary part of the model must thus remain the same. Therefore, for any two nodes $i, j \in V_0$, we again denote $p_{ij} = 1 - e^{-\lambda x_i y_j}$. The value of λ is determined in the exact same way as it was in the unweighted model, by imposing

$$\sum_{i,j} 1 - e^{-\lambda x_i y_j} = L.$$

Unless the network is dense and all possible connections are realized, λ will have a finite value. The distribution $F(\cdot|x_i, y_j)$ therefore has a point mass of $e^{-\lambda x_i y_j}$ in 0 and can not be a continuous distribution. With this in mind, there are several discrete distributions that satisfy all three assumptions, such as the Poisson, Binomial or negative Binomial distribution. However, for these distributions imposing the point mass in zero to be a certain value also determines its mean and variance. A discrete distribution is therefore not appropriate, since to make a proper model, it has to be possible to tune separately the magnitude of the weights on links and the density of the model. For example, the reconstruction of the binary topology of a financial network shouldn't be dependent on whether the weights are given in millions or billions of euros. We are therefore looking for a mixed distribution $F(\cdot|f(x_i, y_j))$ that has a density function f that can be written as

$$f(x|x_i, y_j) = (1-p_{ij})\delta_0(x) + p_{ij}g(x|x_i, y_j), \tag{4.1}$$

where g is the density function of a probability distribution $G = G(\cdot|x_i, y_j)$ and δ_0 is the dirac delta function. The latter in fact being a generalized probability density function, used to denote the density function of a discrete probability distribution that is surely zero. The distribution as in equation (4.1) is called a mixed distribution, because with a probability of p_{ij} the random variable follows the probability density function g and with probability $1 - p_{ij}$ it follows the (generalized) probability density function δ_0. This mixed distribution naturally leads to the definition of a two-step model:

1. Draw $z_{ij} \sim$ G$(\cdot|x_i, y_j)$.
2. Let
$$w_{ij} = \begin{cases} z_{ij} & \text{w.p. } p_{ij} \\ 0 & \text{w.p. } 1 - p_{ij} \end{cases}.$$

It is clear that w_{ij} now indeed has the density function as in equation (4.1). So far, we have failed in finding a distribution G such that assumption two is satisfied and we strongly feel that there doesn't exist any. We therefore relax the assumption:

2) The model is semi-renormalizable. That is, for any partition $P = \{V_i\}_{i=1}^n$ of V_0, equation (3.1) holds and

$$p_{V_iV_j} z_{V_iV_j} = \sum_{\substack{u \in V_i \\ v \in V_j}} p_{uv} z_{uv} \quad \text{for} \quad i, j = \{1, \dots, n\}. \tag{4.2}$$

The interpretation is that for two nodes $i, j \in V_0$, $p_{ij}z_{ij}$ is the average value for the weight w_{ij} once z_{ij} is drawn. That is, we have $\mathbb{E}[w_{ij}|z_{ij}] = p_{ij}z_{ij}$. By imposing equation (4.2) we in a sense average out all the possible topologies. Therefore, we have renormalizability on average over all topologies.

To satisfy this assumption, we pick $G(\cdot|x_i, y_j) = \text{Gamma}(\frac{x_i y_j}{\alpha}, \frac{\alpha}{p_{ij}W})$, where $\alpha \in \mathbb{R}_{>0}$ and $W := \sum_i s_i^{out} = \sum_i s_i^{in}$. For details about the Gamma distribution, see Appendix B. The Gamma distribution is not the only distribution that could have been used to satisfy the relaxed constraint, for example any infinitely divisible distribution would work. The advantages of the Gamma distribution are, however, that it has a strictly positive support and an easy to work with probability distribution function. Let us show that the model is now indeed semi-renormalizable. Let $P = \{V_i\}_{i=1}^n$ be an arbitrary partition of V_0. We already showed that equation (3.1) holds. Moreover, for $i, j \in \{1, \dots, n\} : i \neq j$ we see

$$p_{V_iV_j} z_{V_iV_j} \sim \text{Gamma}\left(\frac{x_{V_i} y_{V_j}}{\alpha}, \frac{\alpha}{W}\right)$$

and, because of the properties of the Gamma distribution

$$\sum_{\substack{u \in V_i \\ v \in V_j}} p_{uv} z_{uv} \sim \text{Gamma}\left(\sum_{\substack{u \in V_i \\ v \in V_j}} \frac{x_u y_v}{\alpha}, \frac{\alpha}{W}\right).$$

Equation (3.2) then indeed leads to semi-renormalizability

$$p_{V_iV_j} Z_{V_iV_j} = \sum_{\substack{u \in V_i \\ v \in V_j}} p_{uv} z_{uv} \;\; \text{for} \;\; i, j = \{1, \dots, n\}.$$

A desirable property this model has is that the expected values of the in- and out-strength of any node coincide with their observed value:

$$\mathbb{E}[\sum_j w_{ij}] = \sum_j \mathbb{E}[w_{ij}] = \sum_j p_{ij} \mathbb{E}[z_{ij}] = \sum_j \frac{x_i y_j}{W} = x_i = s_i^{out}.$$

and

$$\mathbb{E}[\sum_j w_{ji}] = \sum_j \mathbb{E}[w_{ji}] = \sum_j p_{ji} \mathbb{E}[z_{ji}] = \sum_j \frac{x_j y_i}{W} = y_i = s_i^{in}.$$

4.1 Analysis on a trading network

We will now analyse how well the weighted properties of a real network are captured by our model. For this purpose, the same trade network as in the previous chapter is used and the computed properties are the directed and weighted properties outlined in Chapter 2.

Since the model follows a two-step procedure, we are now dealing with dependent random variables a_{ij} and w_{ij}. Since it is cumbersome to compute the expected value for functions of dependent random variables, we instead generate 50 instances of our model and calculate the average value of each property. This procedure only relies on the average value of the weights, which is independent of our choice of α. For simplicity we therefore set $\alpha = 1$. Once again, this has been done with Matlab for both our model and the weighted fiCM [5]. The code can be found in Appendix B. For the weighted fiCM, the weights are given by

$$w_{ij} = \begin{cases} \frac{x_i y_j}{W} & \text{w.p. } p_{ij} = \frac{\delta x_i y_j}{1+\delta x_i y_j} \\ 0 & \text{w.p } 1 - p_{ij} \end{cases}.$$

In Figure 4.1 the plot of the computed and observed average nearest neighbour out-strength, average nearest neighbour in-strength and the weighted clustering coefficient can be seen. The quality of the reconstruction very much resembles the reconstruction of the binary properties. Arguably then, errors in the weighted model are the result of errors introduced in the binary part of the model. Once more, our model performs slightly worse than, but very comparable to, the fiCM.

The problem with this method of analysis is that by averaging over 50 instances of the model, we only consider the average weights. This means that this does not validate the choice of the Gamma distribution. According to our model, each non-zero weight w_{ij} observed in the network should be drawn from the Gamma$(\frac{x_i y_j}{\alpha}, \frac{\alpha}{W})$ distribution. Since all of these distributions have different parameters and the network only has one instance of each non-zero weight, we can't analyse the empirical distribution of the weights directly. Instead, we want to find for $i, j \in V_0 : i \neq j$ a function h_{ij} such that $h_{ij}(w_{ij}) \sim$ Gamma$(1, 1)$. We can then compare the empirical distribution over all pairs i, j of $h_{ij}(w_{ij})$ with the Gamma$(1, 1)$ distribution. To find h, we see

$$\begin{aligned} \mathbb{P}(w_{ij} \leq x) &= \mathbb{P}(h(w_{ij}) \leq h(x)) \\ \frac{\gamma(\frac{x_i y_j}{\alpha}, \frac{xW}{\alpha})}{\Gamma(\frac{x_i y_j}{\alpha})} &= \frac{\gamma(1, h(x))}{\Gamma(1)}. \end{aligned}$$

Using the definition of the lower incomplete Gamma function and the

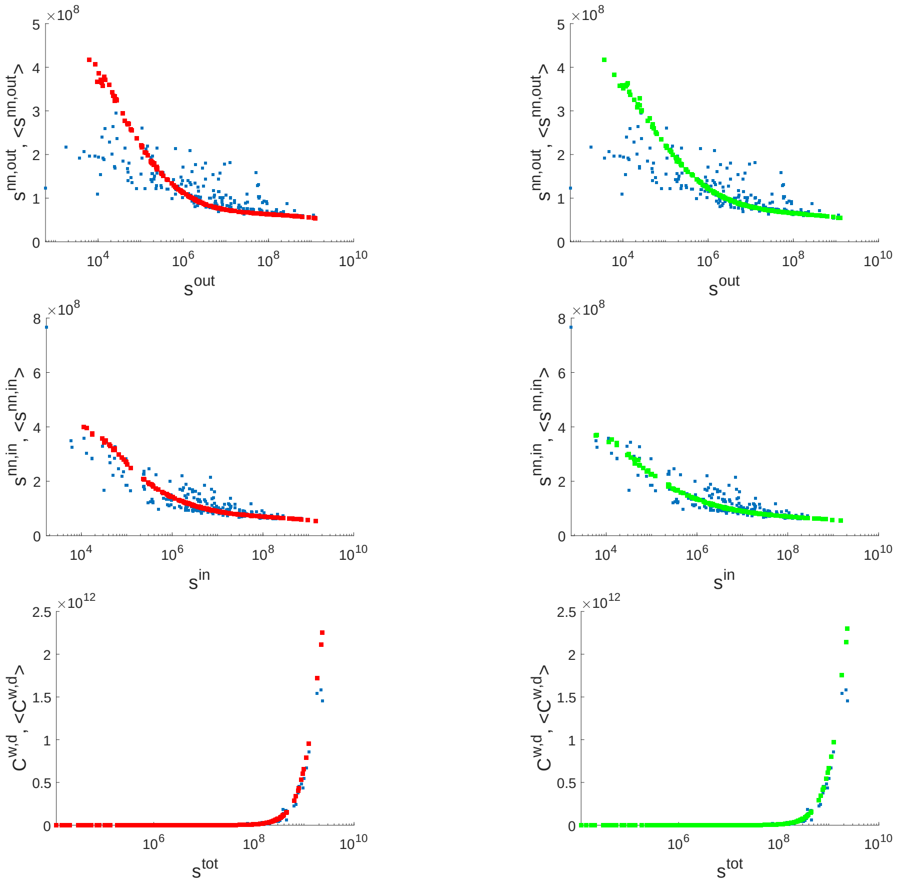

Figure 4.1: Reconstruction of the average nearest neighbour strength and the weighted clustering coefficient. *Scatter plots of node average nearest node out-strength s_{nn}^{out}, average nearest node in-strength s_{nn}^{in}, and weighted clustering coefficient $c^{w,d}$ observed in the network (blue), the average of our random graph (red) and the average of the fiCM (green).*

Gamma function in Appendix B, we find

$$\frac{\gamma(\frac{x_iy_j}{\alpha},\frac{xW}{\alpha})}{\Gamma(\frac{x_iy_j}{\alpha})} = \frac{\displaystyle\int_0^{h(x)} t^0 e^{-t} dt}{\displaystyle\int_0^{\infty} t^0 e^{-t} dt}$$

$$\frac{\gamma(\frac{x_iy_j}{\alpha},\frac{xW}{\alpha})}{\Gamma(\frac{x_iy_j}{\alpha})} = 1 - e^{-h(x)}.$$

We rewrite this equality to find h:

$$\begin{aligned}
e^{-h(x)} &= 1 - \frac{\gamma(\frac{x_iy_j}{\alpha},\frac{xW}{\alpha})}{\Gamma(\frac{x_iy_j}{\alpha})}\\
e^{-h(x)} &= \frac{\Gamma(\frac{x_iy_j}{\alpha}) - \gamma(\frac{x_iy_j}{\alpha},\frac{xW}{\alpha})}{\Gamma(\frac{x_iy_j}{\alpha})}\\
e^{-h(x)} &= \frac{\displaystyle\int_0^{\infty} t^{\frac{x_iy_j}{\alpha}-1} e^{-t} dt - \int_0^{\frac{xW}{\alpha}} t^{\frac{x_iy_j}{\alpha}} e^{-t} dt}{\Gamma(\frac{x_iy_j}{\alpha})}\\
h(x) &= -\ln\left(\frac{\Gamma(\frac{x_iy_j}{\alpha},\frac{xW}{\alpha})}{\Gamma(\frac{x_iy_j}{\alpha})}\right),
\end{aligned}$$

where $\Gamma : \mathbb{R}^2_{>0} \to R_{>0}$ is the upper incomplete Gamma function

$$\Gamma(s,x) = \int_x^{\infty} t^{s-1} e^{-t} dt.$$

The empirical distribution of $h(w)$ in the network is calculated with Matlab (see Appendix B) and shown in figure 4.2. It is evident that the weights of the network are not actually Gamma distributed. Therefore, further research is needed to find the appropriate distribution.

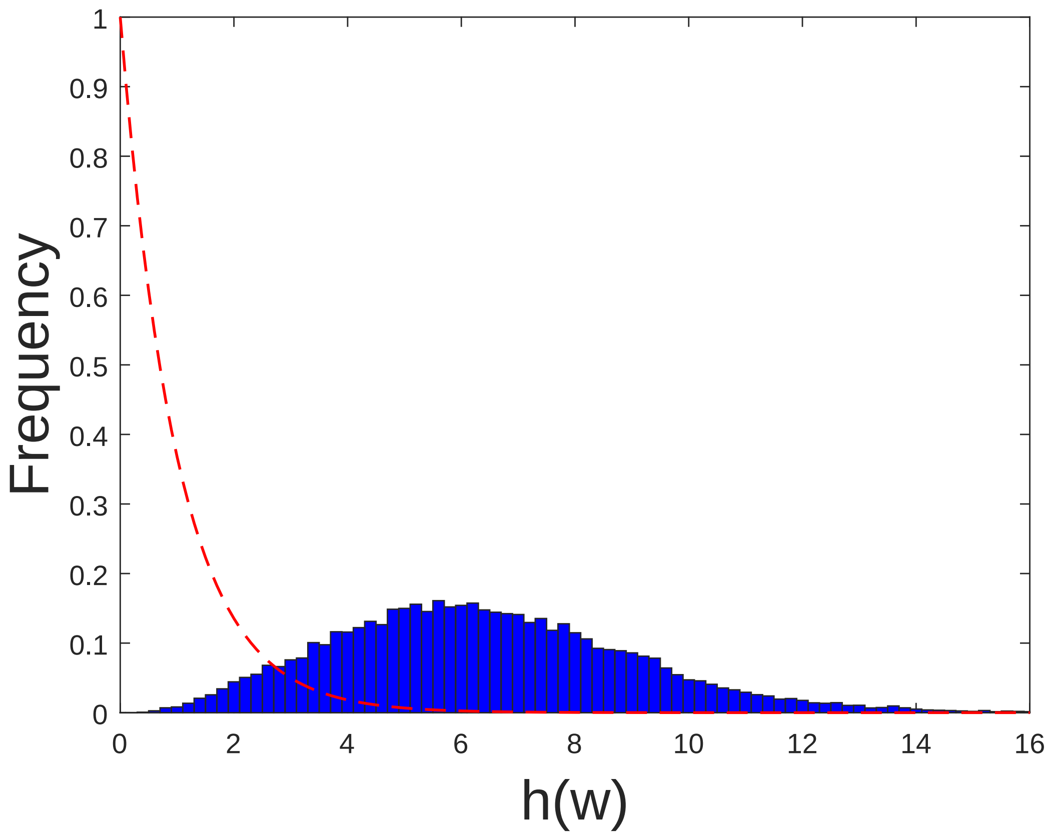

Figure 4.2: The empirical distribution of $h(w)$. *Histogram of the empirical distribution of $h(w)$. For comparison, the dashed line shows the density function of the Gamma$(1,1)$ distribution.*

Chapter 5

Discussion

In this thesis, we studied a binary renormalizable random graph model. This model allows to estimate binary properties of a network, using only intrinsic properties specific to nodes or groups of nodes. We failed in trying to introduce a renormalizable weighted extension of this model and instead proposed a semi-renormalizable weighted extension. In this method, the Gamma distribution is used to assign weights to links, based on the same intrinsic properties the binary model uses. It is shown that this distribution does not accurately represent the distribution of the weights in an empirical network. Future research could focus on finding a distribution that does match with empirical data.

Analysis on a trade network showed that there is a great parallel between the performance of the renormalizable model and that of the fiCM. This is because for the renormalizable model, the Taylor expansion around 0 of the connection probability between nodes i and j as a function λ is

$$p_{ij}(\lambda) = \lambda x_i y_j + O(\lambda^2).$$

For the fiCM, as a function of δ, this is

$$p_{ij}(\delta) = \delta x_i y_j + O(\delta^2).$$

For most empirical networks, the density is very small and therefore λ and δ are too. This means that $p_{ij} \approx \lambda x_i y_j$ and $p_{ij} \approx \delta x_i y_j$ are good approximations for the models. However, as we group together nodes, the density will naturally increase and this approximation becomes invalid. Therefore, the two models will diverge as we analyze renormalizations of a network. They will, however, not diverge indefinitely, since for a very high density, the connection probability will in both models be almost identically equal to one. Future research might therefore focus on analysing the models' performance on different renormalizations of a network. If it is shown that the renormalizable model performs better across renormalizations, it proves to serve as a valuable tool for understanding partially accessible networks.

This would promote the idea that the process in which the networks were created might be self-similar and renormalizable too. It could then be argued that a self-similar and renormalizable creation process would naturally induce scale-freeness and self-similarity on the network, since they can all be thought of as forms of scale invariance. This argument is strengthened by the fact that renormalizability induces the connection probability between nodes i and j of

$$p_{ij}(\lambda) = 1 - e^{-\lambda x_i y_j}.$$

As mentioned earlier, $p_{ij} \approx \lambda x_i y_j$ is often a good approximation. If we now look at the expected degree of nodes, we find

$$\mathbb{E}[k_i] = \sum_{j \neq i} \mathbb{E}[a_{ij}] = \sum_{j \neq i} p_{ij} \approx \lambda x_i \sum_{j \neq i} y_j.$$

Therefore, if x_i has a power law distribution, so too does $\mathbb{E}[k_i]$. Power law distributed fitnesses thus naturally induce scale freeness on our model. Renormalizability therefore provides a possible explanation as to where scale-freeness and self-similarity of complex networks find their origin.

Bibliography

[1] Y. Moreno, R. Pastor-Satorras, and A. Vespignani, *Epidemic outbreaks in complex heterogeneous networks,* The European Physical Journal B **26**, 521 (2002).

[2] S. Markose, S. Giansante, and A. R. Shaghaghi, *'Too interconnected to fail' financial network of US CDS market: Topological fragility and systemic risk,* Journal of Economic Behavior and Organization **83**, 627 (2012).

[3] J. P. Onnela, J. Saramäki, J. Kertész, and K. Kaski, *Intensity and coherence of motifs in weighted complex networks,* Physical Review E - Statistical, Nonlinear, and Soft Matter Physics **71** (2005).

[4] P. E. Mistrulli, *Assessing financial contagion in the interbank market: Maximum entropy versus observed interbank lending patterns,* Banking & Finance **35** (2011).

[5] G. Cimini, T. Squartini, D. Garlaschelli, and A. Gabrielli, *Systemic Risk Analysis on Reconstructed Economic and Financial Networks,* Scientific Reports **5**, 1 (2015).

[6] S. Battiston and M. D. Errico, *Leveraging the network : a Stress - Test Framework based on DebtRank,* arXiv:1503.00621 , 1 (2015).

[7] S. Klavžar and M. J. Nadjafi-Arani, *Wiener index in weighted graphs via unification of Θ*-classes,* European Journal of Combinatorics **36**, 71 (2014).

Appendix A

In this Appendix, we give the proof to a theorem that states that if an additive function is continuous, it is also linear.

Theorem 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an additive function, that is for $x_1, \ldots \in \mathbb{R}^n$, we have*

$$f(\sum_{i=1}^{\infty} x_i) = \sum_{i=1}^{\infty} f(x_i).$$

If f is continuous, it is linear.

Proof. Let $x \in \mathbb{R}$ arbitrarily. By assumption, we have for $m \in \mathbb{N}$:

$$mf(x) = f(x) + f(x) + \ldots + f(x) = f(mx) \tag{1}$$

and

$$f(x) = f(\frac{x}{m}) + f(\frac{x}{m}) \ldots + f(\frac{x}{m}) = mf(\frac{x}{m})$$

from which it follows that

$$\frac{1}{m} f(x) = f(\frac{x}{m}). \tag{2}$$

Let $\frac{p}{q} \in \mathbb{Q}$ arbitrarily, then by (1) and (2) we see

$$f(\frac{p}{q}x) = pf(\frac{x}{q}) = \frac{p}{q} f(x).$$

Since for any $c \in \mathbb{R}$ there is a rational sequence that converges to c, the continuity of f tells us

$$f(cx) = cf(x).$$

Together with the additivity, f is therefore linear. $\square$

Appendix B

In this Appendix, the gamma probability distribution is discussed. The gamma distribution is a family of continuous probability distributions characterized by two parameters. There are multiple parametrizations for this family, but the one adopted in this thesis has a shape parameter $k \in \mathbb{R}_{>0}$ and a scale parameter $\theta \in \mathbb{R}_{>0}$. A gamma-distributed random variable X with shape k and scale θ is then denoted by

$$X \sim \text{Gamma}(k, \theta).$$

The probability density function is

$$f_X(x) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)} \ \text{ for } \ x > 0,$$

where $\Gamma : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ is the Gamma function

$$\Gamma(k) = \int_0^\infty x^{k-1} e^{-x} dx.$$

The probability distribution function is

$$F_X(x) = \frac{\gamma(k, \frac{x}{\theta})}{\Gamma(k)},$$

where $\gamma : \mathbb{R}^2_{>0} \to \mathbb{R}_{>0}$ is the lower incomplete gamma function

$$\gamma(s, x) = \int_0^x t^{s-1} e^{-t} dt.$$

The expectation and variance are given by

$$\mathbb{E}[X] = k\theta \ \text{ and } \ \text{Var}[X] = k\theta^2.$$

Moreover, if $c \in \mathbb{R}_{>0}$ arbitrarily, then

$$cX \sim \text{Gamma}(k, c\theta)$$

and if for $i = 1, \ldots, n$ we have $X_i \sim \text{Gamma}(k_i, \theta)$, then

$$\sum_{i=1}^{n} X_i \sim \text{Gamma}(\sum_{i=1}^{n} k_i, \theta).$$

Appendix C

Matlab is used several times throughout this thesis. All of the code can be found in this Appendix. The code has been written to be as general as possible. When using this code, make sure that W is stored in the matlab workspace as the weighted(!) adjacency matrix for the network you want to analyse. On top of that, the value of λ should be manually saved in the workspace to match the density of your network; this isn't done automatically by the code. The limit of the x-axis of the plots is by default $[0, 10^{10}]$, this will probably need to be adjusted depending on the strenghts of the nodes in the network. The code has not deliberately been optimised, but rather has been designed with sole focus on functionality.

Connection probability function

```
1 %Either line 4 or 6 should be commented out, depending on which
      model is
2 %being used.
3 function p = connection_prob(x,y, lambda)
4     % Fitness induced CM:
5     p = lambda* x *y/(1+lambda*x*y);
6     % Renormalizable model:
7     p = 1 - exp(-lambda * x * y);
8 end
```

Main file unweighted model

```
G                        = digraph(W);
A                        = full(adjacency(G));
[n_nodes,~]              = size(G.Nodes);
[n_edges,~]              = size(G.Edges);
out_fitnesses            = sum(W,2);
in_fitnesses             = sum(W,1);
total_fitnesses          = zeros(n_nodes, 1);
outdegrees               = outdegree(G);
indegrees                = indegree(G);

expected_indegree        = zeros(n_nodes, 1);
expected_outdegree       = zeros(n_nodes, 1);
ANND_in_in               = zeros(n_nodes, 1);
expected_ANND_in         = zeros(n_nodes, 1);
ANND_out_out             = zeros(n_nodes, 1);
expected_ANND_out        = zeros(n_nodes, 1);
clustering               = zeros(n_nodes, 1);
expected_clustering      = zeros(n_nodes, 1);

for i=1:n_nodes
    expected_indegree(i)        = 0;
    expected_outdegree(i)       = 0;
    reciprocal_links            = 0;
    expected_reciprocal_links   = 0;
    denom1                      = 0;
    denom2                      = 0;
    denom3                      = 0;
    total_fitnesses(i)          = in_fitnesses(i) +
       out_fitnesses(i);
    for j=1:n_nodes
        if j ~= i
            reciprocal_links            = reciprocal_links
                        + A(j,i) * A(i,j);
            expected_reciprocal_links   =
               expected_reciprocal_links + connection_prob(
               out_fitnesses(j), in_fitnesses(i), lambda) *
               connection_prob(out_fitnesses(i), in_fitnesses(j
               ), lambda);
            expected_indegree(i)        = expected_indegree(i)
                    + connection_prob(out_fitnesses(j),
               in_fitnesses(i), lambda);
            expected_outdegree(i)       = expected_outdegree(i)
                    + connection_prob(out_fitnesses(i),
               in_fitnesses(j), lambda);
            for k=1:n_nodes
                if k ~= j
                    expected_ANND_in(i)     = expected_ANND_in(
                       i)       + connection_prob(out_fitnesses
                       (j), in_fitnesses(i), lambda) *
                       connection_prob(out_fitnesses(k),
                       in_fitnesses(j), lambda);
                    expected_ANND_out(i)    = expected_ANND_out
                       (i)      + connection_prob(out_fitnesses
                       (i), in_fitnesses(j), lambda) *
                       connection_prob(out_fitnesses(j),
                       in_fitnesses(k), lambda);
                    expected_clustering(i)  =
                       expected_clustering(i)    + (
                       connection_prob(out_fitnesses(j),
                       in_fitnesses(i), lambda) +
                       connection_prob(out_fitnesses(i),
                       in_fitnesses(j), lambda)) * (
                       connection_prob(out_fitnesses(j),
                       in_fitnesses(k), lambda) +
                       connection_prob(out_fitnesses(k),
                       in_fitnesses(j), lambda)) * (
                       connection_prob(out_fitnesses(k),
                       in_fitnesses(i), lambda) +
                       connection_prob(out_fitnesses(i),
                       in_fitnesses(k), lambda));

                    denom1          = denom1 + connection_prob(
                       out_fitnesses(j), in_fitnesses(i),
                       lambda) * connection_prob(out_fitnesses(
                       k), in_fitnesses(i), lambda);
                    denom2          = denom2 + connection_prob(
                       out_fitnesses(i), in_fitnesses(j),
                       lambda) * connection_prob(out_fitnesses(
                       i), in_fitnesses(k), lambda);
                    denom3          = denom3 + connection_prob(
                       out_fitnesses(j), in_fitnesses(i),
                           lambda) * connection_prob(out_fitnesses(
                           i), in_fitnesses(k), lambda);

                        ANND_in_in(i)   = ANND_in_in(i)     + A(j,i
                           ) * A(k,j);
                        ANND_out_out(i) = ANND_out_out(i)   + A(i,j
                           ) * A(k,j);
                        clustering(i)   = clustering(i)     + (A(j,
                           i) + A(i,j)) * (A(j,k) + A(k,j)) * (A(k,
                           i) + A(i,k));
                    end
                end
            end
        end
        expected_ANND_in(i)     = expected_ANND_in(i)   /
           expected_indegree(i);
        ANND_in_in(i)           = ANND_in_in(i)         / indegrees
           (i);
        expected_ANND_out(i)    = expected_ANND_out(i)  /
           expected_outdegree(i);
        ANND_out_out(i)         = ANND_out_out(i)       /
           outdegrees(i);

        denominator             = denom1 + denom2 + 2 * (denom3 -
           expected_reciprocal_links);
        total_degree            = indegrees(i) + outdegrees(i);
        expected_clustering(i)  = 0.5 * expected_clustering(i) /
           denominator   ;
        clustering(i)           = 0.5 * clustering(i) / (
           total_degree*(total_degree - 1) - 2*reciprocal_links);
end

sz=20;

% Plot of indegrees
y = indegree(G);
x = in_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log', 'FontSize', 15)
ylabel('k_{in}, <k_{in}>', 'FontSize', 20)
xlabel('s_{in}', 'FontSize', 20)
```

```
xlim([0, 10^10])

hold on
scatter(x,expected_indegree, 'square', 'red', 'filled');
hold off

figure()
%Plot of outdegrees
y = outdegrees;
x = out_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log', 'FontSize', 15)
ylabel('k_{out}, <k_{out}>', 'FontSize', 20)
xlabel('s_{out}', 'FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,expected_outdegree, 'square', 'red', 'filled');
hold off

figure()
% Plot of ANND_in
y = ANND_in_in;
x = in_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log','FontSize', 15)
ylabel('k_{nn}^{in}, <k_{nn}^{in}>','FontSize', 20)
xlabel('s_{in}','FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,expected_ANND_in, 'square', 'red', 'filled');
hold off

figure()
% Plot of ANND_out
y = ANND_out_out;
x = out_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log','FontSize', 15)
ylabel('k_{nn}^{out}, <k_{nn}^{out}>','FontSize', 20)
```

```
xlabel('s_{out}','FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,expected_ANND_out, 'square', 'red', 'filled');
hold off

figure()
% Plot of clustering vs total degree
y = clustering;
x = total_fitnesses;
sz= 20;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log','FontSize', 15)
ylabel('C_i^{d}, <C_i^{d}>','FontSize', 20)
xlabel('s_{tot}','FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,expected_clustering, 'square', 'red', 'filled');
hold off
```

Main file weighted model

```
G                        = digraph(W);
A                        = full(adjacency(G));
max_W                    = max(W(:));
[n_nodes,~]              = size(G.Nodes);
[n_edges,~]              = size(G.Edges);
out_fitnesses            = sum(W,2);
in_fitnesses             = sum(W,1);
total_fitnesses          = zeros(n_nodes, 1);
total_weight             = sum(in_fitnesses);
outdegrees               = outdegree(G);
indegrees                = indegree(G);

iterations               = 50;

ANNS_in                  = zeros(n_nodes, 1);
```

```
calculated_ANNS_in      = zeros(n_nodes, iterations);
ANNS_out                = zeros(n_nodes, 1);
calculated_ANNS_out     = zeros(n_nodes, iterations);
clustering              = zeros(n_nodes, 1);
calculated_clustering   = zeros(n_nodes, iterations);

for n=1:iterations
    disp(n)
    calculated_W        = rand(n_nodes);
    for i=1:n_nodes
        for j=1:n_nodes
            if j~=i
                if  calculated_W(i,j) <= connection_prob(out_fitnesses(i), in_fitnesses(j), lambda)
                    prob    = connection_prob(out_fitnesses(i), in_fitnesses(j), lambda);
                    pd      = makedist('Gamma', 'a', out_fitnesses(i)*in_fitnesses(j), 'b', 1/(prob*total_weight));
                    calculated_W(i,j) = random(pd);
                else
                    calculated_W(i,j) = 0;
                end
            else
                calculated_W(i,j)=0;
            end
        end
    end

    calculated_G                = digraph(calculated_W);
    calculated_A                = full(adjacency(calculated_G));
    calculated_max_W            = max(calculated_W(:));
    calculated_indegrees        = indegree(calculated_G);
    calculated_outdegrees       = outdegree(calculated_G);
    calculated_out_fitnesses    = sum(calculated_W, 2);
    calculated_in_fitnesses     = sum(calculated_W, 1);

    for i=1:n_nodes
        reciprocal_links                = 0;
        calculated_reciprocal_links     = 0;
        total_fitnesses(i)              = in_fitnesses(i) + out_fitnesses(i);
        for j=1:n_nodes
            if j ~= i
                reciprocal_links            = reciprocal_links + A(j,i) * A(i,j);
                calculated_reciprocal_links = calculated_reciprocal_links   + calculated_A(j,i) * calculated_A(i,j);
                for k=1:n_nodes
                    if k ~= j
                        ANNS_in(i)                      = ANNS_in(i)                + A(j,i) * W(k,j);
                        calculated_ANNS_in(i,n)         = calculated_ANNS_in(i,n)   + calculated_A(j,i) * calculated_W(k,j);
                        ANNS_out(i)                     = ANNS_out(i)    + A(i,j) * W(k,j);
                        calculated_ANNS_out(i,n)        = calculated_ANNS_out(i,n)  + calculated_A(i,j) * calculated_W(j,k);
                        clustering(i)                   = clustering(i) + (W(j,i) + W(i,j)) * (W(j,k) + W(k,j)) * (W(k,i) + W(i,k)) / max_W;
                        calculated_clustering(i,n)      = calculated_clustering(i,n) + (calculated_W(j,i) + calculated_W(i,j)) * (calculated_W(j,k) + calculated_W(k,j)) * (calculated_W(k,i) + calculated_W(i,k)) / calculated_max_W;
                    end
                end
            end
        end
        calculated_ANNS_in(i,n) = calculated_ANNS_in(i,n)   / calculated_indegrees(i);
        ANNS_in(i)              = ANNS_in(i)                / indegrees(i);
        calculated_ANNS_out(i,n)= calculated_ANNS_out(i,n)  / calculated_outdegrees(i);
        ANNS_out(i)             = ANNS_out(i)               / outdegrees(i);

        total_degree                = indegrees(i) + outdegrees(i);
        calculated_total_degree     = calculated_indegrees(i) + calculated_outdegrees(i);
        clustering(i)               = 0.5 * clustering(i)                 / (total_degree*(total_degree - 1) - 2*reciprocal_links);
        calculated_clustering(i,n)  = 0.5 * calculated_clustering(i,n)  / ( calculated_total_degree*(calculated_total_degree - 1) - 2*calculated_reciprocal_links);
    end
end
calculated_ANNS_in      = sum(calculated_ANNS_in,   2) / iterations;
calculated_ANNS_out     = sum(calculated_ANNS_out,  2) / iterations;
calculated_clustering   = sum(calculated_clustering,2) / iterations;
sz=20;

% Plot of ANNS_in
y = ANNS_in;
x = in_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log', 'FontSize', 15)
ylabel('s^{nn,in}, <s^{nn,in}>', 'FontSize', 20)
xlabel('s^{in}', 'FontSize', 20)
xlim([0, 10^10])

hold on
```

```
scatter(x,calculated_ANNS_in, 'square', 'red', 'filled');
hold off

figure()
% Plot of ANNS_out
y = ANNS_out;
x = out_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log', 'FontSize', 15)
ylabel('s^{nn,out}, <s^{nn,out}>', 'FontSize', 20)
xlabel('s^{tot}', 'FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,calculated_ANNS_out, 'square', 'red', 'filled');
hold off

figure()
% Plot of clustering
y = clustering;
x = total_fitnesses;
scatter(x,y,sz, 'square', 'filled');
set(gca,'xscale','log', 'FontSize', 15)
ylabel('C^{w,d}, <C^{w,d}>', 'FontSize', 20)
xlabel('s^{tot}', 'FontSize', 20)
xlim([0, 10^10])

hold on
scatter(x,calculated_clustering, 'square', 'red', 'filled');
hold off
```

Distribution of weights file

```
G                       = digraph(W);
[n_nodes,~]             = size(G.Nodes);
out_fitnesses           = sum(W,2);
in_fitnesses            = sum(W,1);
total_weight            = sum(in_fitnesses);
max_out                 = max(out_fitnesses);
```

```
max_in                  = max(in_fitnesses);
renorm_W                = [];

for i=1:n_nodes
    for j=1:n_nodes
        if i~=j
            if W(i,j) ~= 0
                constant = 1e18;
                prob        = connection_prob(out_fitnesses(i),
                     in_fitnesses(j), lambda);
                renorm_W = [renorm_W, -log(gammainc((W(i,j)*
                   prob*total_weight)/constant, (out_fitnesses(
                   i) * in_fitnesses(j))/constant, 'upper'))];
            end
        end
    end
end

binWidth     = 0.2;
binCtrs      = 0:binWidth:20;
n            = length(renorm_W);
counts       = hist(renorm_W, binCtrs);
prob         = counts / (n*binWidth);
H            = bar(binCtrs,prob,'hist');
set(H,'facecolor','blue');

hold on;
x = 0:.1:20;
y = gampdf(x,1,1);
plot(x,y,'r--','linewidth',1);
ylabel('Frequency', 'FontSize', 20);
xlabel('h(w)','FontSize', 20);
xlim([0, 16])
```