University of Strathclyde

Department of Mathematics and Statistics

Computational Tools for Complex Networks

Alan J. Taylor

A thesis presented in fulfilment of the requirements
for the degree of Doctor of Philosophy.

December 2009

Dedication

This thesis is dedicated to my mum, Irene Mary Taylor (1949-2004), for all her love and support, and for being the greatest role model I ever had.

Acknowledgements

I would like to thank my supervisor Professor Desmond J. Higham for his patience and support over the duration of my studies. As well as being a wonderful mentor, he has been a constant source of encouragement. Without his guidance, writing this thesis would have been a far more intimidating prospect. I would also like to thank my colleagues Jonathan, Keith and Ernesto, whose invaluable contributions, advice and encouragement were a great help to me.

My family have been wonderfully supportive of me and I would like to thank my dad for letting me stay in Kilmarnock during the last year of my studies and not complaining too much about the subsequent electricity bills. To my flatmates Pete and Louise I'd like to offer my sincere thanks and heartfelt apologies. They had to tolerate some hairy moments of angst and frustration but were always there when I needed someone to talk to (or drink with).

During my time in the department I've had a great many officemates and I'd like to thank them all for making the department a happy place to be. Special mention must go to the longest suffering officemates; Gavin for his invaluable help in getting over the LaTeX learning curve and Adam for his refreshing optimism. Thanks also to Maxim for passing on his wisdom on many occasions and for introducing me to the music of Tom Waits and Nick Cave. Aside from my officemates I would like to thank all the friends I made during my stay in the department for making it such a happy working environment.

Finally I would like to thank Ann for making the last three years the most enjoyable of my life. She gave me confidence in my abilities and was always able to cheer me up and take my mind off work. I hope that some day I'll be able to repay all of her patience and support.

Abstract

The field of network science has experienced increased interest in the past decade due to a combination of new analytical insights, improved computational performance and the availability of datasets from a broad spectrum of applications. Many tasks involving networks lend themselves to interpretation as problems in linear algebra. One such area of study is the systematic decomposition of large networks into smaller substructures, analysis of which may aid the understanding of the global properties of the network.

The aims of this thesis are twofold. Firstly, we provide two tools to aid in the formulation and testing of applications for processing complex networks. The first of these is a MATLAB toolbox for generating and processing instances of various random graph models for use as sparse test matrices. The second is a network repository that provides an accessible set of networks from a variety of real world application areas. Secondly, we propose and discuss two procedures designed to identify a particular pattern of connectivity within directed networks; approximate bipartivity. The first of these is based on the singular value decomposition of the adjacency matrix. The second is a mapping that produces a symmetric real-valued matrix where the entries give a measure of the similarity of nodes. Both methods are tested extensively and applied to directed networks from biology and sociology. Finally, we make a comparison of the two methods before summarizing our findings and suggesting possible directions for future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Outline of Thesis

The field of complex networks is currently an area of intensive research due in part to its applicability to data from a variety of sources. Complex networks are found for example in biology [55, 85, 92], finance [9, 16], sociology [56, 71, 74] and technology [1,33]. A convenient representation of complex networks is as an adjacency matrix and this allows the application of concepts from linear algebra to problems arising in these particular fields of study. The aim of this thesis is to provide some computational tools to aid the testing of methods for complex networks and to develop and explore in detail two approaches to a particular problem - that of identifying approximately directed bipartite substructures.

In Chapter 2 we introduce the topic of network science and briefly discuss the key ideas that gave impetus to the study of complex networks in the past decade. We outline the measurements that are commonly calculated on networks to attempt to describe or classify them. A description of the notation and pictorial representations of networks used throughout this thesis is also given.

Much of the early work in the topic of network science was concerned with various probabilistic models for producing networks and the classical models by Erdös and Rényi and Gilbert are discussed in Chapter 3. More recently, there has been interest in probabilistic models that aim to capture key properties of real life networks and these are also discussed here.

One area in which random graph models are employed is as test matrices for numerical algorithms. To this end, the MATLAB toolbox CONTEST (CONtrollable TEST matrices) was developed. The models and utilities implemented in CONTEST are described in Chapter 4 and results of numerical tests carried out on the models are also presented.

As part of the Institute of Advanced Studies' programme in network science, a repository of datasets from real-world applications was created with a view to becoming a set of test networks for participants' ideas and algorithms. In Chapter 5 each of the twelve networks in NESSIE (Network Example Source Supporting Innovative Experimentation) is described and plots of standard measures on networks are given.

The remainder of the original work in this thesis is concerned with methods for the discovery of approximate directed bipartite subgroups within networks. Our first approach to this problem involves the singular value decomposition (SVD) of the adjacency matrix and is presented in Chapter 6. This work is an extension to the directed case of the lock-and key algorithm described in [66]. The procedure is applied to test data before being used to investigate a network of genes relating to the oncogene p53.

In order to test a second procedure for identifying directed bipartite substructure, we first present a random graph model in Chapter 7 which aims to produce directed networks that match a given target distribution of in and out degrees. Two additional extensions of this model are presented and these are also used in the testing.

The second procedure for discovering directed bipartite subgraphs concerns counts of *alternating walks* on a network. In Chapter 8 we introduce the concept of an alternating walk and argue that by assuming a directed bipartite substructure and considering counts of even and odd length alternating walks, we can obtain a measure of 'similarity' for two nodes and hence a real-valued symmetric matrix. We show that clustering algorithms applied to this mapped matrix can reveal directed bipartite substructures within the original network. This procedure is tested on synthetic networks before being applied to a dataset from neuroscience.

In Chapter 9 we present a brief comparison of the two algorithms for detecting approximate bipartite substructures, based on the genetic and neuroscience networks. Finally we summarise the findings of this work and identify areas of future interest in Chapter 10.

1.2 Publications and Presentations

Much of the material presented in Chapters 3 and 4 has appeared in the article

- *CONTEST: A controllable test matrix toolbox for MATLAB*, A. Taylor and D. J. Higham, ACM Tran. Math. Software, **39**, 2009, 26:1–26:17.

The material presented in Chapter 5 will appear as an invited chapter co-authored with D. J. Higham in the Springer publication *Network Science: Complexity in Nature and Technology.*

The material presented in Chapter 6 has been written as a technical report and was first presented as a poster entitled "Discovering directed bipartite subgraphs" at NetSci 08, Norwich Biosciences Institutes, June 2008. It has been submitted for publication as a paper co-authored with D. J. Higham and J. K. Vass.

The material presented in Chapters 7 and 8 has also been written as a technical report and was first given as a presentation entitled “Mapping directed networks” at the 23$^{\mathrm{rd}}$ Biennial Conference on Numerical Analysis, University of Strathclyde, June 2009. It has been submitted for publication as a paper co-authored with J. J. Crofts, E. Estrada and D. J. Higham.

Chapter 2

Background Material

2.1 Complex Networks

In the past decade, the study of complex networks has undergone a period of renewed interest due largely to the seminal papers by Watts and Strogatz [90] and Barabási and Albert [4]. Network science has its roots in graph theory, a subject thought to have originated in Euler's seminal paper of 1736 [31] where the nonexistence of an *Eulerian circuit* in the system of bridges in Königsberg was proven. In the late 1950s, the fields of graph theory and probability theory were combined in the works by Erdös and Rényi [25] and Gilbert [35].

Taking inspiration from the psychologist Stanley Milgram's work on chains of acquaintances [63], Watts and Strogatz proposed a pseudo-random graph model which lay between a regular lattice and a random graph that more closely matched qualitative properties of real world networks than either extreme case [90]. Their work on the so-called "small world" phenomenon prompted further research into complex networks, most notably Barabási and Albert's paper on scale-free degree distributions [4]. As a direct result of these two influential papers, coupled with improved computer processing power and the volume of readily available datasets, the field of network science has undergone a period of intense research.

Put simply, a network is a collection of objects (called *nodes* or *vertices*) and relationships between these objects (called *edges*). Networks may be directed (there is a direction associated with each edge, i.e. the existence of an edge from node i to node j does not guarantee the existence of an edge from node j to node i) or undirected (i.e. edges can be thought of as bidirectional). The network of scientific collaborations, for instance, is undirected since a collaboration between X and Y is equivalent to a collaboration between Y and X. In contrast, the world wide web is a directed network since a hyperlink from page A to page B may not have a reciprocal hyperlink from page B to page A.

Similarly, networks can be classified into weighted and unweighted categories. In a weighted network, some quantity is associated with each edge whereas in an unweighted network connections are binary: an edge is either present or absent. An example of a weighted network is given by distances between cities, where the cities are nodes and the shortest distance between cities by road are the edges. An unweighted example is the network of co-starring actors. If two actors have appeared in a film together, they are linked by an edge; if they have not appeared together, there is no edge.

Some examples of real world complex networks are given below[1]. Figure 2.1 shows a high school dating network [8] where nodes are students and an edge exists between two nodes if they have been romantically involved. Figure 2.2 shows a network of protein-protein interactions in yeast [61], where each node is a protein and there is an edge between two nodes if they are observed to interact with each other. Finally, Figure 2.3 shows a network drawn from the novel *Les Miserables* [68] where each node is a character and two characters are linked by an edge if they appear in a scene together.

[1]Figures taken from `http://www-personal.umich.edu/~mejn/networks/`

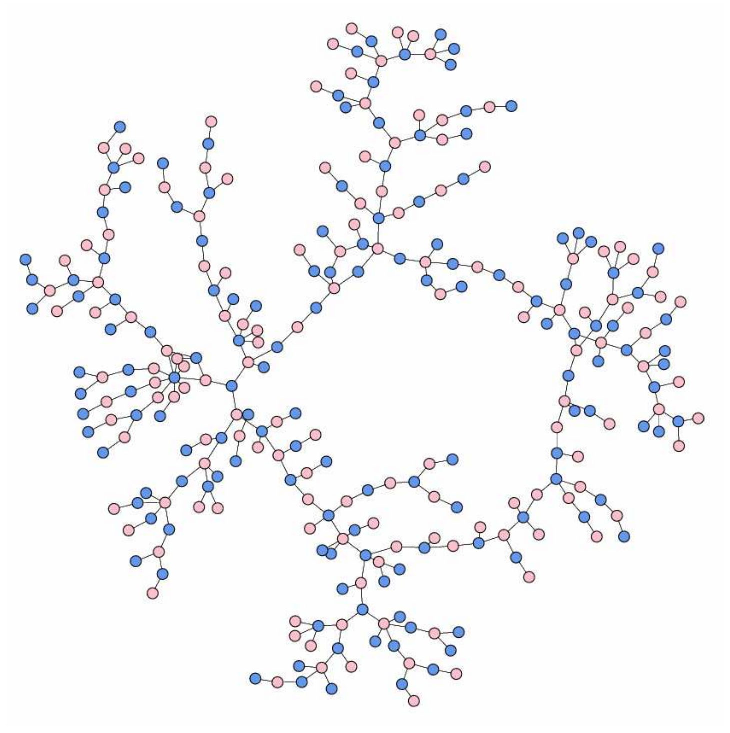


Figure 2.1: High school dating network. Blue and pink nodes represent male and female students respectively.

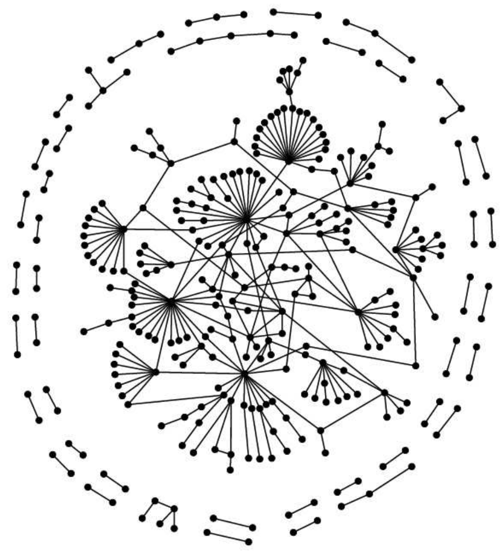


Figure 2.2: Protein-protein interaction network in yeast.

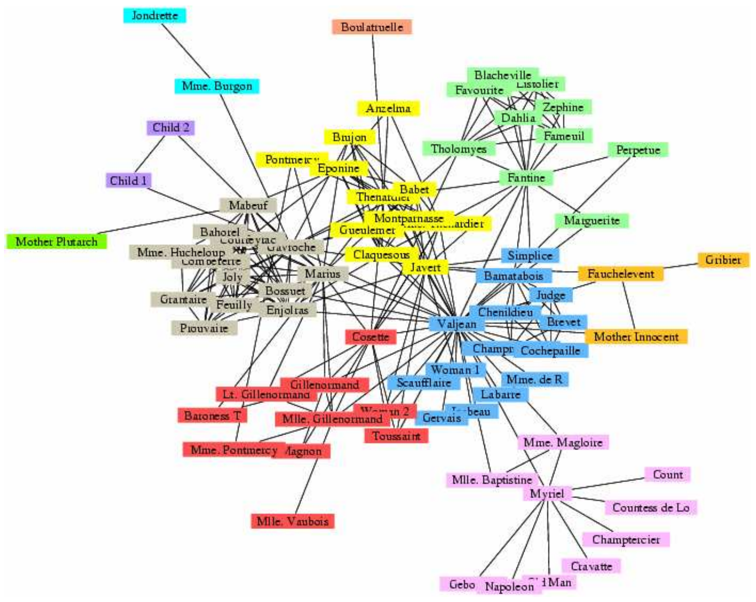


Figure 2.3: Social network of characters in the novel *Les Miserables*.

2.2 Notation

Many operations or processes on networks reduce to problems in linear algebra and as such it is convenient to write a network in terms of its adjacency matrix. For a graph $G(V, E)$ where V is the set of vertices and E a list of edges, the adjacency matrix A is a square matrix consisting of one row/column per vertex in V with a 1 in the i, j position if there is an edge between vertices i and j in the set E and 0 otherwise. An undirected network must then correspond to a symmetric adjacency matrix since an edge from i to j can be thought to have a reciprocal edge from j to i, therefore $A(i, j) = A(j, i) = 1$. In the case of a weighted network, we think of a real valued matrix W where the $W(i, j)$ entry corresponds to the weight on the edge from i to j.

In certain cases it may be appropriate to permit networks with self links, that is to say nodes that are connected to themselves. One instance where such structures may occur is in food webs, where species are nodes and an edge represents a predator-prey relationship between species. A self-link in a food web is indicative of cannibalism within a species. Self links are represented by nonzeros in the main diagonal of an adjacency matrix. In this work we exclusively consider networks without self-links.

Figure 2.4 shows a simple network which will yield the adjacency matrix

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

Throughout this thesis we will use matrix notation when referring to the edges in a network. The nature of much of the work is such that it is more convenient to represent results pictorially in the form of a matrix spy-plot reordering than to list nodes and edges. The techniques we will use to present such results are described in the next section.

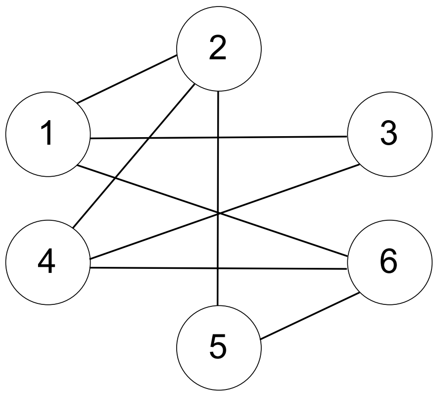


Figure 2.4: An undirected network.

2.3 Visualisation of Networks

It is often necessary to present results in complex networks pictorially, particularly in applications such as clustering or reordering algorithms. Rather than attempting to find an optimal two dimensional placement of nodes in a network, we will adopt the convention of using MATLAB's `spy` function to produce a spy-plot of an adjacency matrix. This is a pictorial representation of a matrix where nonzeros are represented by dots. Since we deal mainly with binary matrices, no information is lost in such a representation. This form of representation is particularly useful when we attempt to qualify the success of a given reordering on an adjacency matrix. By applying a permutation to the rows and columns, we can see whether a particular node reordering reveals any patterns of connectivity or cliques. In Figure 2.5 a spy-plot of an adjacency matrix is shown next to the spyplot of the same adjacency matrix after the rows and columns have been reordered according to the first eigenvector. We see that the reordering has revealed structures that were hidden by the arbitrary initial ordering. Developing reordering strategies is a major theme in this thesis.

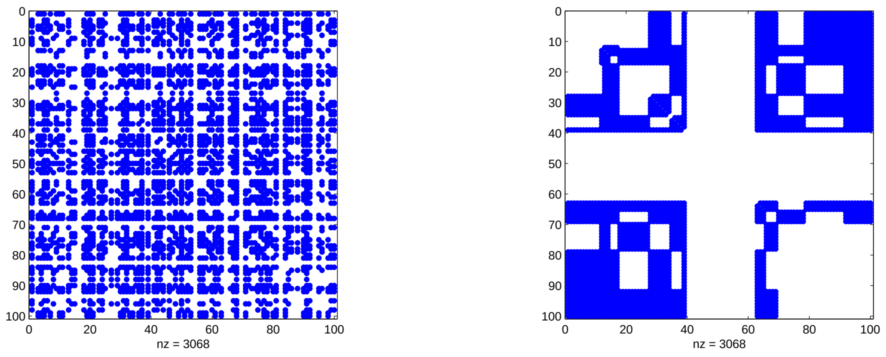


Figure 2.5: Spyplot of an adjacency matrix before and after reordering.

In the instances where we deal with a real valued or weighted network, we will use a heat map as a visualisation tool, where matrix entries are assigned a point on a colour scale according to their value, for instance a large positive entry may be red whereas a large negative entry may be blue. For such matrices there may be few or no zero-entries, so a spy-plot will not be a helpful representation of the data.

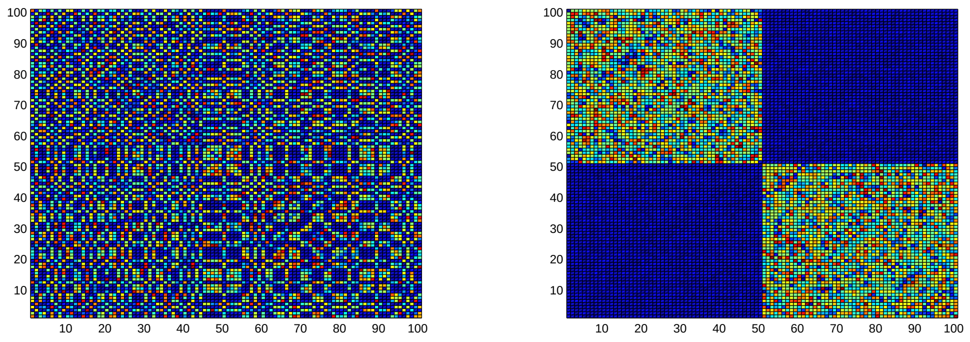


Figure 2.6: Heat map of a weighted adjacency matrix before and after reordering.

2.4 Measures on Networks

2.4.1 Degree Distribution

The *degree* of a node is defined as the number of edges incident with it. In the case of unweighted graphs where repeated edges or self links are disallowed, the degree of a node is equal to the number of neighbours it has, that is, the number of nodes that can be reached by traversing a single edge. In the case of directed networks, we consider two quantities per node, the *in-degree* and *out-degree*. In this instance, an edge from node i to node j contributes to the out-degree of i and the in-degree of j. Degree distributions are easily computable by considering an adjacency matrix A. The in-degree of node i simply corresponds to the sum of entries in the i^{th} column. Similarly, the out-degree of node i may be computed by summing entries in the i^{th} row of A. In the case of an undirected network, the in-degree and out-degree are necessarily equal so either the row or column sum of the adjacency matrix may be used. Figure 2.7 shows a spy-plot of an adjacency matrix corresponding to an undirected network together with a log-log plot of the degree distribution. For a given degree k, $P(k)$ is defined as the number of nodes having degree k.

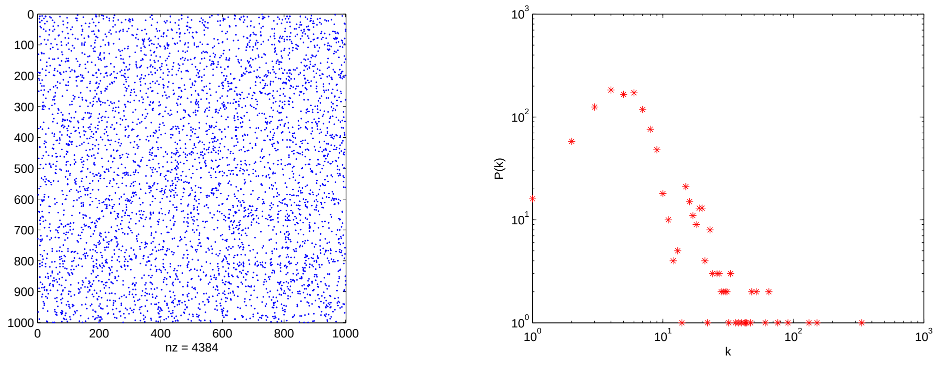


Figure 2.7: Geometric random network and degree distribution.

2.4.1.1 Scale-free Degree Distributions

Degree distributions are of interest when considering real world networks as nodes of high degree may correspond to some kind of hub through which many paths in the network must flow [5]. A network is said to be *scale free* if its degree distribution is fat-tailed, that is to say that for a given degree k, the number of nodes with degree equal to k is given by $p(k) \approx k^{-\gamma}$ for some positive γ. In [4], Barabási and Albert showed that scale free degree distributions can be found in networks as diverse as actor collaboration data, the world wide web and the electrical power grid of the western United States. Figure 2.8 shows a spy-plot of a synthetic network together with a log-log plot of its degree distribution. From the spy-plot, it is clear that the first few nodes have many connections compared to the remainder of the nodes. This is mirrored in the log-log plot, where we see that many nodes have a few connections whereas comparatively few nodes have many connections. The points on this plot also appear to fall roughly in a straight line with negative gradient, i.e. they follow an inverse power law.

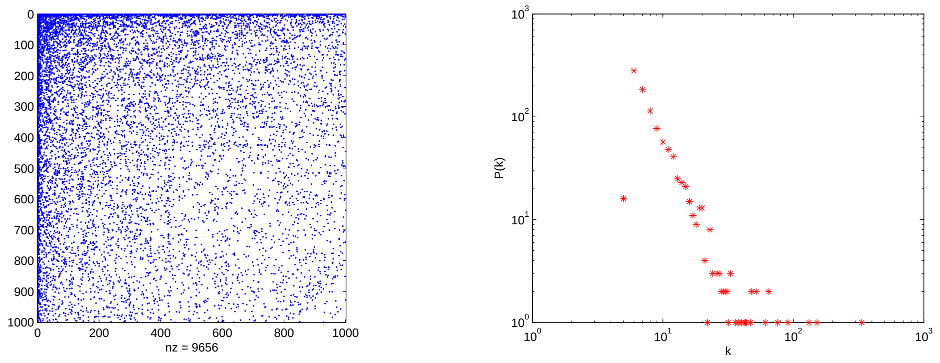


Figure 2.8: Scale-free network and degree distribution.

There has been considerable debate in the literature over scale-free networks, specifically whether networks in nature tend to be scale free or whether this observed phenomenon is an artefact of the processes by which data is sampled [43, 81]. Owing to their structure and the existence of “hub” nodes, scale-free networks tend to be resilient to failures which are random in nature, such as node or edge removal, but are vulnerable to targeted attacks, i.e. the removal of hubs. This property allows meaningful questions to be posed concerning real world networks where a scale-free structure is present. Protein regulatory networks, the network of internet routers and sexual contact networks have been observed to have scale-free degree distributions [4, 5], allowing investigation into processes such as transmission of disease or the cascade effects of power failure.

2.4.2 Clustering Coefficient

The *clustering coefficient* (or *curvature*) of a node is a measure of the connectivity of its neighbours. Put simply, it is the ratio of the number of existing closed walks of length 3 from a node to itself to the number of such walks that could potentially exist. By a walk we mean an alternating sequence of nodes and edges such that each node is incident to the edges that preceed and follow it in the sequence. If the first and last nodes in the sequence are the same, we refer to this as a *closed walk*. Similarly, when we refer to a *path*, we are speaking about an *open walk*. The clustering coefficient c_i of a vertex v_i is given by

$$c_i = \frac{2|\sum_{j \neq k \in N_i} a_{jk}|}{k_i(k_i - 1)}$$

where N_i is the set of neighbours of i, k_i is the degree of i and

$$a_{jk} = \begin{cases} 1 & \text{if there is an edge connecting } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

Hence, a node whose neighbours are completely connected will have a clustering coefficient of 1. In real world networks, the clustering coefficient is often seen as an indication of how well a network is organised into cliques. This is particularly apparent in social networks where an individual's acquaintances are likely to be acquaintances of each other. Figure 2.9 shows a spy-plot of an adjacency matrix together with a plot of the clustering coefficient of each node. The clustering coefficients have been reordered so that their distribution is more apparent.

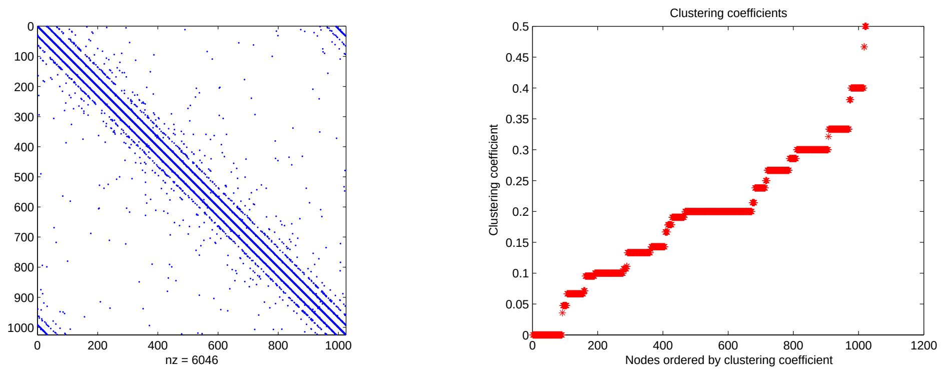
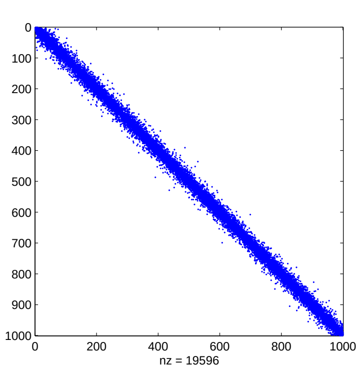


Figure 2.9: Network spy-plot and clustering coefficients.

An alternative (and non-equivalent) measure has been proposed for the global clustering coefficient of a network; this is defined as three times the number of triangles in the network divided by the number of connected triples of vertices.

2.4.3 Pathlength

A quantity that gives insight into the connectivity of a network yet is straightforward to compute is the *shortest pathlength* between pairs of nodes. The shortest pathlength between nodes i and j is the fewest number of edges that need to be traversed in a path from i to j. If j is a neighbour of i, then the shortest pathlength between i and j is 1. Longer paths may exist but the shortest path is typically the most useful to compute. Pathlengths have an intuitive analog in terms of adjacency matrices. The unweighted adjacency matrix A counts all the paths of length 1 while A^2 counts all paths of length 2. In general the n^{th} power of the adjacency matrix A^n counts all walks of length n. By using Dijkstra's algorithm [22] or by raising the adjacency matrix to higher powers, a matrix of shortest pathlengths may be computed. Having computed this matrix, it is often of interest to compute the average pathlength of a network. Short pathlengths may be desirable in instances such as communications networks. The small-world property defined by Watts and Strogatz [90] and discussed in Section 3.2 is characterised by short average pathlengths and high clustering coefficients. Figure 2.10 shows the spyplot of an adjacency matrix together with a plot of the average pathlength for each node.



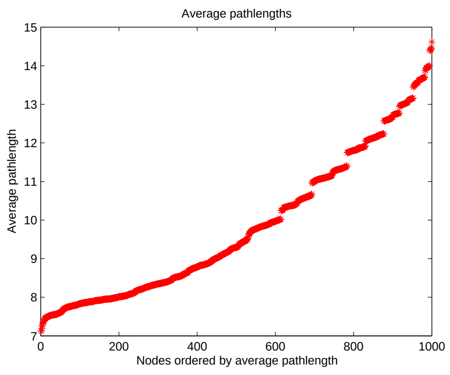


Figure 2.10: Network spy-plot and average pathlengths.

Chapter 3

Random Graph Models

Typical data mining and visualisation tasks reduce to linear system or eigenvalue computations with the large, sparse adjacency matrices that define the interactions. Several random graph models, that is, formulas for probabilistically inserting connections, have been derived that attempt to capture the key topological properties of real-life networks. In this chapter, we give brief introductions to some popular models. Pictures illustrating instances of these graphs can be found in Chapter 4.

3.1 Classical

Random graph theory began in the late 1950s with the two classical models in [35] and [25]. These models are usually referred to as $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$, but to help distinguish between them we will use the names Gilbert and Erdös-Rényi respectively.

In Gilbert's model [35] a fixed probability p is specified, and then pairs of nodes are connected independently at random with probability p. In the Erdös-Rényi model [25] the number of edges in the network, m, is specified. We then select uniformly at random from the set of all graphs containing n nodes and m edges. In this case, m must be no more than the maximum possible number of edges, $n(n-1)/2$.

Statistical properties of these random graphs have been well studied [2, 11] although in terms of currently adopted measures, such as pathlengths, clustering coefficients and graphlet frequencies, they cannot be regarded as accurate models of realistic networks [20, 75, 90].

3.2 Small World

Motivated by the "small-world" concept of the experimental psychologist Stanley Milgram [63], Watts and Strogatz [90] proposed a random graph model that can be regarded as interpolating between a regular, periodic lattice and a classical random graph. We begin with a k-nearest neighbour ring (nodes i and j are connected if and only if $|i-j| \leq k$ or $|n-|i-j|| \leq k$). Then, each edge is considered independently in turn. With fixed probability p that edge is rewired so that it has one new end point, a node chosen uniformly at random from the network (with self-links being disallowed).

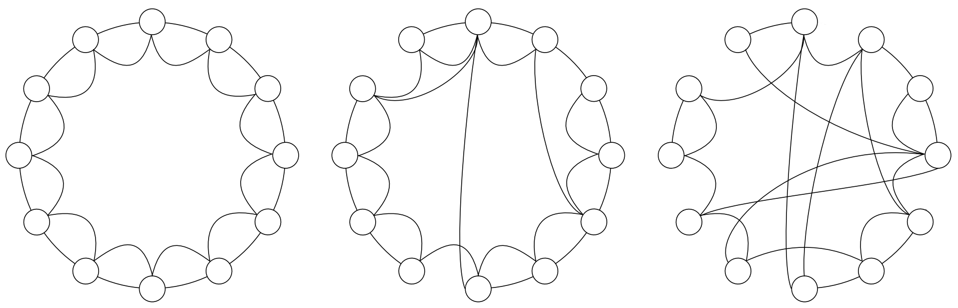


Figure 3.1: Small world network with $p = 0$, 0.2 and 0.5.

3.3 Geometric

A two-dimensional, non-periodic, *geometric random graph* may be defined as follows. First, each of the n nodes is placed at random in the unit square – more precisely, the ith node is given coordinates (x_i, y_i) where $\{x_i, y_i\}_{i=1}^n$ are independent and identically distributed with uniform (0,1) distribution. Next, for some specified radius r, nodes i and j are connected if and only if $(x_i - x_j)^2 + (y_i - y_j)^2 \leq r^2$. In words, an edge denotes that two nodes were placed no more than Euclidean distance r apart.

We emphasise that the resulting graph is simply the usual list of nodes and edges. Information about the precise node locations $\{x_i, y_i\}_{i=1}^n$ is not part of the final mathematical object. Natural generalizations are possible.

Dimension: the nodes can be randomly assigned to locations in the unit cube in $\mathbb{R}^m$, for some $m > 2$.

Periodicity: distance can be measured in a wrap-around fashion, so that, for example, in the unit square, $(x_i - x_j)^2 + (y_i - y_j)^2$ is replaced by

$$(\min(|x_i - x_j|, 1 - |x_i - x_j|))^2 + (\min(|y_i - y_j|, 1 - |y_i - y_j|))^2.$$

Norm: the Euclidean norm can be replaced by any other vector norm.

Much theory is available concerning properties of geometric random graphs [73]. It was shown in [75] that two and three dimensional non-periodic versions, using the Euclidean norm, give surprisingly accurate reproductions of many features of real biological networks and an algorithm that tests for geometric structure is developed in [51].

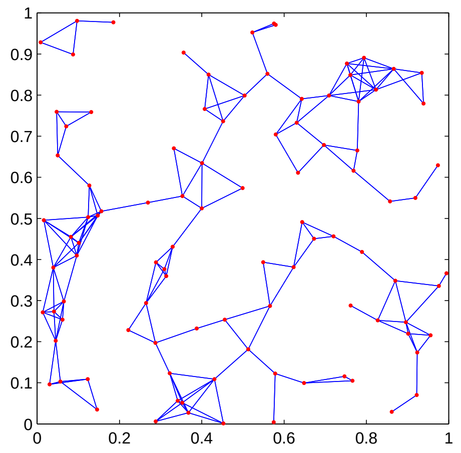


Figure 3.2: Geometric random graph with nodes distributed in the unit square.

3.4 Preferential Attachment

Barabási and Albert [4] used the concept of preferential attachment to develop random graphs with scale-free degree distributions. In this model, the network grows – new nodes are added and connected to the existing network – until n nodes have been created. For some fixed integer $d \geq 1$, each new node is given d edges on arrival. These new connections are not chosen uniformly; the new node links to an existing node with a probability that is proportional to the current degree of that node. In this way, well-connected nodes tend to become even better connected (the rich get richer) as the network evolves.

3.5 Range Dependent

3.5.1 RENGA

Yeast two hybrid protein-protein interaction (PPI) networks have proteins as nodes. Two nodes share an undirected edge if they have been experimentally observed to interact [94]. Motivated by the structure of PPI networks, Grindrod

[40] proposed and analysed a random graph model that, in a sense, generalizes Watts-Strogatz. In this model, the nodes have a natural linear ordering, $i = 1, 2, \cdots, n$. Independently over all pairs of nodes, we then insert a link between nodes i and j with probability $\alpha\lambda^{|j-i|-1}$, where $\alpha \in (0, 1]$ and $\lambda \in (0, 1)$ are fixed parameters. The choice $\alpha = 1$ ensures that adjacently ordered nodes are always connected. The geometric factor $\lambda^{|j-i|-1}$ causes long-range edges to be less common than short range edges. Figure 3.3 shows an example of such a network with $\alpha = 1$.

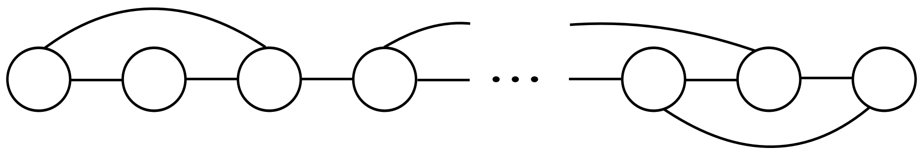


Figure 3.3: RENGA random graph with nodes arranged in a linear ordering.

Further analysis and generalizations of this model, sometimes referred to as RENGA, appear in [41, 45, 47]. Closely related models have also been used in percolation theory [39].

3.5.2 Kleinberg

Kleinberg [57] defined a variation of the Watts-Strogatz model, and used it to examine which types of navigation algorithm can exploit the existence of short cuts. Kleinberg's model is based on a periodic, two-dimensional lattice: the $n = m^2$ nodes can be thought of as being equally spaced throughout a square, with each node having a location of the form $(i, j) \in \mathbb{R}^2$, where the integers i and j run from 1 to m. Every node is given short range connections to its neighbours that are a lattice (Manhattan) distance of at most p away. Then each node is given q further 'long-range' connections. For a given node, u, the recipient, v, of each such long-range connection is chosen independently at random, with probability proportional to $r^{-\alpha}$. Here, r is the lattice distance between u and v and $\alpha \geq 0$ is a fixed parameter. Figure 3.4 shows an instance of the Kleinberg model, with nodes arranged in a lattice with nearest neighbours (Manhattan distance = 1) connected and shortcuts added.

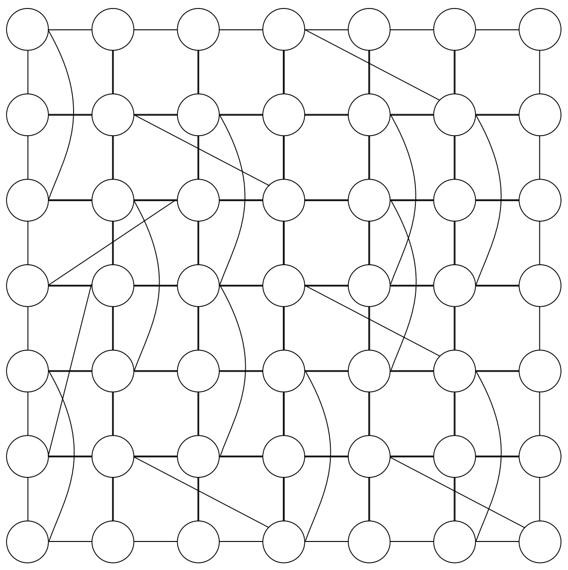


Figure 3.4: Kleinberg random graph.

3.6 Lock and Key

Using some basic biological insights, Thomas et al. [83] proposed a class of random graphs that model PPI networks. This class of models was further analysed in [66], where it was used to extract new biological information from real PPI data sets. The underlying modelling idea is that two proteins interact because they share physically matching parts, which, following [66], we refer to as *locks* and *keys*. There will be several different types of key, which we can think of as labeled by colours (red, green, blue, etc.) and for each type of key, there is a matching lock (red, green, blue, etc.). In the model, each protein has the same chance of possessing each colour of lock and each colour of key.

More precisely, for a given number of colours, m, we take each node in turn and independently assign it each possible lock and key with some fixed probability p. The graph is then generated according to the rule that two nodes share an edge if and only if one possesses a key and the other possesses a lock of the same colour. Self links are removed. We remark that the lock and key concept will be revisited in Chapter 6.

3.7 Stickiness

The stickiness model was introduced in [76] to model PPI networks. It was motivated as a simplified version of the lock and key framework in which parameters could be fitted to real data. Here, a nonnegative vector $\hat{d} \in \mathbb{R}^n$ is given, representing the scaled degree distribution of some target network; more precisely, $\hat{d}_i = \deg_i / \sqrt{\sum_{j=1}^{n} \deg_j}$, where $\deg_i$ is the degree of the ith node in the target. Then a new random network is produced by connecting nodes i and j with probability $\hat{d}_i \hat{d}_j$. In this way the *expected degrees* in the random model match the target degrees. (A generalization of this model to the case of directed edges will be given in Chapter 7.) This model was found to be more accurate than previously proposed models at reproducing topological properties of PPI networks.

Chapter 4

CONTEST

4.1 Motivation

From a numerical analysis perspective, the random graph models described in Chapter 3 provide an extremely useful source of realistic, controllable test matrices for linear algebra software. This provides the motivation for the MATLAB toolbox CONTEST (CONtrollable TEST matrices) developed here, which implements nine popular random graph models along with various utility functions for post-processing the networks. The codes were developed and tested under MATLAB Version 7.4.0.287 (R2007a).

A call to one of the random network functions in the toolbox will generate an $A \in \mathbb{R}^{n\times n}$ as an independent instance drawn from a random network model. MATLAB's built in pseudo-random number generators `rand` and `randn` provide the randomness in each model, and our codes do not alter their states. This means that it is possible to reproduce an adjacency matrix by resetting the states of the random number generators. Although certain parameter values may result in the creation of a dense or even full adjacency matrix, we adopt the convention of generating adjacency matrices with the `sparse` attribute, since the applications for which these codes may produce suitable test matrices typically work with sparse datasets.

Similarly, we adopt the convention of producing only symmetric adjacency matrices, although it is straightforward to create unsymmetric versions, corresponding to directed networks, simply by combining the upper and lower triangles from two independent samples from the same model. For instance, calling `A = erdrey(n,m)` and `B = erdrey(n,m)` then `C = triu(A) + tril(B)` would create an unsymmetric version of the Erdös-Rényi model as described in Section 4.2.1. For some models however, such as preferential attachment, it is unclear whether this is the best method of producing a directed network whilst preserving the key topological features of that model.

A recent and rapid expansion in theoretical and empirical research activity has produced several models for computing networks in a controlled manner that are "close" to real life networks in a well-defined sense. It is our tenet that these computable networks are therefore excellent candidates for test matrices.

Although well established sparse matrix test sets exist [10, 19, 23], they have been built around fixed instances arising in particular application areas. Randomness is typically incorporated very simplistically. For example, Matrix Market [10] with website URL `http://math.nist.gov/MatrixMarket/` makes available the random generators `DLATMR/ZLATMR` from LAPACK [3], which independently assign random samples from a given distribution across the entries of an array and then randomly reset elements to zero in order to achieve a given level of sparsity. In [19], Davis argues that "random sparse matrices" are not appropriate for testing sparse matrix algorithms; however, these comments would appear to be aimed at different classes of matrices to those considered here. The models implemented in CONTEST use randomness to capture properties that are commonly observed in complex interaction networks.

The code in CONTEST was written to exploit vectorization and to use matrix-vector level operations where possible, but ultimately our priority was to allow sparse matrices of the largest possible dimension to be computed. A secondary aim was to produce short, readable and maintainable programs. The importance of memory allocation and usage when generating sparse matrices in MATLAB is discussed in [36] and in NA DIGEST at `http://www.netlib.org/na-digest-html/07/v07n28.html1`. Our justification for not focusing on execution time is that the tasks that will typically be performed with the matrices–eigensolves, linear systems solves, factorizations–will usually be more computationally expensive than the matrix generation phase.

4.2 Models

In this section we briefly show how to use the MATLAB functions corresponding to each of the nine models described in Chapter 3. In each case, the output argument, `A`, is a sparse, symmetric zero-diagonal matrix of dimension `n` with `n` being the first of the input arguments. The remaining input arguments take default values if not specified in the function call. Default parameters have been chosen to ensure that `A` corresponds to a connected (irreducible) graph with high probability, with the exception of `sticky` in Section 4.2.8, which, by construction, may produce many small disconnected subgraphs. Throughout this chapter we show spy-plots of the nine models using `n`=100; this dimension was chosen to make the visualisation clearer - in practice values of `n` of the order 10^4 or higher would be more realistic.

4.2.1 Classical Codes: `gilbert` and `erdrey`

The function `gilbert(n,p)` returns an instance from the Gilbert class described in Section 3.1. The optional second input argument defaults to $log(n)/n$, so

`A = gilbert(n)` is equivalent to `A = gilbert(n,log(n)/n)`. Similarly, `A = erdrey(n,m)` produces an Erdös-Rényi random graph, with m defaulting to the smallest integer bigger than $n\,log(n)/2$. Figures 4.1 and 4.2 show spy-plots of instances of `gilbert` and `erdrey` for varying values of `p` and `m`.

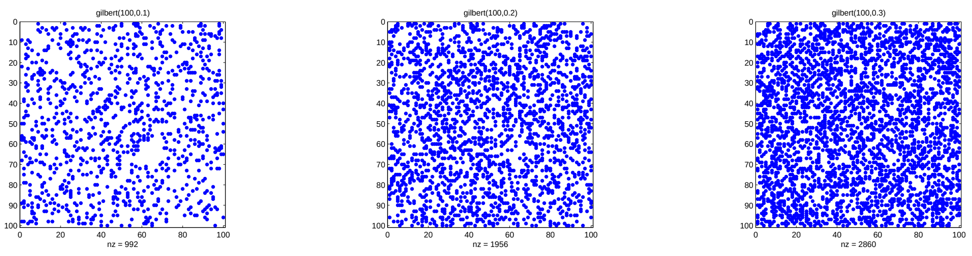


Figure 4.1: Gilbert random graphs with `p` taking values 0.1, 0.2 and 0.3.

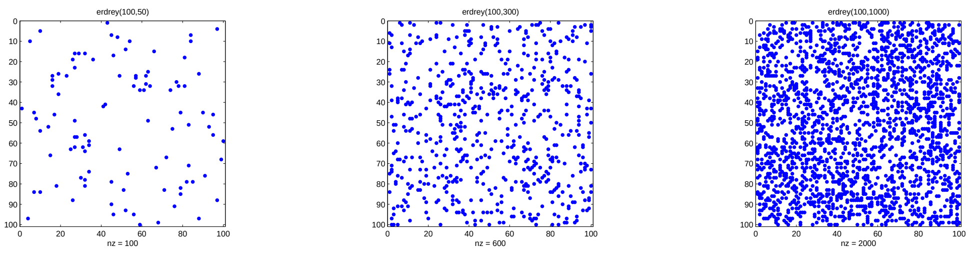


Figure 4.2: Erdös-Rényi random graphs with `m` taking values 50, 300 and 1000.

4.2.2 Small World Code: `smallw`

The function `smallw` returns an instance of the Watts-Strogatz model described in Section 3.2, with syntax according to `A = smallw(n,k,p)`. The optional input arguments `k` and `p` default to 2 and 0.1, respectively. From a linear algebra perspective, the adjacency matrix has a symmetric, banded Toeplitz structure, with extra nonzeros added uniformly and symmetrically at random. We note that `smallw` makes use of the utility function `short` that is described in Section 4.3.2. Figure 4.3 shows spy-plots of instances of `smallw` for varying values of `k` and `p`.

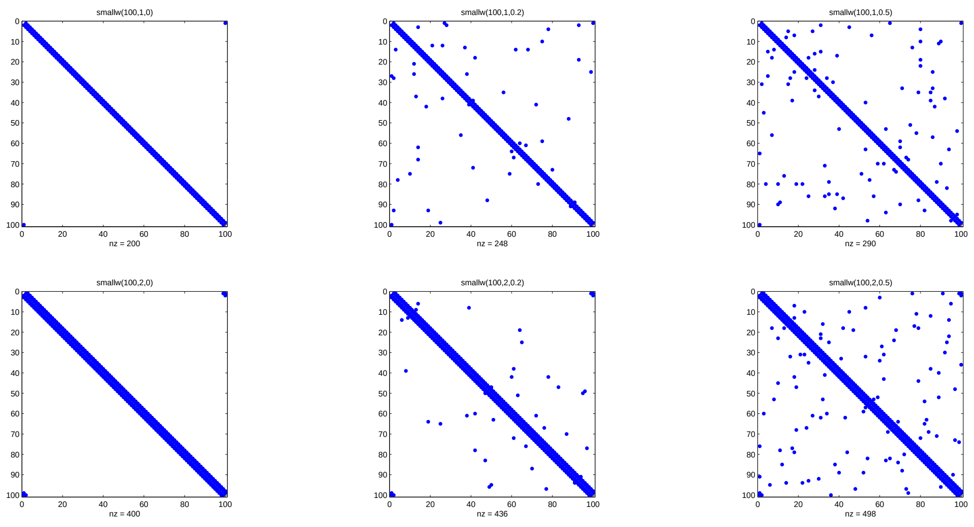


Figure 4.3: Small-world random graphs with `k` taking values 1 and 2 in rows 1 and 2, and p taking values 0, 0.2 and 0.5 in columns 1, 2 and 3 respectively.

4.2.3 Geometric Code: `geo`

The call `A = geo(n,r,m,per,pnorm)` returns an instance of a geometric random graph as described in Section 3.3. There are four optional input arguments:

- `r` specifies the radius, defaulting to $\sqrt{1.44/n}$, which is motivated by the asymptotic ($n \to \infty$) level that guarantees connectivity in two dimensions [73].
- `m` specifies the dimension, defaulting to 2.
- `per` is a logical variable specifying whether periodic distance is to be used, defaulting to `per = 0`; not periodic.
- `pnorm` specifies the L_p-norm to be used, defaulting to 2.

Figure 4.4 shows spy-plots of instances of `geo` for varying values of `per` and `r`.

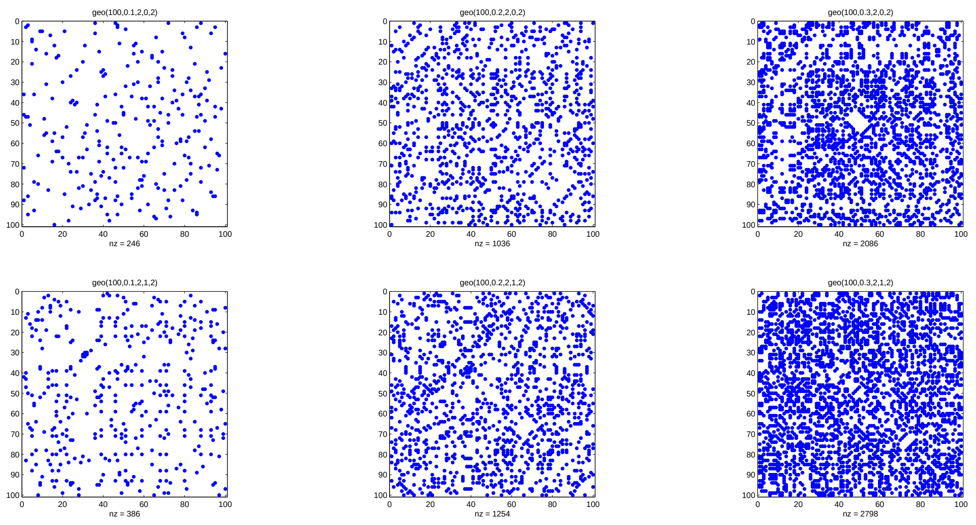


Figure 4.4: Geometric random graphs with `per` taking values 0 and 1 in rows 1 and 2, and `r` taking values 0.1, 0.2 and 0.3 in columns 1, 2 and 3 respectively. `m` and `pnorm` both take the default value 2.

4.2.4 Preferential Attachment Code: `pref`

The call `A = pref(n,d)` returns an instance of a preferential attachment graph as described in Section 3.4, using a single node as the initial network. The degree parameter `d` defaults to 2. Our precise model is a translation into MATLAB of [7, Algorithm 5] which uses the specification in [12]. Figure 4.5 shows spy-plots of instances of `pref` for varying values of `d`.

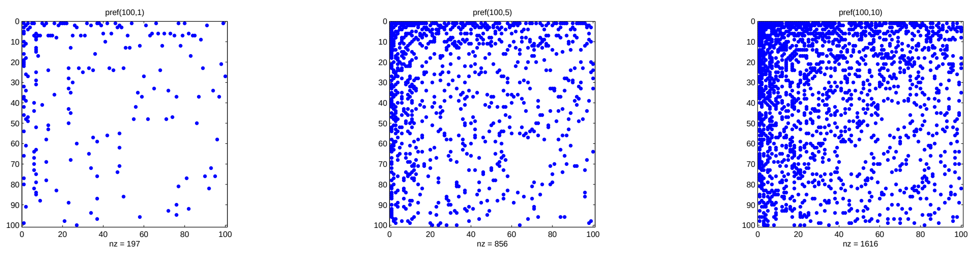


Figure 4.5: Preferential attachment graphs with `d` taking values 1, 5 and 10.

4.2.5 RENGA Code: `renga`

The call `A = renga(n,lambda,alpha)` returns an instance of a RENGA as described in Section 3.5.1, with `lambda` defaulting to 0.5 and `alpha` defaulting to 1. Figure 4.6 shows spy-plots of instances of `renga` for varying values of `lambda` and `alpha`.

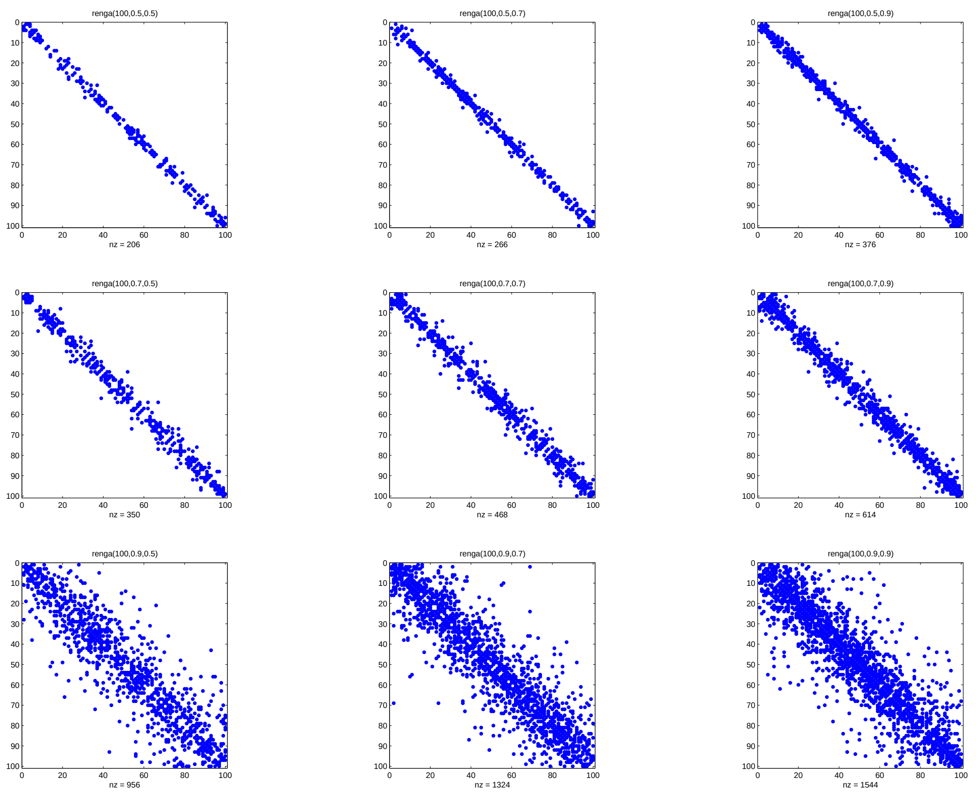


Figure 4.6: RENGA random graphs with `lambda` taking values 0.5, 0.7 and 0.9 in rows 1, 2 and 3, and `alpha` taking values 0.5, 0.7 and 0.9 in columns 1, 2 and 3 respectively.

4.2.6 Kleinberg Code: `kleinberg`

The call `A = kleinberg(n,p,q,alpha)` generates an instance of the Kleinberg model described in Section 3.5.2. If the input dimension, `n`, is not a perfect square then the output matrix has dimension `(round(sqrt(n)))`. Default values are `p = 1`, `q = 1` and `alpha = 2`. Figure 4.7 shows spy-plots of instances of `klein` for varying values of `p`, `q` and `alpha`.

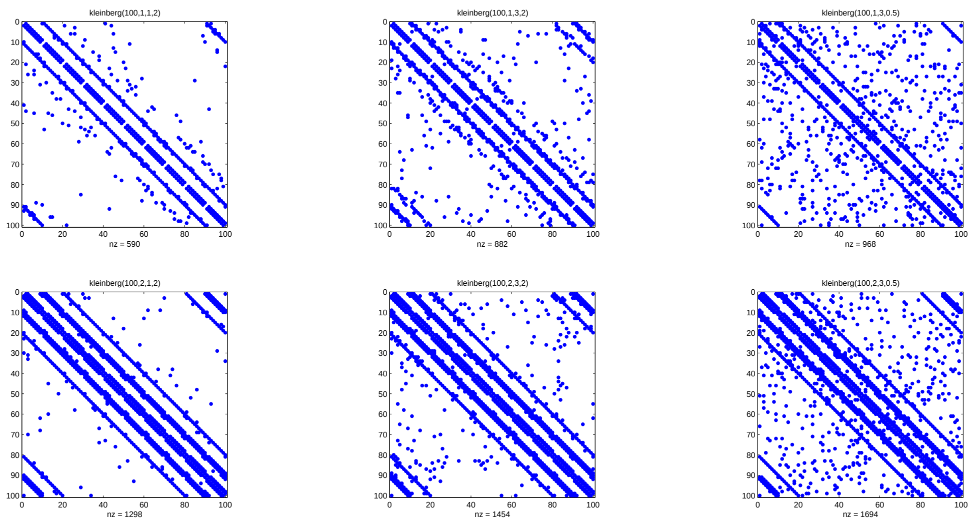


Figure 4.7: Kleinberg random graphs with `p` taking values 1 and 2 in rows 1 and 2 respectively. `q` takes the value 1 in column 1 and 3 in columns 2 and 3, and `alpha` takes the value 2 in columns 1 and 2 and 0.5 in column 3.

4.2.7 Lock and Key Code: `lockandkey`

The call `A = lockandkey(n,m,p)` returns an instance of a lock and key graph as described in Section 3.6, where there are `m` different lock and key colours and each type of lock and key is handed out independently with fixed probability `p`. Default values are `m = ceil(n*log(n))` and `p = 1/n`. Figure 4.8 shows spy-plots of instances of `lockandkey` for varying values of `m` and `p`.

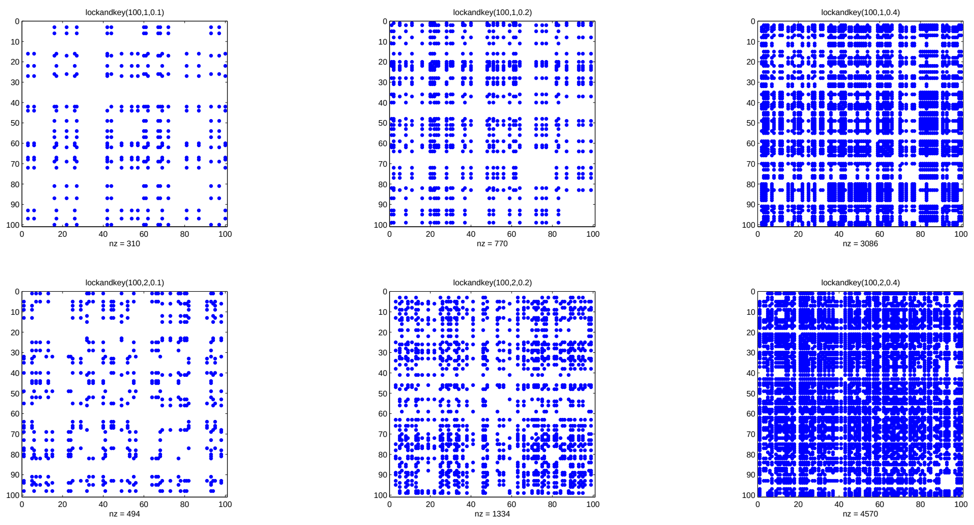


Figure 4.8: Lock and key graphs with `m` taking values 1 and 2 in rows 1 and 2, and `p` taking values 0.1, 0.2 and 0.4 in columns 1, 2 and 3 respectively.

4.2.8 Stickiness Code: `sticky`

The call `A = sticky(deg)` generates an instance of a stickiness graph as described in Section 3.7, with expected degree distribution given by the one-dimensional array `deg`. To be consistent with our general philosophy that all models can be called with a single input argument, `n`, representing the dimension, we allow an exception where `sticky` is called as `A = sticky(n)`, with `n` a positive integer. In this case `A` will be an instance of a stickiness graph of dimension `n` with a scale-free expected degree distribution of the form

$$\frac{\text{Number of nodes of degree } k}{n} \approx k^{-\gamma} \tag{4.1}$$

with $\gamma = 2.5$. It is also possible to specify two input parameters; a call `A = sticky(n,gamma)` specifies the value of γ to be used in 4.1. Figure 4.9 shows spy-plots of instances of `sticky` for varying values of `gamma`.

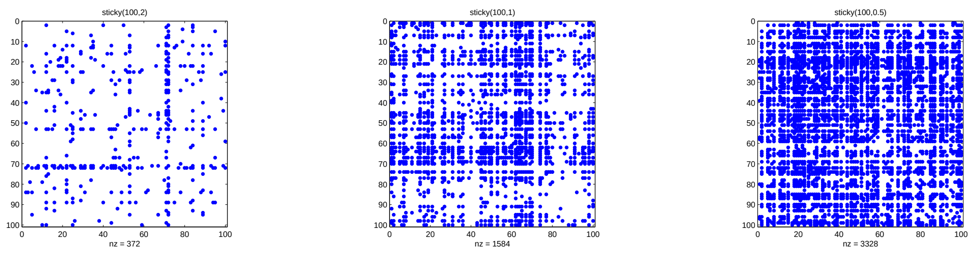


Figure 4.9: Stickiness graphs with `gamma` taking values 2, 1 and 0.5.

4.3 Utility Functions

4.3.1 Rewiring Code: `rewire`

The Watts-Strogatz model [90] added randomness to a ring network by *rewiring* some edges. For a general undirected network, we define a rewiring process as follows, in terms of a fixed parameter p. Each entry in the lower triangle of the original adjacency matrix is examined in turn. If $a_{ij} \neq 0$ then, independently with probability p, we reset $a_{ij} = a_{ji} = 0$, choose a node k uniformly at random from all non-neighbours of node i, and set $a_{ik} = a_{ki} = 1$.

The call `R = rewire(A,p)` takes an adjacency matrix `A` and returns a rewired adjacency matrix `R`. The rewiring probability `p` defaults to `p = log(n)/n`. Figure 4.10 shows the results of rewiring a Gilbert random graph when the rewiring probability is 0.5 and 1.

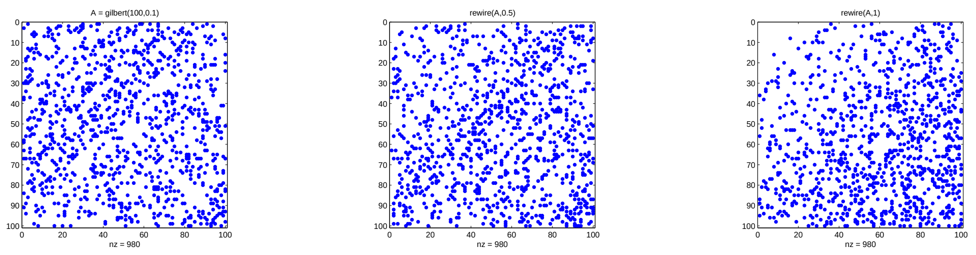


Figure 4.10: Gilbert random graph and rewired graphs with `p` taking values 0.1, 0.5 and 1.

4.3.2 Shortcut Code: `short`

Rewiring has the theoretical drawback that it may cause a connected network to become disconnected. Adding *shortcuts* is an alternative procedure that gives very similar topological effects [69] but does not degrade connectivity. In this case the parameter p is a fixed probability that is used independently over all nodes. For each node, with probability p we add a new link from that node to a node chosen uniformly at random across the whole network. Self links are then removed and repeated links treated as single links.

The call `S = short(A,p)` takes an adjacency matrix `A`, adds shortcuts and returns the new adjacency matrix `S`. The shortcut probability `p` defaults to `log(n)/n`. Figure 4.11 shows a Gilbert random graph together with plots of the same graph once shortcuts have been added, with `p` taking values 0.5 and 1.

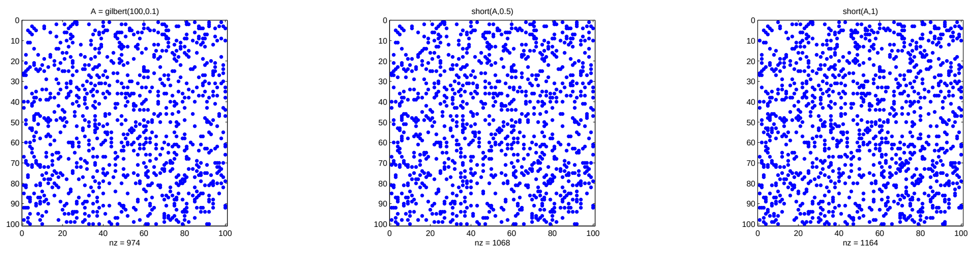


Figure 4.11: Gilbert random graph and graphs with shortcuts added, with `p` taking values 0.5 and 1.

4.3.3 Subsampling Codes: `unisample` and `baitsample`

Information is often missing from real life connectivity data sets [21]. These omissions may be caused, for example, by errors in experimental observations (false negatives) or by an inherent restriction on the number or type of observations that can be made. In the case of yeast two hybrid PPI networks, it is widely accepted that the reported network is merely a noisy subset of the underlying "true" network, and we can think of the given network as being generated from a "subsampling" operation on the larger version [84]. Interestingly, it has been discovered that the subsampling operation may dramatically alter the topological properties of a network [21, 43, 79].

We have implemented two subsampling algorithms. Given a set of nodes and edges, they return the adjacency matrix for a network consisting of a subset of those nodes and edges. The first algorithm does an unbiased, uniform node removal involving a fixed parameter p. Each node is considered in turn, and with independent probability $1-p$ we remove that node and all edges that involve it, that is, we delete that row and column from the adjacency matrix. The second algorithm uses a bait and prey approach, along the lines of [43], which models the generation of certain PPI data sets. Here, we use two fixed parameters, `bait` and `prey`. A proportion `bait` of the nodes are chosen as baits. Then, for each bait, a proportion `prey` of its edges are recorded, along with the prey nodes that are linked to the bait by those edges. The final subsampled network consists of the bait-prey edges and all the nodes that they involve.

The call `U = unisample(A,p)` takes an adjacency matrix `A` and returns a subnetwork `U` formed from an unbiased, uniform node removal. The probability `p` defaults to `0.5`. The bait and prey algorithm can be called as `B = baitsample(A,bait,prey)` with defaults `bait = 0.5` and `prey = 0.5`.

Figure 4.12 shows an instance of a Gilbert random graph and two subgraphs sampled using `unisample` with `p` taking the values 0.5 and 0.2. Figure 4.13 shows the same graph, this time sampled using `baitsample` with input arguments `bait = 0.5`, `prey = 0.5` and `bait = 0.2`, `prey = 0.9`. The degree distribution of each network is plotted underneath the corresponding spy-plot.

Figures 4.14 and 4.15 show an instance of a preferential attachment graph sampled under the same conditions described above. Again, the degree distributions for each network are plotted directly under their spy-plot. In this instance we can clearly see that sampling methodology has a large effect on the preservation of topological features, as a scale free degree distribution is observable in Figure 4.15 but not in Figure 4.14.

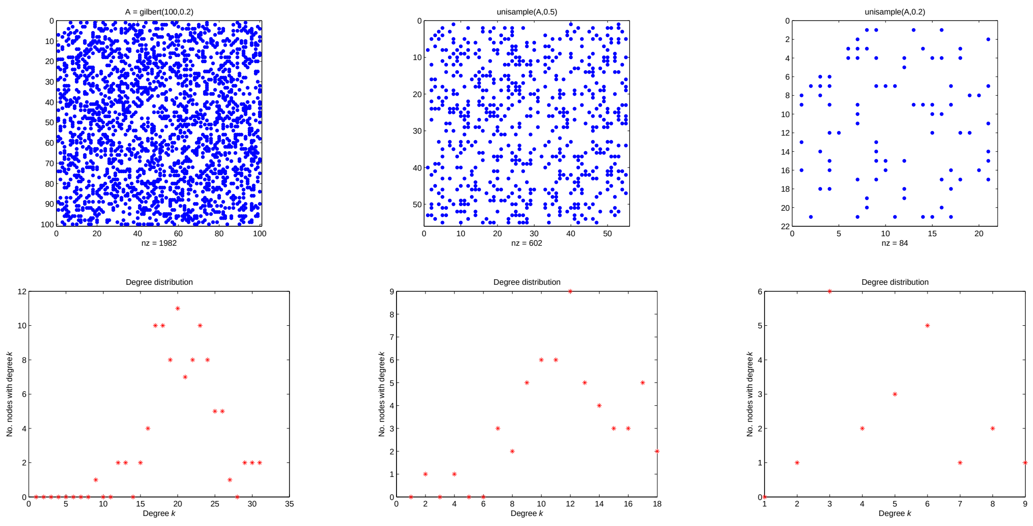


Figure 4.12: Spyplots and degree distributions for an instance of a Gilbert random graph and subgraphs sampled using `unisample`, with `p` taking values 0.5 and 1.

4.3.4 Laplacian Matrix Code: `lap`

An undirected network can be characterised by its adjacency matrix, and basic linear algebra tells us that the eigenvectors and eigenvalues of this matrix carry relevant information. However, spectral graph theory [15] has shown that it is generally more useful to look at the spectrum of the so-called Laplacian. There are two different matrices that take this name in the literature. We distinguish between them as follows.

- The *graph Laplacian* has the form $D - A$.
- The *normalised graph Laplacian* has the form $\hat{D}^{-\frac{1}{2}}(D - A)\hat{D}^{-\frac{1}{2}}$.

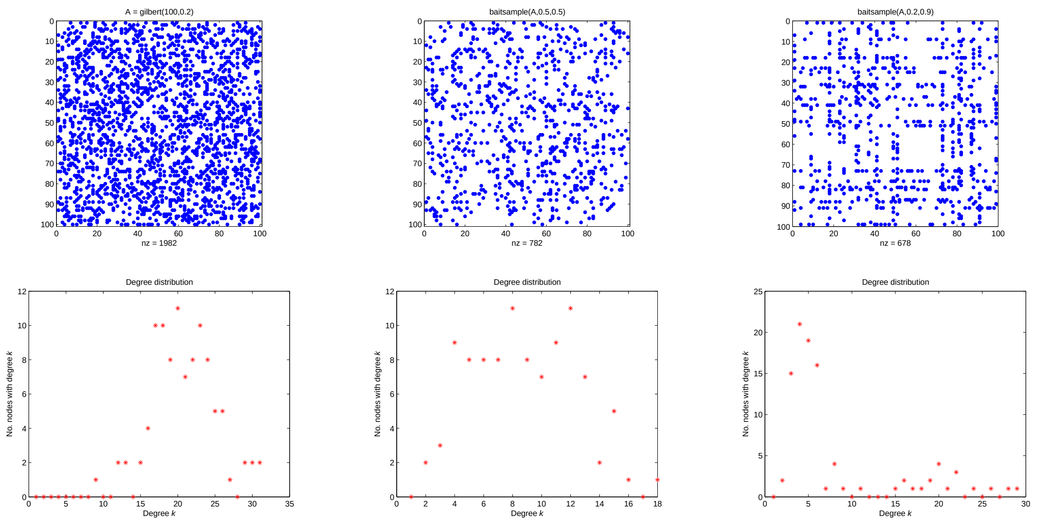


Figure 4.13: Spyplots and degree distributions for an instance of a Gilbert random graph and subgraphs sampled using `baitsample`, with `bait` taking values 0.5 and 0.2, and `prey` taking vales 0.5 and 0.9.

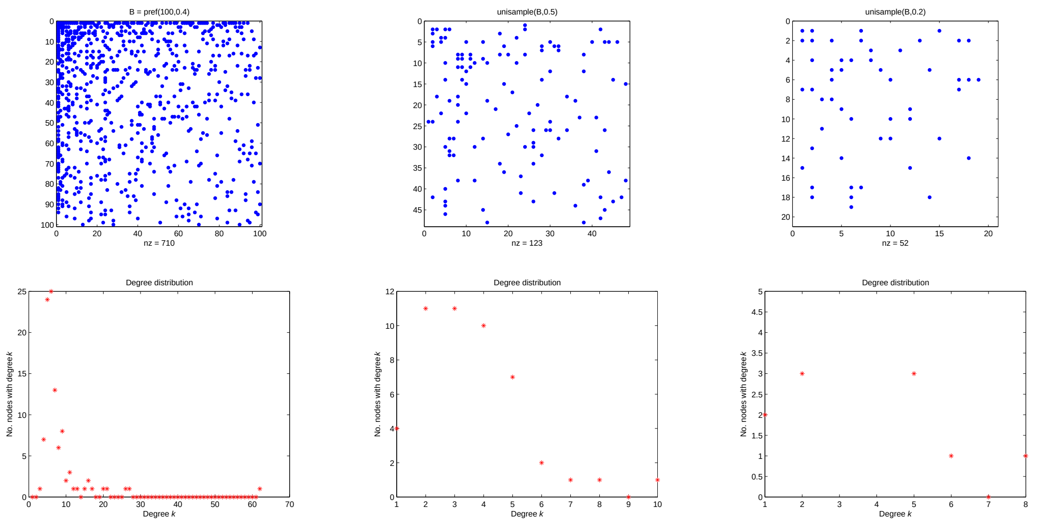


Figure 4.14: Spyplots and degree distributions for an instance of a preferential attachment graph subgraphs sampled using `unisample`, with `p` taking values 0.5 and 0.2.

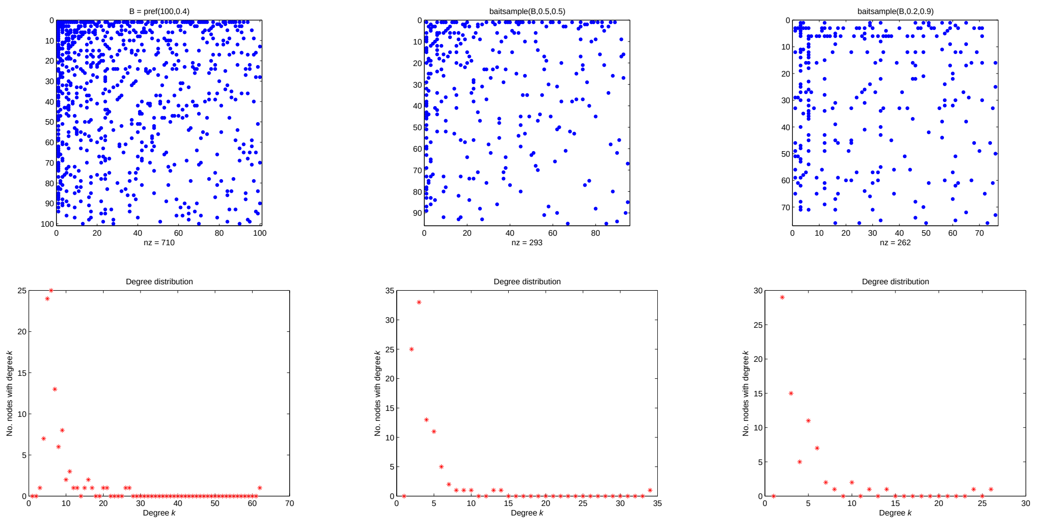


Figure 4.15: Spyplots and degree distributions for an instance of a preferential attachment graph subgraphs sampled using `baitsample`, with `bait` taking values 0.5 and 0.2 and `prey` taking values 0.5 and 0.9.

Here $D = \mathrm{diag}(\mathrm{deg}_i)$ and $\hat{D} = D$ with the exception that we take $\hat{D}_{ii} = 1$ in the case where $\mathrm{deg}_i = 0$.

In the case of a connected graph, the eigenvalues of the Laplacian are non-negative, with smallest eigenvalue $\lambda_1 = 0$ corresponding to the eigenvector $\mathbf{1}$. Similarly, the normalised Laplacian has smallest eigenvalue $\mu_1 = 0$ corresponding to the eigenvector $D^{\frac{1}{2}}\mathbf{1}$ and it can further be shown that the eigenvalues are bounded by $0 \le \mu_i \le 2$ [88].

Clustering and partitioning tasks can be tackled by computing eigenvectors corresponding to small eigenvalues of these matrices. In particular the *Fiedler vector* and *normalised Fiedler vector* of a connected network are defined to be the eigenvectors corresponding to the second smallest eigenvalues of the Laplacian and normalised Laplacian, respectively [49]. Specific software exists for computing this type of information [17, 44, 53].

The call `L = lap(A,nl)` takes a symmetric adjacency matrix `A` and returns a Laplacian; `nl=0` for unnormalised and `nl=1` for normalised. The default is `nl=1`.

4.3.5 PageRank Code: `pagerank`

The PageRank algorithm returns a vector whose ith entry indicates the "importance" of the ith node in a network. The algorithm was invented by Page and Brin and forms the heart of the search engine Google [58, 72]. PageRank was originally designed for the directed network where nodes are web pages and edges are hyperlinks, but it has also been used on networks in biology [65]. Given an adjacency matrix A, the PageRank vector x solves the linear system

$$Px = \mathbf{1} \text{ where } P = I - dA^T\hat{D}^{-1}.$$

Here, $d \in (0, 1)$ is a scalar parameter, the diagonal degree matrix $\hat{D}$ is defined in Section 4.3.4 and $\mathbf{1}$ denotes the vector of 1s. More precisely, when A is unsymmetric we consider the *out degree*, so $D = \text{diag}(\sum_{j=1}^{N} a_{ij})$ and $\hat{D} = \text{diag}(\max(D_{ii}, 1))$.

The call `P = pagerank(A,d)` takes an adjacency matrix `A` and returns the PageRank matrix `P`, with `d` defaulting to 0.15. The matrix `A` is not assumed to be symmetric - directed edges are allowed.

4.3.6 Mean Hitting Time Code: `mht`

In many applications it is useful to consider the discrete time, finite state space, Markov chain that arises naturally from a network [60]. Here, if we are currently at node i then at the next time level we move to a node chosen uniformly among the neighbours of node i. The *transition matrix* for this Markov chain thus has the form $D^{-1}A$. Fixing a node, i, the *mean hitting time* for node j is defined to be the average number of steps required for the Markov chain to reach state j, given that it starts at state i. The vector of mean hitting times can be found by solving the linear system $Mx = \mathbf{1}$ where $M \in \mathbb{R}^{n-1 \times n-1}$ is the transition matrix with its ith row and column removed [70].

The call `M = mht(A,i)` takes an adjacency matrix `A` with nonzero out degrees and returns the mean hitting time matrix `M` for a chain that starts at node `i`, with `i` defaulting to 1. The matrix `A` is not required to be symmetric.

4.3.7 Pathlength Code: `pathlength`

The *pathlength* between nodes i and j is the smallest number of edges that must be crossed to reach j starting from i. In terms of the adjacency matrix, A, the pathlength between nodes i and j can be characterised as the smallest integer $k \geq 1$ such that $(A^k)_{ij} \neq 0$. If $(A^k)_{ij} = 0$ for all $1 \leq k \leq n-1$ then there is no suitable path and the pathlength may be regarded as infinite.

The call `Path = pathlength(A)` returns an array `Path` of the same dimension as the adjacency matrix `A`, such that `Path(i,j)` is the pathlength from node `i` to node `j`. We always set `Path(i,i)=0` and we use `Path(i,j)=inf` to denote that no path exists.

4.3.8 Curvature Code: `curvature`

The *curvature*, or *clustering coefficient*, of a node was defined in Section 2.4.2. In MATLAB notation, the vector of clustering coefficients may be computed as

```
diag(A^3)/(sum(A).*(sum(A) - 1)).
```

The call `curv = curvature(A)` takes an adjacency matrix `A` of dimension `n` and returns a one-dimensional array `curv` of length `n`, such that `curv(i)` records the curvature of node `i`. A second input argument is allowed. The call `curv = curvature(A,ind)` returns the maximum curvature if `ind` is the string `‘max’`, the average curvature if `ind` is the string `‘ave’` and the curvature for the `i`th node if `ind` is the integer `i`. Undefined curvature evaluates to `NaN`.

4.4 Numerical Testing

For a brief illustration of the toolbox in use, we follow Davis [19] by examining the complexity of the minimum degree ordering algorithm as implemented in MATLAB's `amd`. Let L denote the Cholesky factor of the appropriate permuted version of A and $|L|$ be the operation count for computing L. We calculate the run time (scaled by $|L|$) and plot against $|L|$ on a log-log scale. Davis [19] distinguished between matrices from a deterministic test set coming from problems with and without inherent geometry. To mirror this, Figure 4.16 shows results for matrices arising from the Gilbert class, using `gilbert`, where there is no inherent structure, and Figure 4.17 shows results arising from the Kleinberg class, using `klein`, where there is an underlying lattice. A best-fit line has been superimposed onto each plot using the method of least-squares. In each case, the matrix dimension n was varied between 50 and 10,000. The figures are consistent with the rule of thumb mentioned in [19] that the run time is typically below $O(|L|)$. The figures for the remaining seven models implemented in CONTEST may be found in Appendix B.1.

For each of the random graph models implemented in CONTEST, the following testing procedure was carried out. We generated one hundred instances of a random graph with dimension $n = 10,000$ and any other input arguments taking default values. Each adjacency matrix was then supplied as input to the function `pagerank`, with the default value `d=0.15`. The linear system of equations $A\mathbf{x} = \mathbf{1}$ was then solved, where A is the PageRank matrix and $\mathbf{1}$ is a vector of ones. Each system was solved by nine different numerical solvers in MATLAB with no preconditioning, with the LU preconditioner and with the

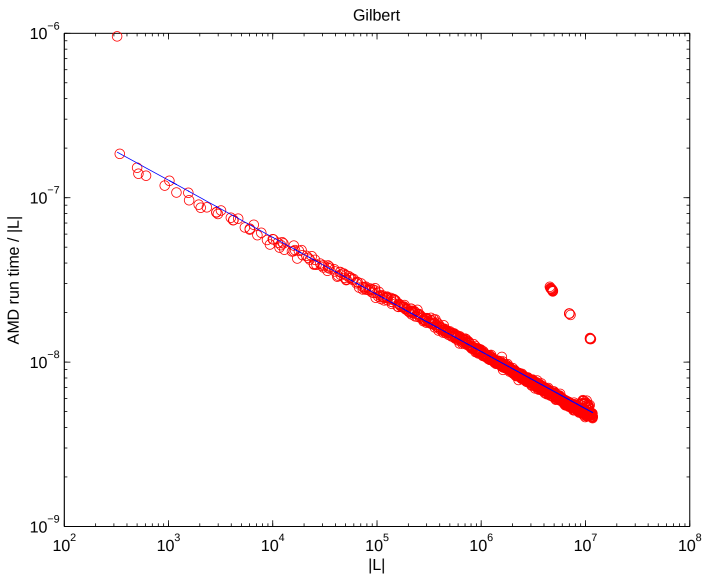


Figure 4.16: `amd` run times for Gilbert model.

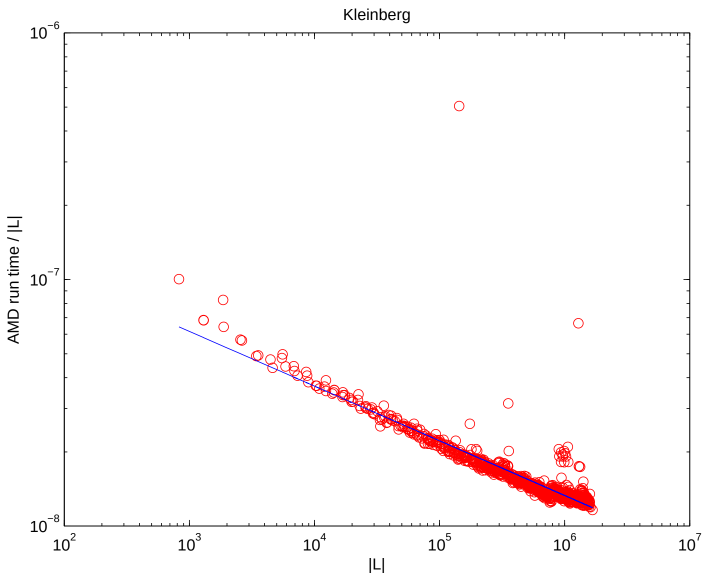


Figure 4.17: `amd` run times for Kleinberg model.

Cholesky preconditioner. In each case the running time, relative residual and iteration count were recorded.

We present here the results for the function `smallw`. Each entry in a table represents the mean over 100 test runs with the standard error given in parentheses. Note that the figures in the tables represent the mean over those test cases where the system converged to a solution, with a tolerance of $1e-10$ and a maximum of 100 iterations. Results are presented for both symmetric matrices and unsymmetric matrices obtained in the manner described in Section 4.1. The results for the remaining eight random graph models can be found in Appendix B.2. Tests were run on a 2 GHz AMD Opteron$^{\text{TM}}$ Processor 252 with 11 gigabytes of memory.

Footnotes indicate the state under which a computation was ended and we explain these here. State 0 indicates that the process coverged to the specified tolerance within the maximum number of iterations. State 1 indicates that the maximum number of iterations was reached without convergence to a solution. State 2 indicates that the preconditioner was ill-conditioned. State 3 indicates that the process stagnated, i.e. successive iterations were the same. State 4 indicates that a scalar quantity in the computation became too large or too small for MATLAB to process. If no footnote is present, this indicates that all processes finished in state 0, i.e. they converged to the required tolerance within the maximum number of iterations.

4.4.1 Small World Symmetric

[2]45% of cases finished in state 0, 50% in state 1 and 5% in state 4

[3]All cases finished in state 1

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$2.8571\,\mathrm{e}-2$ $(1.1\,\mathrm{e}-4)$	$6.1093\,\mathrm{e}-2$ $(2.5\,\mathrm{e}-4)$	$1.2558\,\mathrm{e}-1$ $(4.1\,\mathrm{e}-4)$
qmr	$4.5762\,\mathrm{e}-2$ $(1.9\,\mathrm{e}-4)$	$1.4697\,\mathrm{e}-1$ $(2.6\,\mathrm{e}-4)$	$1.6393\,\mathrm{e}-1$ $(1.1\,\mathrm{e}-3)$
symmlq	$2.3310\,\mathrm{e}-2$ $(8.7\,\mathrm{e}-5)$	$7.1339\,\mathrm{e}-2$ $(2.5\,\mathrm{e}-4)$	N/A[3]
lsqr	$7.5776\,\mathrm{e}-2$ $(2.9\,\mathrm{e}-4)$	$1.9750\,\mathrm{e}-1$ $(5.1\,\mathrm{e}-4)$	$1.7368\,\mathrm{e}-1$ $(4.4\,\mathrm{e}-4)$
minres	$2.3502\,\mathrm{e}-2$ $(8.6\,\mathrm{e}-5)$	$7.5534\,\mathrm{e}-2$ $(2.3\,\mathrm{e}-4)$	N/A[3]
cgs	$5.0228\,\mathrm{e}-2$ $(7.2\,\mathrm{e}-3)$[2]	$5.6789\,\mathrm{e}-2$ $(1.9\,\mathrm{e}-4)$	$7.3794\,\mathrm{e}-2$ $(5.6\,\mathrm{e}-4)$
gmres	$9.2971\,\mathrm{e}-2$ $(1.0\,\mathrm{e}-4)$	$1.3237\,\mathrm{e}-1$ $(2.3\,\mathrm{e}-4)$	$1.4325\,\mathrm{e}-1$ $(2.4\,\mathrm{e}-4)$
bicg	$3.7511\,\mathrm{e}-2$ $(1.1\,\mathrm{e}-4)$	$1.2683\,\mathrm{e}-1$ $(3.1\,\mathrm{e}-4)$	$1.4439\,\mathrm{e}-1$ $(9.3\,\mathrm{e}-4)$
bicgstab	$3.2351\,\mathrm{e}-2$ $(6.0\,\mathrm{e}-4)$	$6.2634\,\mathrm{e}-2$ $(1.9\,\mathrm{e}-4)$	$7.2459\,\mathrm{e}-2$ $(4.8\,\mathrm{e}-4)$

Table 4.1: Mean running times and standard errors for `smallw`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$15\,\mathrm{e}+0$ (0)
qmr	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.4200\,\mathrm{e}+0$ $(5.4\,\mathrm{e}-2)$
symmlq	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[3]
lsqr	$10\,\mathrm{e}+0$ (0)	$5\,\mathrm{e}+0$ (0)	$8\,\mathrm{e}+0$ (0)
minres	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[3]
cgs	$1.0667\,\mathrm{e}+1$ $(1.6\,\mathrm{e}+0)$[2]	$2\,\mathrm{e}+0$ (0)	$4.9300\,\mathrm{e}+0$ $(3.6\,\mathrm{e}-2)$
gmres	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)
bicg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.3900\,\mathrm{e}+0$ $(5.1\,\mathrm{e}-2)$
bicgstab	$4.6000\,\mathrm{e}+0$ $(8.4\,\mathrm{e}-2)$	$2\,\mathrm{e}+0$ (0)	$4.1450\,\mathrm{e}+0$ $(2.3\,\mathrm{e}-2)$

Table 4.2: Mean iteration counts and standard errors for `smallw`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$4.6274\,e-11$ $(1.3\,e-13)$	$1.8375\,e-12$ $(4.4\,e-14)$	$8.0102\,e-11$ $(3.7\,e-13)$
qmr	$4.6382\,e-11$ $(5.1\,e-13)$	$1.8233\,e-12$ $(4.4\,e-14)$	$4.0765\,e-11$ $(3.0\,e-12)$
symmlq	$4.6399\,e-11$ $(1.3\,e-13)$	$1.8375\,e-12$ $(4.4\,e-14)$	N/A[3]
lsqr	$7.1549\,e-11$ $(1.5\,e-13)$	$4.5037\,e-12$ $(5.5\,e-14)$	$3.1691\,e-11$ $(1.5\,e-13)$
minres	$4.6298\,e-11$ $(1.3\,e-13)$	$1.8375\,e-12$ $(4.4\,e-14)$	N/A[3]
cgs	$3.1761\,e-11$ $(3.9\,e-12)$[2]	$4.4617\,e-12$ $(4.5\,e-14)$	$1.1838\,e-11$ $(2.8\,e-12)$
gmres	$2.6102\,e-11$ $(4.3\,e-14)$	$1.8786\,e-12$ $(4.5\,e-14)$	$3.1784\,e-11$ $(3.2\,e-14)$
bicg	$4.1428\,e-11$ $(3.3\,e-13)$	$1.8235\,e-12$ $(4.4\,e-14)$	$4.3715\,e-11$ $(3.0\,e-12)$
bicgstab	$4.8405\,e-11$ $(2.9\,e-12)$	$3.2096\,e-12$ $(9.2\,e-14)$	$4.2227\,e-11$ $(2.7\,e-12)$

Table 4.3: Mean relative residuals and standard errors for `smallw`.

4.4.2 Small World Unsymmetric

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$3.7663\,e-2$ $(2.3\,e-3)$	$6.8470\,e-2$ $(1.7\,e-4)$	$1.4327\,e-1$ $(3.3\,e-4)$
qmr	$5.4876\,e-2$ $(6.1\,e-4)$	$1.5599\,e-1$ $(3.1\,e-4)$	$1.7486\,e-1$ $(9.4\,e-4)$
symmlq	$2.8257\,e-2$ $(3.5\,e-4)$	$7.8491\,e-2$ $(3.1\,e-4)$	N/A[5]
lsqr	$9.0077\,e-2$ $(6.6\,e-4)$	$2.1509\,e-1$ $(3.2\,e-4)$	$1.9354\,e-1$ $(3.2\,e-4)$
minres	$2.9211\,e-2$ $(3.5\,e-4)$	$8.2459\,e-2$ $(3.5\,e-4)$	N/A[5]
cgs	$6.4824\,e-2$ $(9.4\,e-3)$[4]	$6.2438\,e-2$ $(1.8\,e-4)$	$8.1249\,e-2$ $(7.7\,e-4)$
gmres	$1.0034\,e-1$ $(2.3\,e-3)$	$1.3870\,e-1$ $(3.2\,e-4)$	$1.5171\,e-1$ $(3.4\,e-4)$
bicg	$4.5039\,e-2$ $(3.1\,e-4)$	$1.3716\,e-1$ $(2.3\,e-4)$	$1.6204\,e-1$ $(9.8\,e-4)$
bicgstab	$4.1794\,e-2$ $(8.5\,e-4)$	$6.8753\,e-2$ $(1.9\,e-4)$	$8.2370\,e-2$ $(4.7\,e-4)$

Table 4.4: Mean running times and standard errors for unsymmetric `smallw`.

For both classes of network, we see that run times, iteration counts and relative residuals are typically lowest for all methods when *LU* preconditioning is used. Additionally, all runs terminated under normal conditions (i.e. in state 0). For both symmetric and unsymmetric instances of the small world model, all

[4]47% of cases finished in state 0, 48% in state 1 and 5% in state 4
[5]All cases finished in state 1

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$15\,\mathrm{e}+0$ (0)
qmr	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.3100\,\mathrm{e}+0$ ($4.6\,\mathrm{e}-2$)
symmlq	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[5]
lsqr	$10\,\mathrm{e}+0$ (0)	$5\,\mathrm{e}+0$ (0)	$8\,\mathrm{e}+0$ (0)
minres	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[5]
cgs	$1.1255\,\mathrm{e}+1$ ($1.7\,\mathrm{e}+0$)[4]	$2\,\mathrm{e}+0$ (0)	$4.7600\,\mathrm{e}+0$ ($4.5\,\mathrm{e}-2$)
gmres	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)
bicg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.3800\,\mathrm{e}+0$ ($5.3\,\mathrm{e}-2$)
bicgstab	$4.7800\,\mathrm{e}+0$ ($9.9\,\mathrm{e}-2$)	$2\,\mathrm{e}+0$ (0)	$4.1200\,\mathrm{e}+0$ ($2.3\,\mathrm{e}-2$)

Table 4.5: Mean iteration counts and standard errors for unsymmetric `smallw`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$2.8630\,\mathrm{e}-11$ ($3.7\,\mathrm{e}-14$)	$1.3941\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	$6.6990\,\mathrm{e}-11$ ($3.5\,\mathrm{e}-13$)
qmr	$2.6090\,\mathrm{e}-11$ ($6.6\,\mathrm{e}-14$)	$1.3968\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	$4.5115\,\mathrm{e}-11$ ($2.9\,\mathrm{e}-12$)
symmlq	$2.8646\,\mathrm{e}-11$ ($3.7\,\mathrm{e}-14$)	$1.3940\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	N/A[5]
lsqr	$4.1215\,\mathrm{e}-11$ ($5.6\,\mathrm{e}-14$)	$1.3843\,\mathrm{e}-12$ ($1.3\,\mathrm{e}-14$)	$1.9829\,\mathrm{e}-11$ ($3.0\,\mathrm{e}-14$)
minres	$2.8594\,\mathrm{e}-11$ ($3.7\,\mathrm{e}-14$)	$1.3940\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	N/A[5]
cgs	$3.8478\,\mathrm{e}-11$ ($4.5\,\mathrm{e}-12$)[4]	$2.3002\,\mathrm{e}-12$ ($2.8\,\mathrm{e}-14$)	$2.6326\,\mathrm{e}-11$ ($3.9\,\mathrm{e}-12$)
gmres	$1.8578\,\mathrm{e}-11$ ($2.2\,\mathrm{e}-14$)	$1.4386\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	$2.5916\,\mathrm{e}-11$ ($2.1\,\mathrm{e}-14$)
bicg	$2.6052\,\mathrm{e}-11$ ($4.3\,\mathrm{e}-14$)	$1.3970\,\mathrm{e}-12$ ($2.0\,\mathrm{e}-14$)	$4.1125\,\mathrm{e}-11$ ($2.9\,\mathrm{e}-12$)
bicgstab	$4.5101\,\mathrm{e}-11$ ($3.0\,\mathrm{e}-12$)	$2.4079\,\mathrm{e}-12$ ($2.5\,\mathrm{e}-14$)	$4.0653\,\mathrm{e}-11$ ($2.5\,\mathrm{e}-12$)

Table 4.6: Mean relative residuals and standard errors for unsymmetric `smallw`.

runs terminated normally except the `cgs` scheme when no preconditioner was used and the `symmlq` and `minres` schemes when the Cholesky preconditioner was used.

Chapter 5

NESSIE

5.1 Motivation

NESSIE (Network Example Source Supporting Innovative Experimentation) is a web-based facility that makes available 12 realistic examples of complex networks. The motivation for this collection is to provide a test set of example networks, taken from a variety of application areas, that may be useful for testing and comparing algorithms in network science. The networks are made available as adjacency matrices in MATLAB's `.mat` format. Some of the networks are derived by calculating correlation coefficients on non-square matrices, and in these cases both the original data and a sample adjacency matrix are provided. The NESSIE network collection is available from the URL

`http://fox.maths.strath.ac.uk/~aap05145/Nessie/nessie.html.`

Although we are unaware of any directly comparable resource, we mention some related projects here.

CONTEST [82] is a MATLAB toolbox that allows generation of networks as instances of various random graph models. The models and utilities implemented in CONTEST are described in Chapter 4.

GraphCrunch [62] is a network comparison tool. Global and local properties of an input network are calculated and compared with those of random network models. The model networks are calibrated to have a number of nodes and edges within 1% of those in the target network.

The University of Florida Sparse Matrix Collection [19] is a collection of instances of sparse matrices from a variety of areas. Like CONTEST, the collection is intended for the testing and development of sparse matrix algorithms. Although the matrices arise from several application areas in science and engineering, including fluid dynamics, electromagnetics, computer graphics and robotics, some represent network connectivity patters. This differs from CONTEST in that it is a set of static matrices, whereas CONTEST affords the user parameterized control over matrix dimension and features such as sparsity and degree distribution. Some of the test sets included in the University of Florida collection can also be found in Matrix Market [10], a similar web-based facility.

UCINET IV Datasets, available from

`http://vlado.fmf.uni-lj.so/pub/networks/data/Ucinet/UciData.htm`

provides a set of small networks (between 10 and 58 nodes each) from sociology that describe interactions between individuals.

Pajek is a tool for analysis and visualization of large networks. As an addition it includes a set of around 50 examples of networks, primarily related to text mining applications. Pajek is available from `http://vlado.fmf.uni-lj.so/pub/networks/pajek.`

5.2 Philosophy

Our aim is that NESSIE provides an uncluttered, accessible and informative network repository. The networks are made available as `.txt` files and `.mat` files. In the case of MATLAB files, the adjacency matrices have the `sparse` attribute [36, 48]. MATLAB utilities for computing the Pearson correlation coefficient and for Estrada's classification system [27] are included. In the next section we provide basic information about each network and show plots of various measures, such as degree distributions and clustering coefficients.

5.3 The Networks

We now describe the 12 networks comprising the example set. In each case we illustrate the adjacency matrix in a MATLAB spy-plot and show plots of some basic network measures: degree distributions, clustering coefficients, eigenvalues and the components of the Fiedler vector (as well as a spy-plot of the adjacency matrix reordered according to the Fiedler vector). When considering degree, we plot a cumulative degree distribution (as well as a degree histogram) for each network, as this allows more straightforward comparison of networks of different sizes. We also show the spectral scaling property of the network as described in [27]; here we plot $\log_{10}{}^{\mathrm{S}}C_{\mathrm{odd}}(i)$ against $\log_{10}\gamma_1(i)$, where

- γ_1 is a dominant eigenvector of the adjacency matrix, and
- ${}^{\mathrm{S}}C_{\mathrm{odd}}(i)$ is the *odd subgraph centrality* of node i. This is defined by summing over all possible walks of length k, the number of odd-length closed walks

starting and finishing at node i, scaled by $1/(k!)$. On the same axes, we plot the straight line defined by $y = 0.5x - 0.5\log_{10}(\lambda_1)$ where λ_1 is the eigenvalue corresponding to γ_1.

In [27] Estrada distinguishes four topological classes based on such plots.
Class I : the points $(\log_{10}{}^{\mathrm{S}}C_{\mathrm{odd}}(i), \log_{10}\gamma_1(i))$ lie **close to** the straight line.
Class II : the points $(\log_{10}{}^{\mathrm{S}}C_{\mathrm{odd}}(i), \log_{10}\gamma_1(i))$ lie **below** the straight line.
Class III : the points $(\log_{10}{}^{\mathrm{S}}C_{\mathrm{odd}}(i), \log_{10}\gamma_1(i))$ lie **above** the straight line.
Class IV : the points $(\log_{10}{}^{\mathrm{S}}C_{\mathrm{odd}}(i), \log_{10}\gamma_1(i))$ are scattered **above and below** the straight line.
Estrada further argues that Class II is typified by networks with central 'holes' and that Class III is typified by networks with central 'cores'.

5.3.1 Network 1: European Economic Regions

European countries may be broken down into smaller territories [34]. An undirected network consisting of 255 nodes and 580 undirected edges is established by connecting territories that are physically contiguous, i.e. they share a border, therefore a node in the network represents a territory and an edge between two nodes means that they are physically adjacent. For such a network, it may be useful to establish optimal paths between two territories e.g. routes that pass through a minimal number of foreign countries. Similarly, measures of centrality may be of interest to establish which territories are best connected in some sense. Figures 5.1 and 5.2 show plots of some simple measures on the network of contiguous European economic regions.

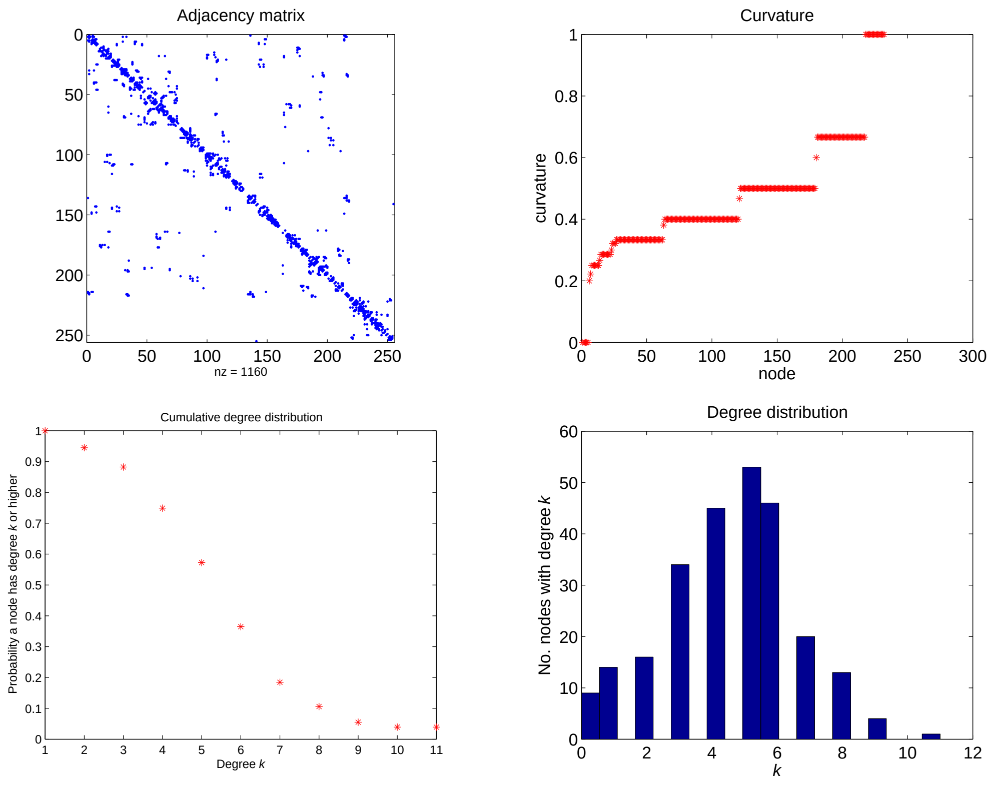


Figure 5.1: Spy-plot, curvature, cumulative degree distribution and degree histogram for the network of European economic regions.

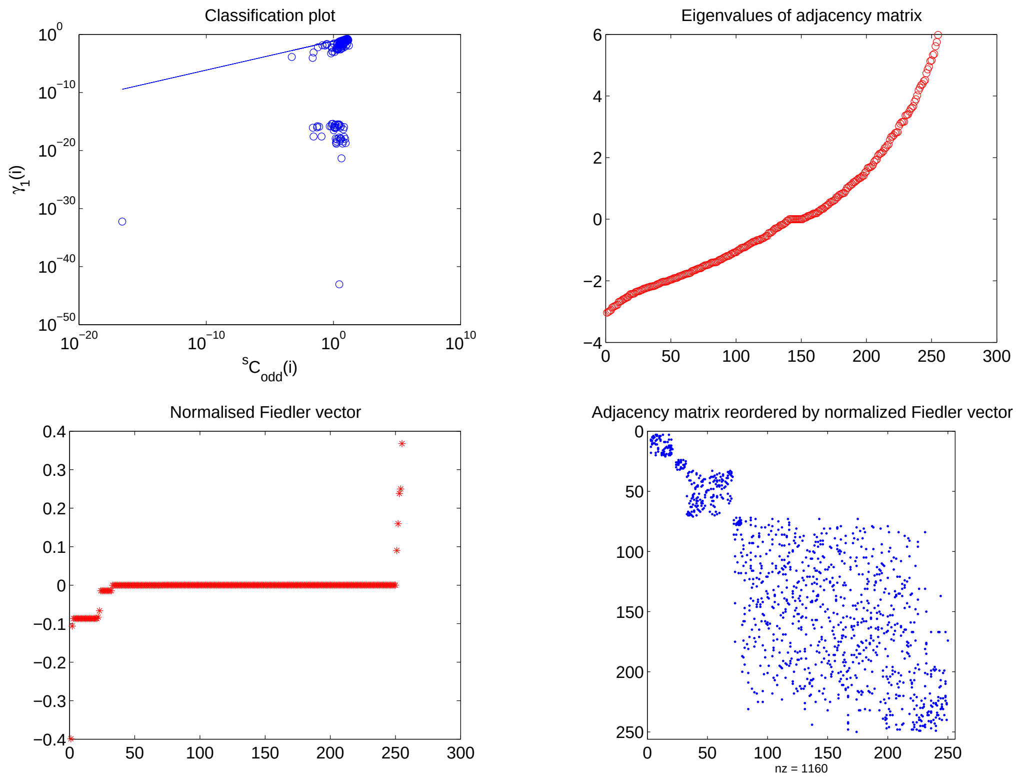


Figure 5.2: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the network of European economic regions.

5.3.2 Network 2: Guppy Social Interactions

This dataset consists of social interactions in a population of guppys [18]. Each node represents a free-ranging guppy and each edge an observed social interaction. The network consists of 99 nodes and 726 undirected edges. Recurring social interactions result in weighted edges and these may be of interest in identifying close-knit communities within the population. Figures 5.3 and 5.4 show plots of some simple measures on the network of guppy social interactions.

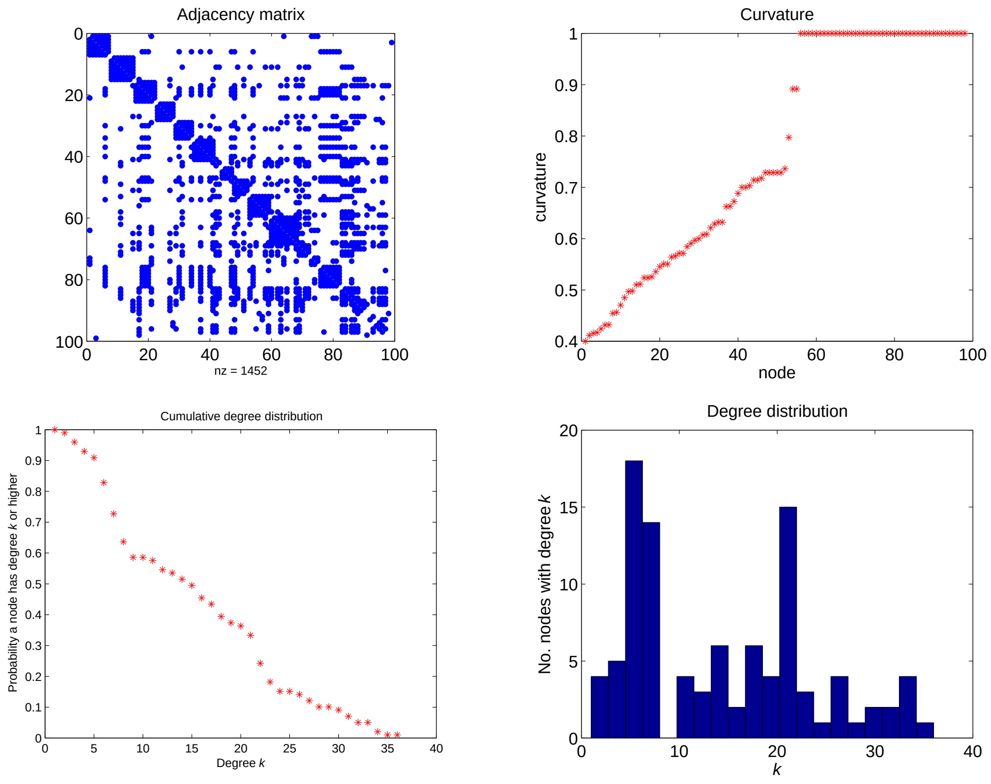


Figure 5.3: Spy-plot, curvature, cumulative degree distribution and degree histogram for the network of guppy social interactions.

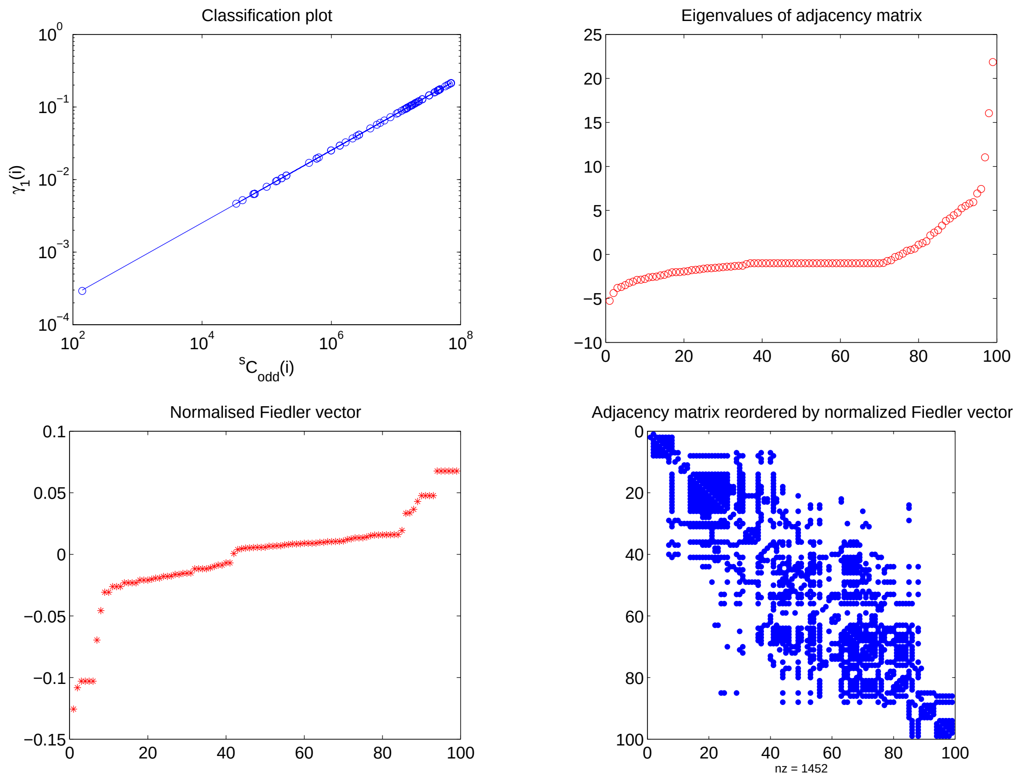


Figure 5.4: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the network of guppy social interactions.

5.3.3 Network 3: Reactor Core Modelling

The function `hexgrid` generates a graph representing the connections between graphite blocks used to encase fuel rods in nuclear reactors [91]. The blocks are modelled as a set of hexagonal tiles arranged in concentric rings, with each node in the network representing an hexagonal block and each edge representing a "keyed connection" between two blocks. The primary aim is to discover how removal of keyed connections influences the modes of movement available to the blocks. Another question is that of symmetry in the network: is it possible, given a pattern of keyed connections to be removed, to eliminate all the analagous cases (i.e. sets of connections whose removal will result in the same modes of movement). Figures 5.5 and 5.6 show plots of some simple measures on network obtained by running the program `hexgrid` with input argument 5 (meaning 5 layers of hexagonal tiles). This results in a network comprising 61 nodes and 192 undirected edges.

5.3.4 Network 4: Classification of Whiskies

86 malt whiskies are scored between 0-4 for 12 different taste categories including sweetness, smoky, nutty etc [93]. Additionally, coordinates of distilleries allow us to obtain pairwise distance information. Using a combination of these datasets it is possible to look for correlations between particular attributes of taste and physical location, for example does a shared local resource have a significant effect on nearby whiskies. By using correlation data it may be possible to provide whisky recommendations based upon an individual's particular preferences. By computing the Pearson correlation coefficient and specifying a threshold value between 0 and 1, we can establish an adjacency matrix where each node is a

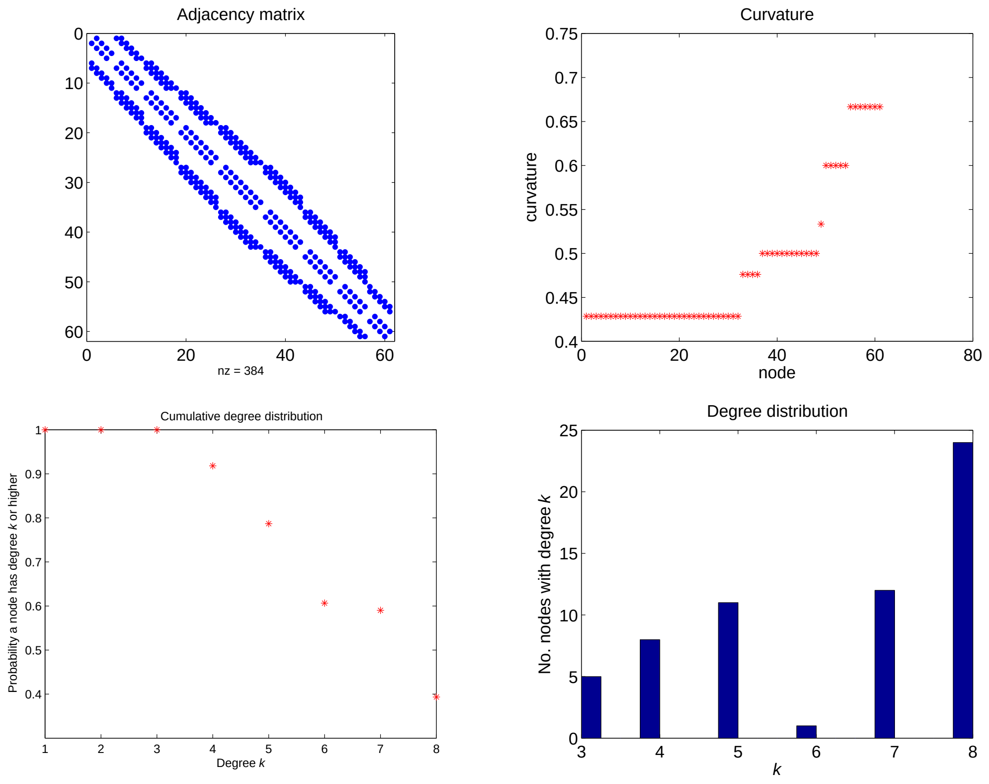


Figure 5.5: Spy-plot, curvature, cumulative degree distribution and degree histogram for the network of reactor components.

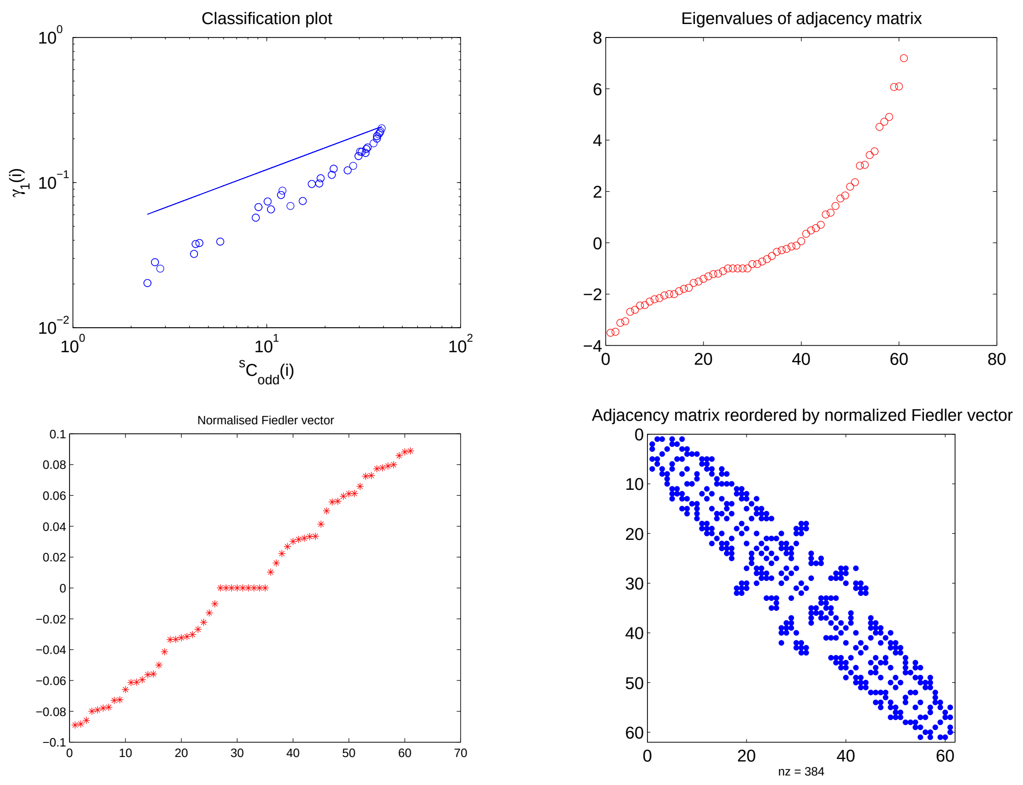


Figure 5.6: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the network of reactor components.

malt whisky and an edge represents a level of similarity above the threshold. By varying the threshold value, the density of nonzeros in the adjacency matrix will change. For instance a high threshold will result in a more sparse adjacency matrix since a higher level of similarity between two whiskies is required to "earn" a nonzero. Figures 5.7 and 5.8 show plots of some simple measures on the network obtained by computing the Pearson correlation coefficient of pairs of whiskies and taking a threshold level of 0.7. This particular network contains 493 undirected connections between 86 nodes.

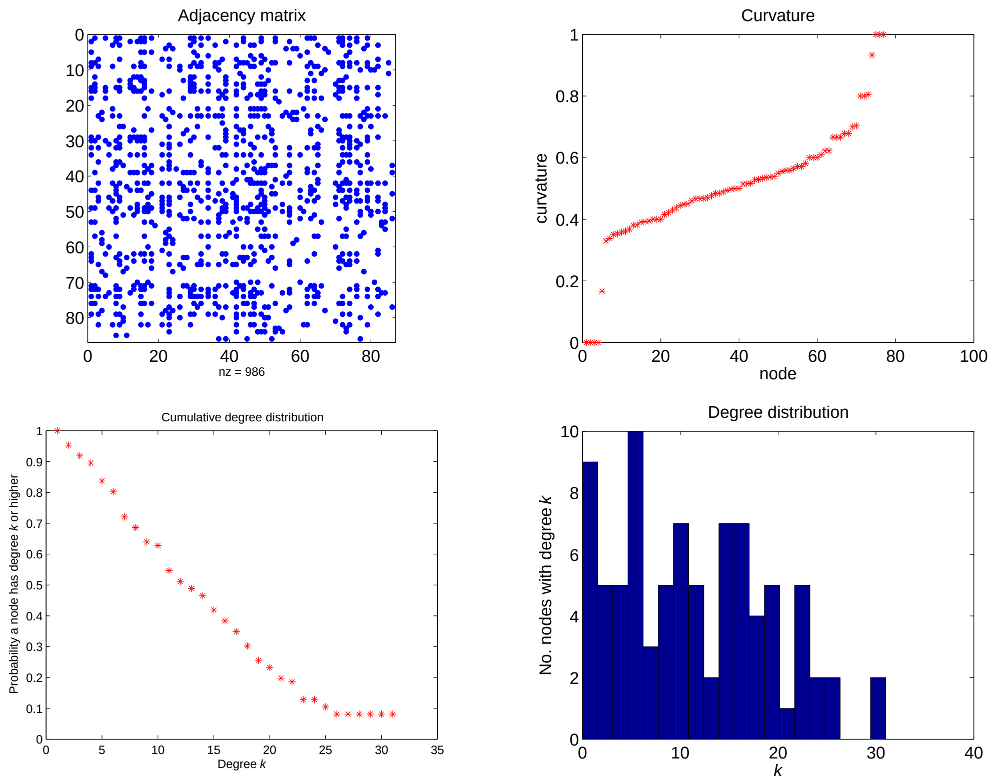


Figure 5.7: Spy-plot, curvature, cumulative degree distribution and degree histogram for the network of malt whisky similarity.

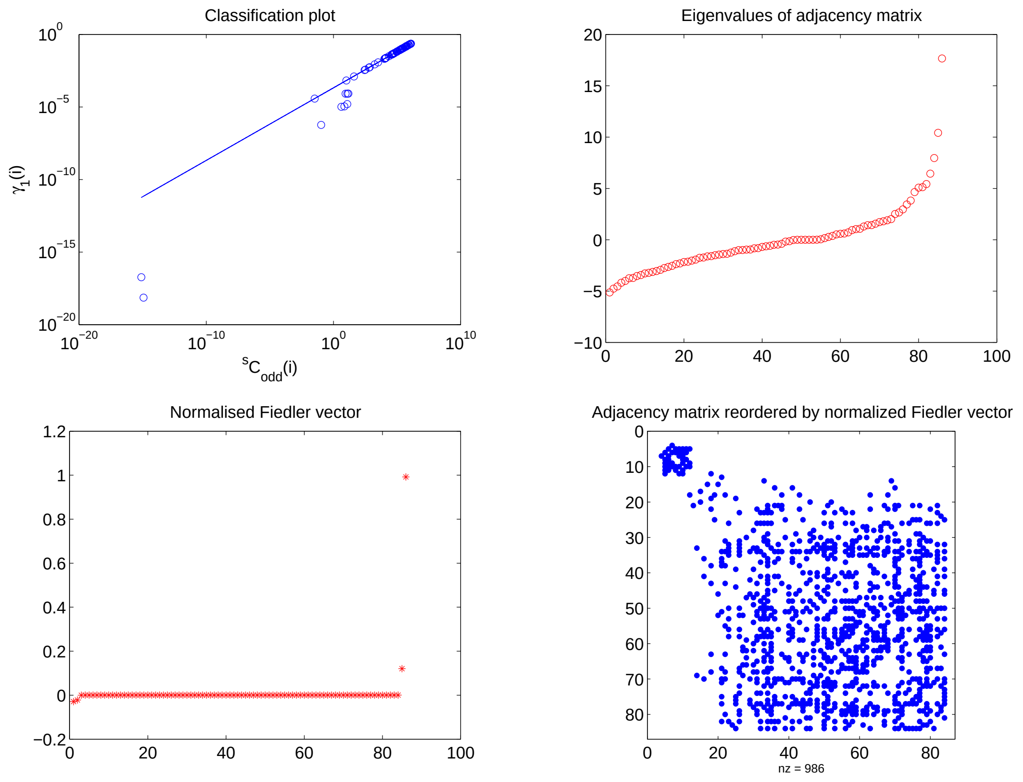


Figure 5.8: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the network of malt whisky similarity.

5.3.5 Network 5: Scottish Football Transfers

Twice annually, Scottish football clubs have an opportunity to transfer players. A list of these transfers is available from

`http://en.wikipedia.org/wiki/Seasons_in_Scottish_football.`

The movement of players between clubs forms a directed graph where each vertex is a football club and each edge represents a transfer in a particular direction. Additionally, this graph may be weighted since multiple players may transfer between two clubs. The transfer fee exchanged could be considered as the weight on each edge although this is complicated by a number of factors. Out of contract players may move between clubs for free, players may be transfered on short-term loan deals and very often transfer fees are not disclosed to the public. This dataset, which lists the transfers to and from Scottish clubs for three consecutive transfer periods, considers only the movement of players between clubs and does not take into account any money involved. Figures 5.9 and 5.10 show plots of some simple measures on the network of Scottish football transfers in the season 08-09. The network has been symmetrized to allow better visualisation. The unsymmetric network for the season 08-09 consists of 242 directed edges and 128 nodes.

5.3.6 Network 6: Scottish Transport Networks

Data regarding journey times between Scottish towns is readily available. This particular dataset comes from

`http://www.transportscotland.gov.uk/.`

Two matrices list typical travel times between Scottish towns, by train or by car. The matrices are non-square, that is to say the data is incomplete. For

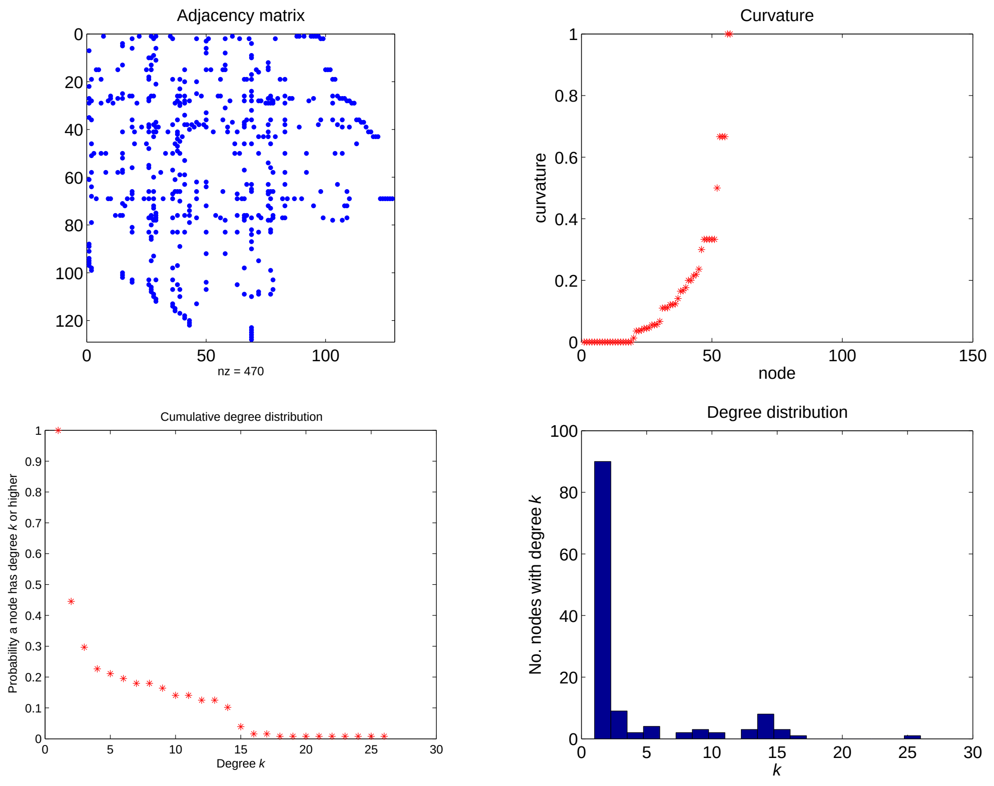


Figure 5.9: Spy-plot, curvature, cumulative degree distribution and degree histogram for the undirected network of Scottish football transfers in the season 08-09.

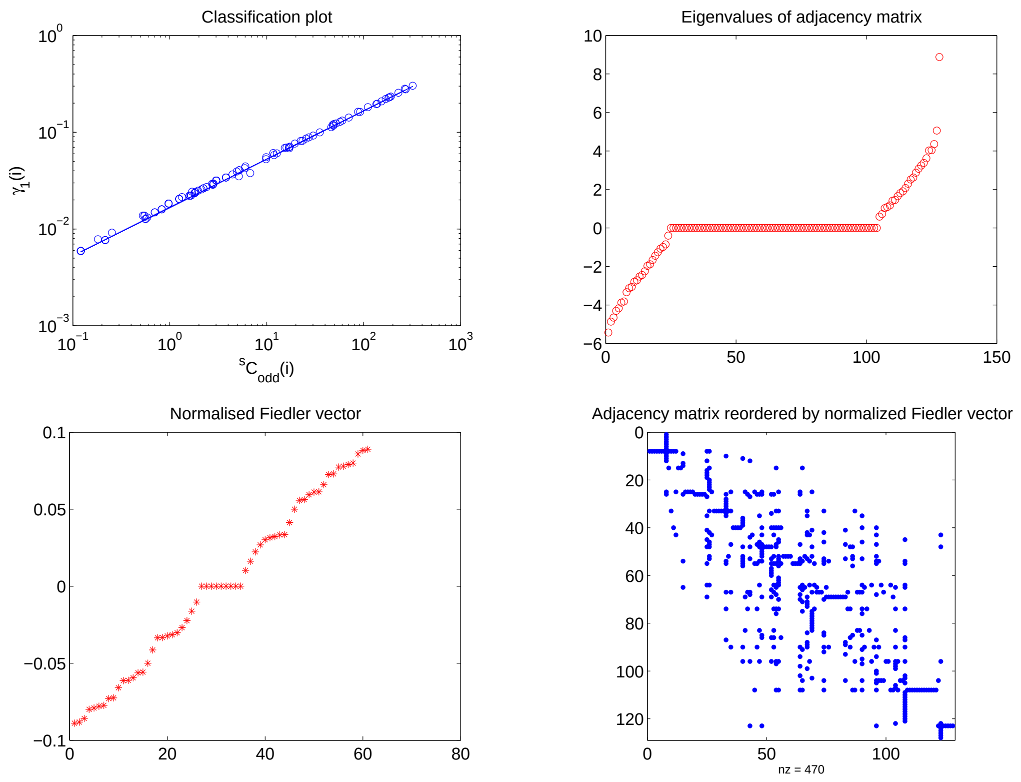


Figure 5.10: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the undirected network of Scottish football transfers in the season 08-09.

instance the travel times from Wick and Tain to Stirling are included but the travel time from Wick to Tain is not. The dataset lends itself to graph-layout problems. Given this limited data, can we find a two dimensional distribution of the towns in question such that the distance between them respects the average travel times? By following the same procedure outlined in Section 5.3.4 we may obtain an undirected adjacency matrix for a given threshold level of “similarity”. In this case, a node represents a town or city and an edge represents a given level of similarity (or inverse-distance) between two towns. Figures 5.11 and 5.12 show plots of some simple measures on the network obtained by computing the Pearson coefficient of pairs of car journey times and taking a threshold level of 0.9. This value yields a network of 25 nodes and 77 undirected edges.

5.3.7 Network 7: Metabolite Network

The nodes are potential chemical formulae obtained by searching databases for formulae with mass with 10ppm of peaks measured in a sample derived from the Trypanosome parasite [78]. Connections are made between two formulae if they differ by one of 80 known transforms. For example, a difference of two Hydrogen atoms suggests the possibility of (de-)hydrogenisation $\pm H_2$. Figures 5.13 and 5.14 show plots of some simple measures on the metabolite network which consists of 376 nodes and 343 edges.

5.3.8 Network 8: p53 Network

A directed network of genes related to the oncogene p53 is obtained by considering pairwise expression levels. An edge is inserted from gene i to j if i expresses significantly above its usual level while j expresses significantly below its usual

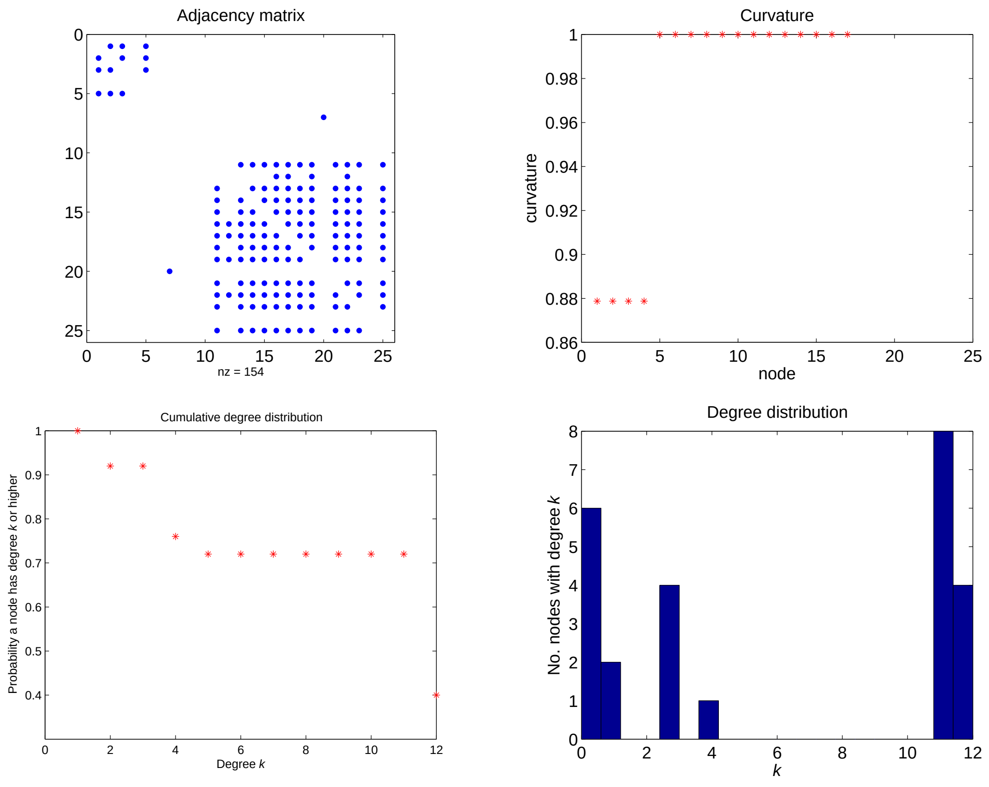


Figure 5.11: Spy-plot, curvature, cumulative degree distribution and degree histogram for the transportation network derived from journey times by car.

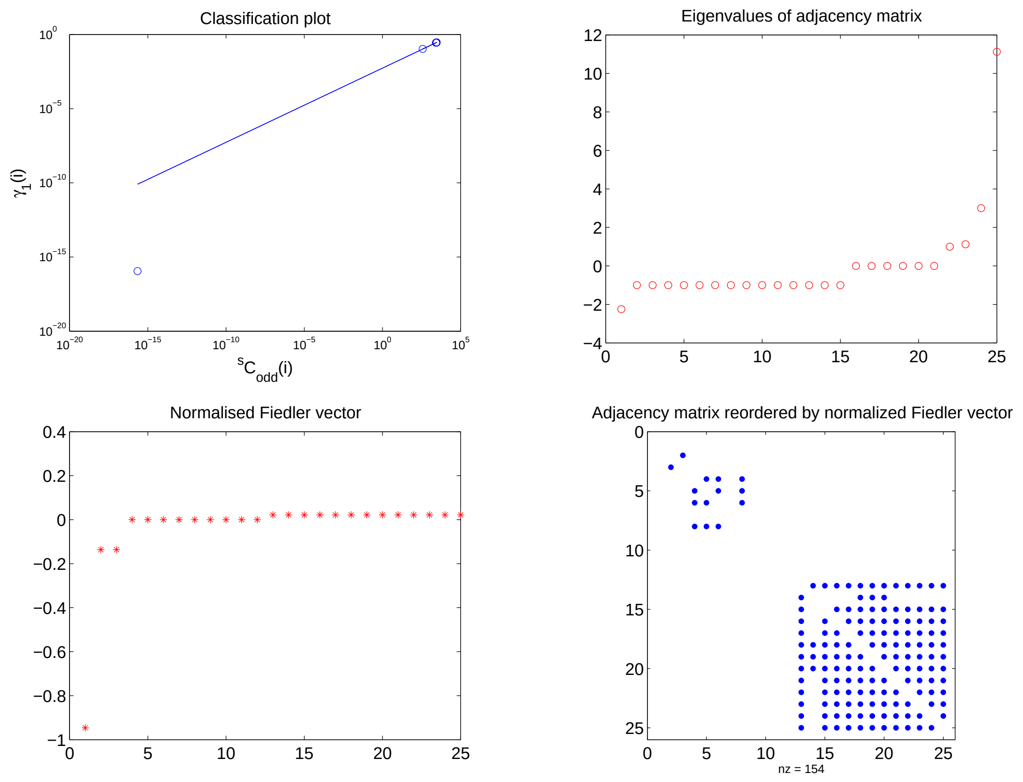


Figure 5.12: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the transportation network derived from journey times by car.

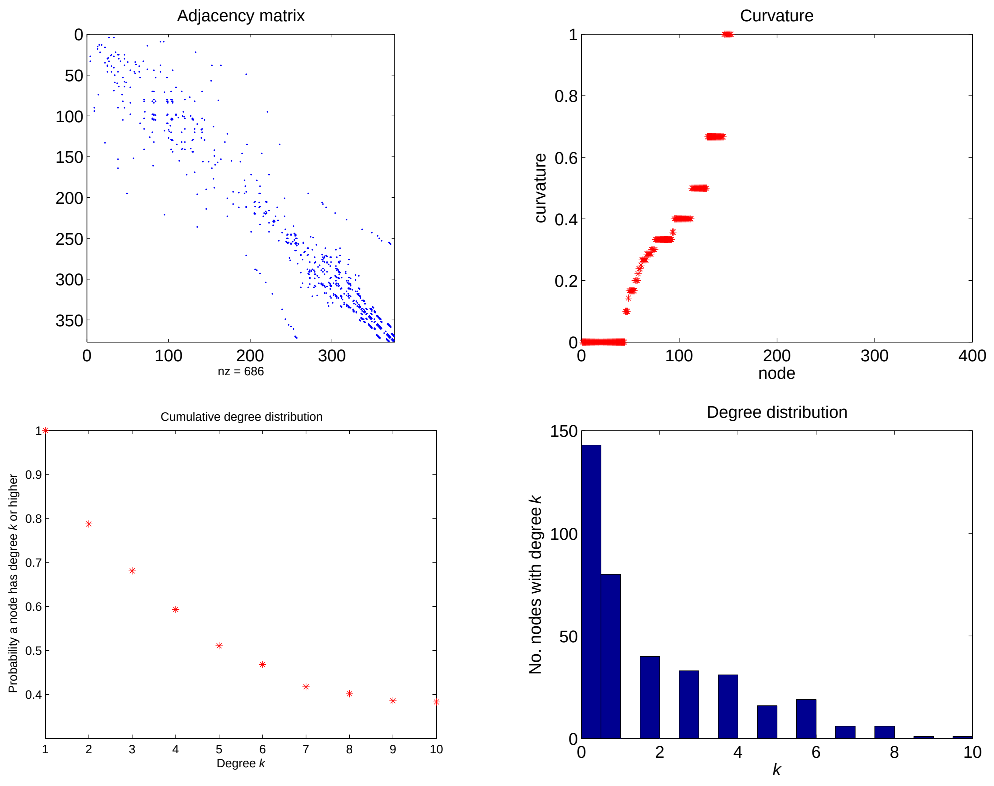


Figure 5.13: Spy-plot, curvature, cumulative degree distribution and degree histogram for metabolite network.

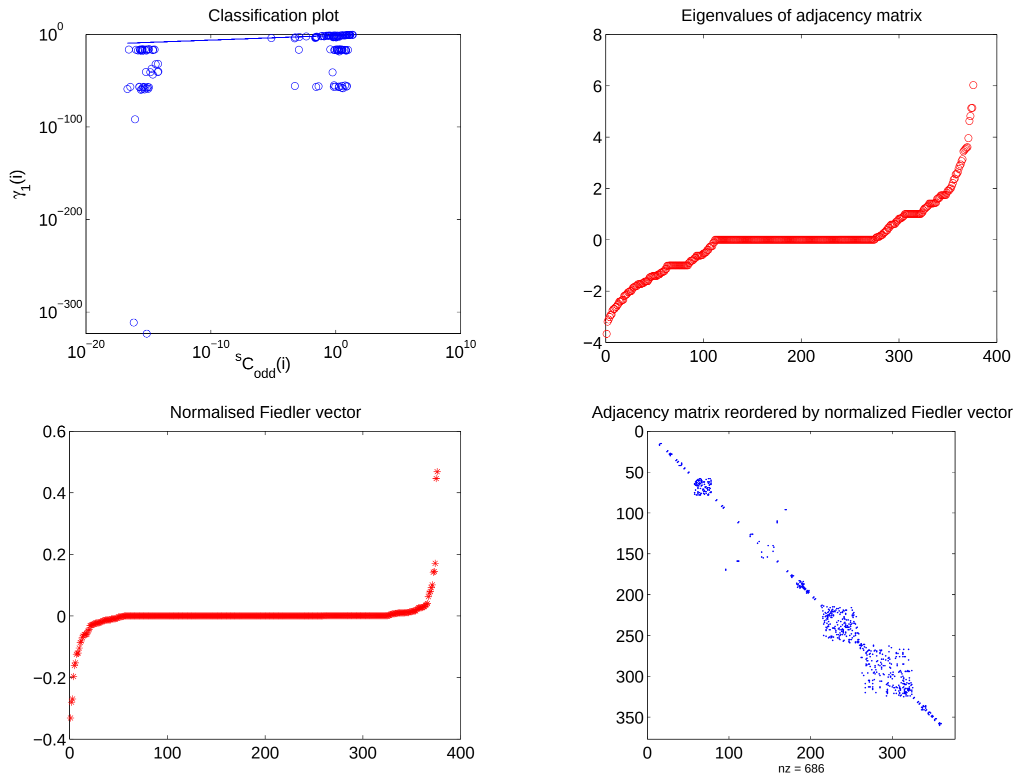


Figure 5.14: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the metabolite network.

level. The resulting 'plus-minus' network is a directed network consisting of 133 nodes (genes) and 558 edges (changes in expression level with opposite polarity) [89]. Figures 5.15 and 5.16 show plots of some simple measures on the directed gene co-expression network. As with the network of football transfers, the data has been symmetrized to allow better visualization.

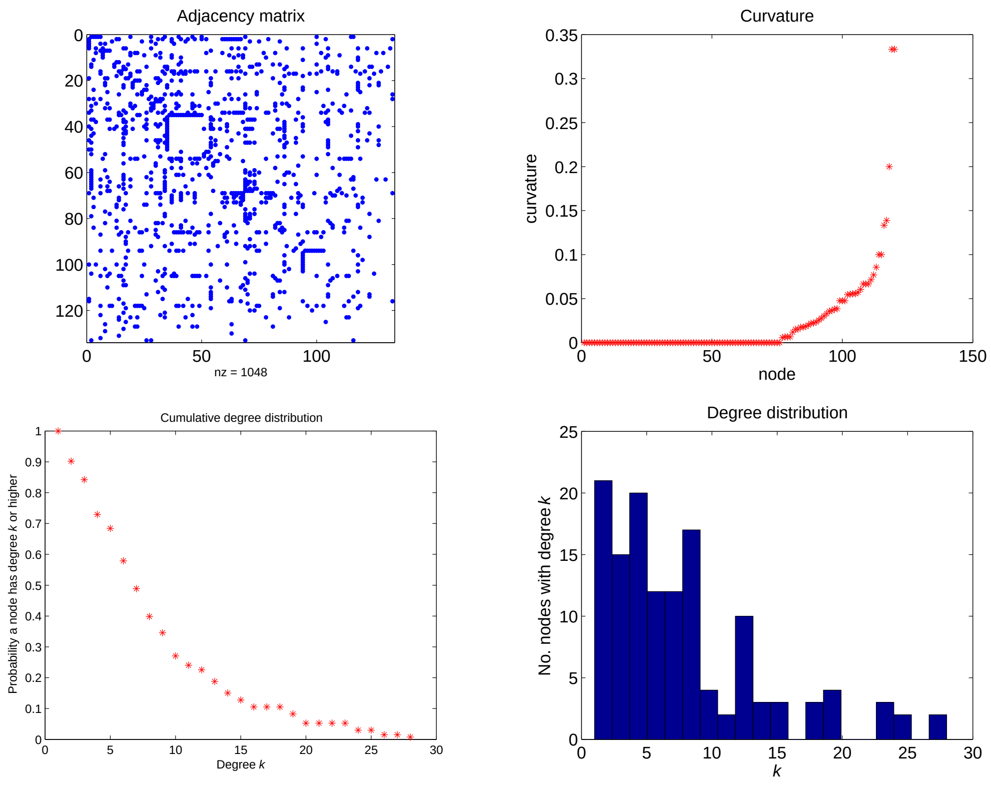


Figure 5.15: Spy-plot, curvature, cumulative degree distribution and degree histogram for the p53 network.

5.3.9 Network 9: Gene Network

Gene expression is typically recorded in a matrix of size $N \times M$ where expression levels of N genes are recorded over M samples [13]. By computing correlation coefficients on such a matrix, we can obtain a square matrix of samples (patients)

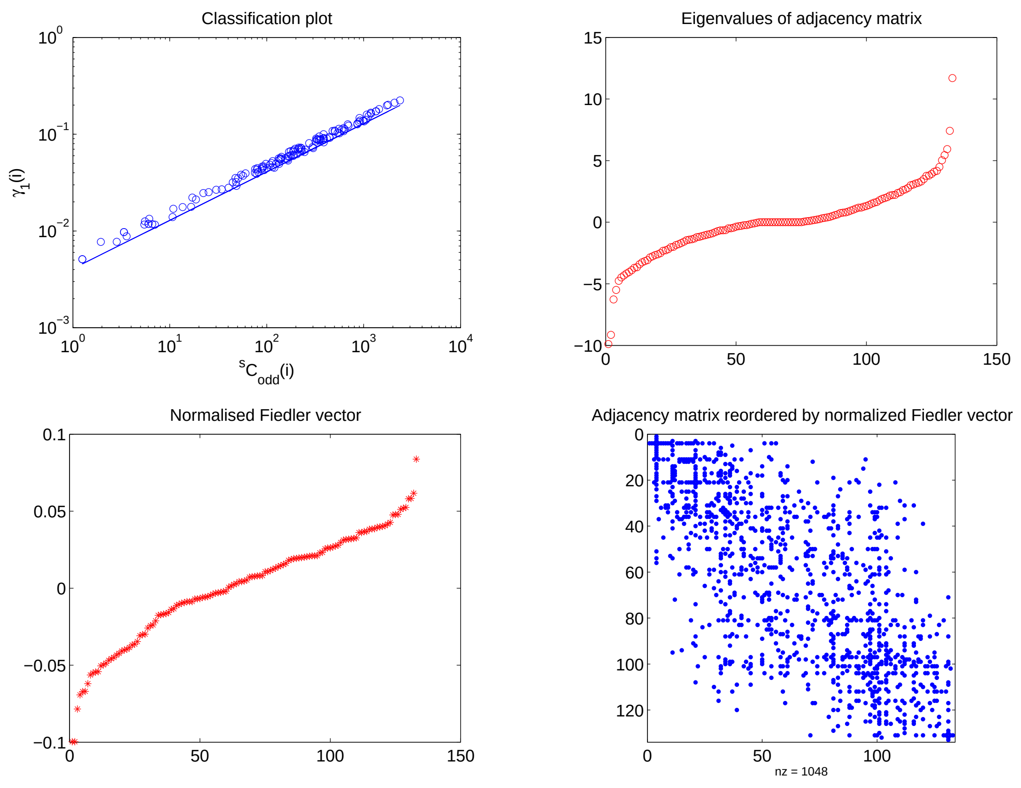


Figure 5.16: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the p53 network.

or genes. For the gene network presented here, we have computed a matrix where each node is a patient and an edge exists between two patients if their gene expression levels yield a correlation coefficient above a threshold value of 0.65. This results in a network of 38 nodes and 180 undirected edges. Applying clustering to such a dataset may allow classification of a new sample into an existing group of patients. In Figures 5.17 and 5.18 we show plots of measures on the gene network.

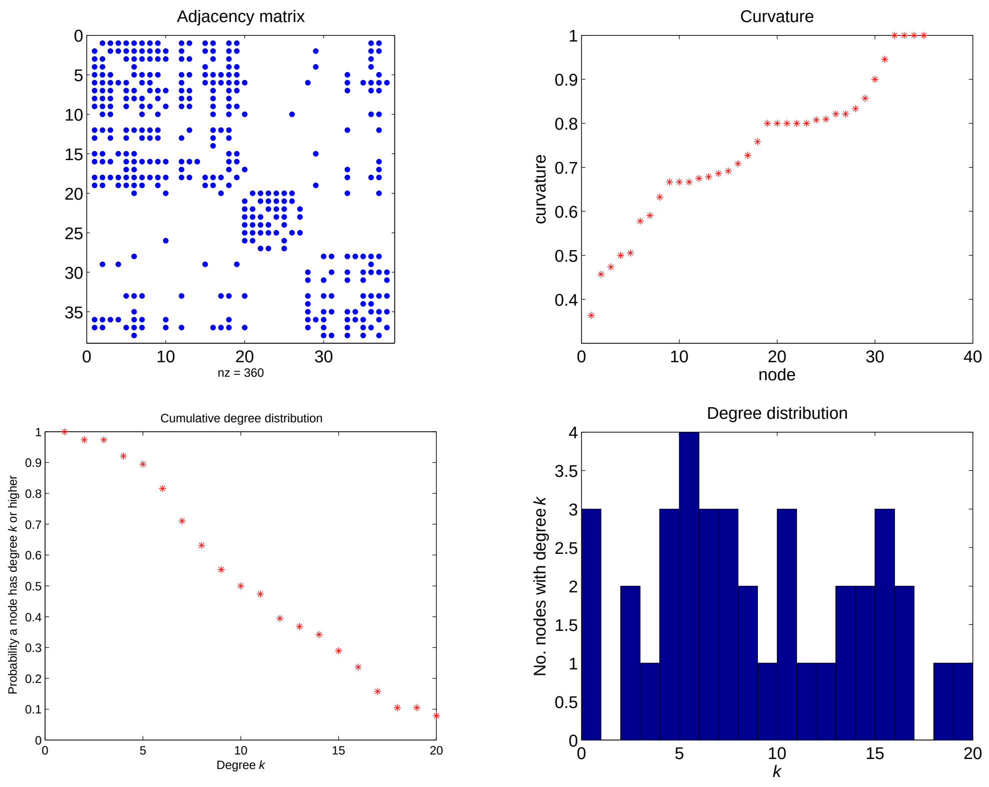


Figure 5.17: Spy-plot, curvature, cumulative degree distribution and degree histogram for the gene network.

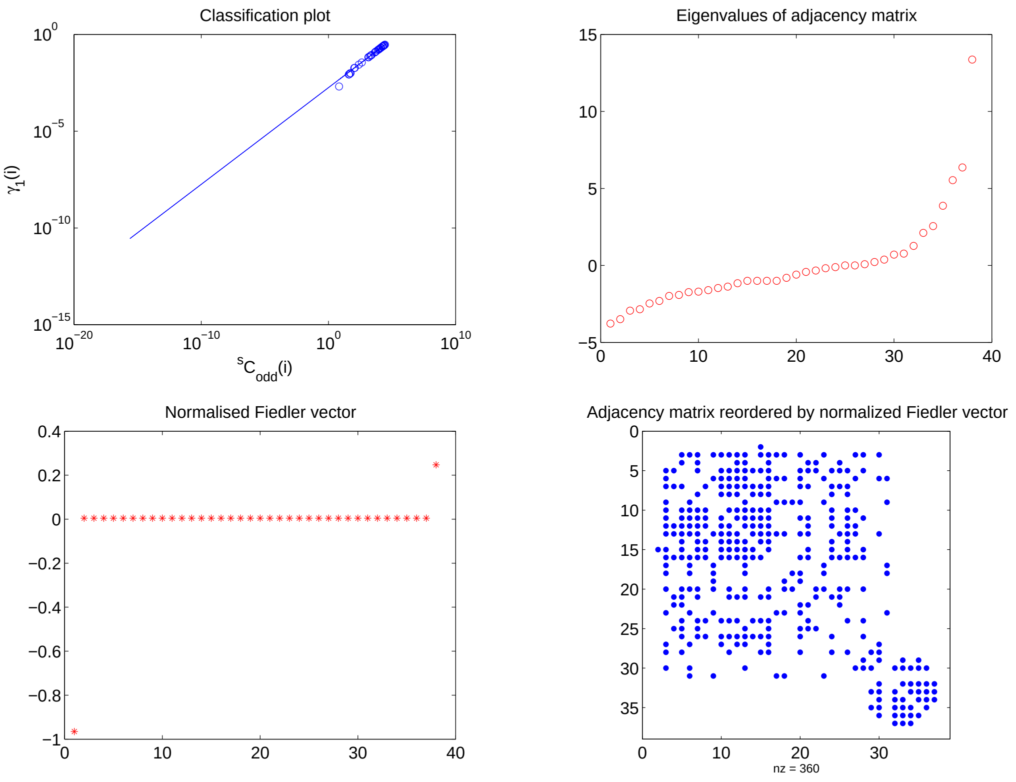


Figure 5.18: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the gene network.

5.3.10 Network 10: Protein-protein Interaction Network

Protein-protein interaction networks consist of observed physical interactions between proteins [54, 86]. Each edge represents a protein and an edge exists between two proteins if they have been observed to interact. Interactions between proteins are bidirectional so the set of interactions between proteins in an organism forms an undirected unweighted network. Protein-protein interaction networks are widely available from repositories such as

`http://www.thebiogrid.org`

and we include here an example of a network taken from yeast consisting of 4388 nodes and 38102 edges (915 of which are self-links). Plots of measures on this network are shown in Figures 5.19 and 5.20.

5.3.11 Network 11: Benguela Marine Ecosystem

A network can be obtained by observing trophic interactions between species. One such network is that of the Benguela ecosystem consisting of species found off the southwest coast of South Africa [95]. In this instance each node represents a particular species, and two species are linked if they interact at the trophic level, i.e. one population impacts upon the size of another. The network considered here consists of 29 nodes and 191 undirected edges. Such networks may be useful if we wish to assess the importance of longer paths in a food-web, i.e. to find out how two species affect each other despite a lack of a direct path between them. We show plots of measures on the Benguela marine ecosystem network in Figures 5.21 and 5.22.

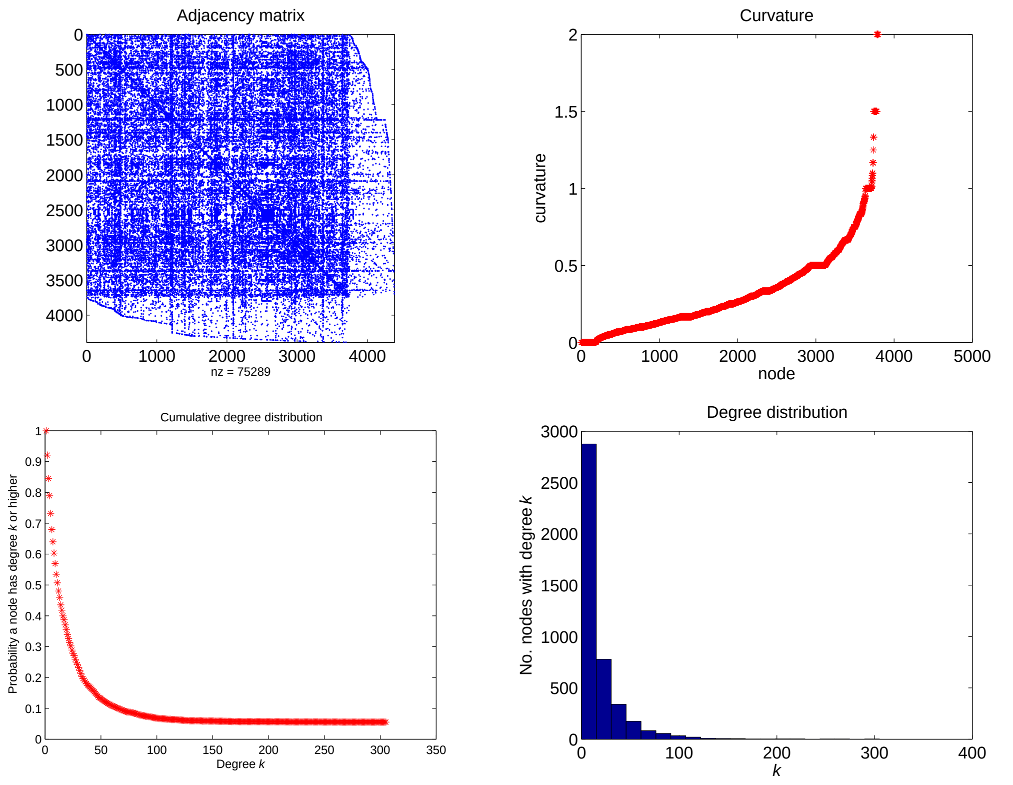


Figure 5.19: Spy-plot, curvature, cumulative degree distribution and degree histogram for the protein-protein interaction network.

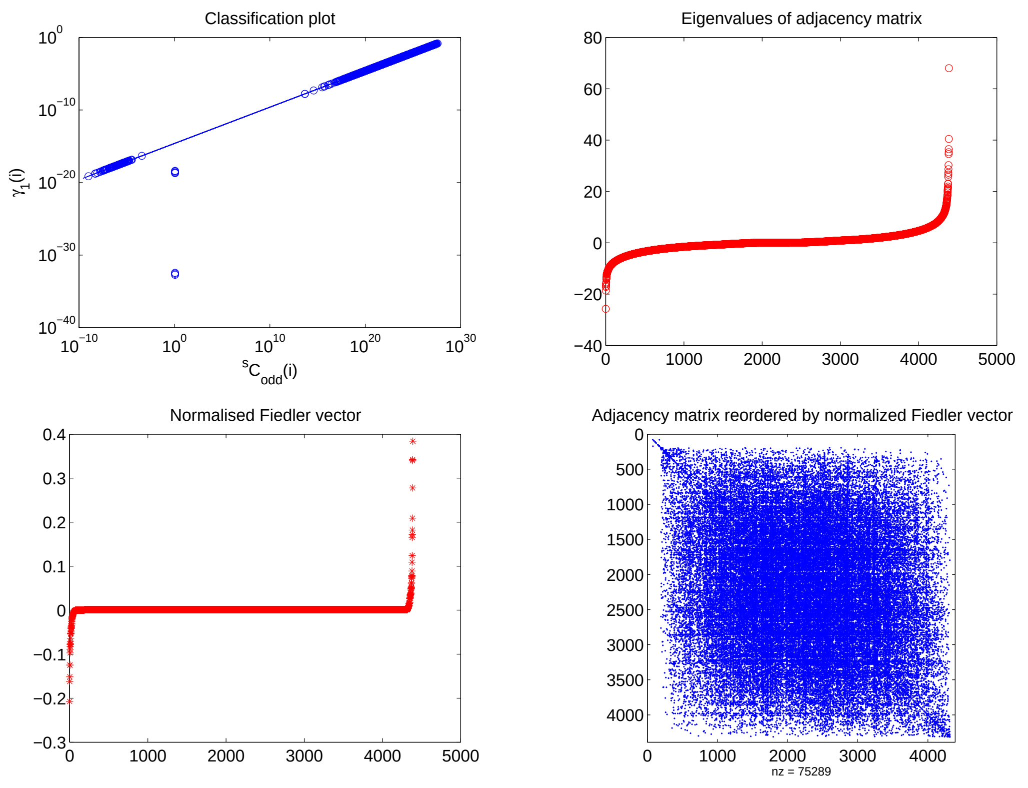


Figure 5.20: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the protein-protein interaction network.

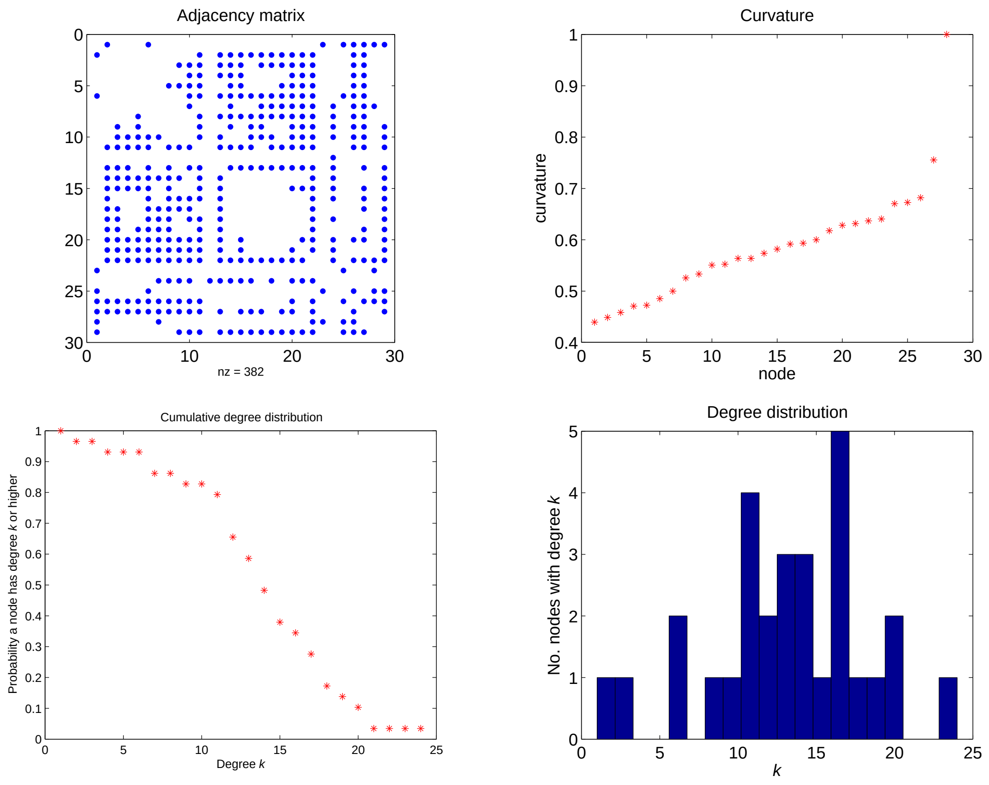


Figure 5.21: Spy-plot, curvature, cumulative degree distribution and degree histogram for the Benguela marine ecosystem network.

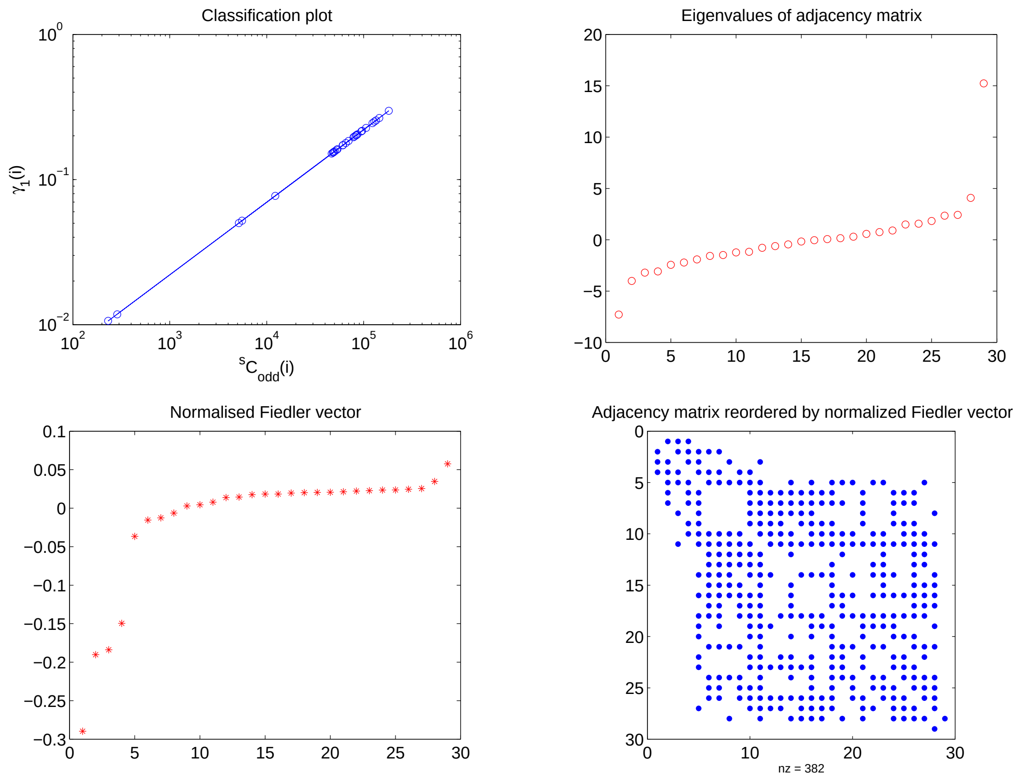


Figure 5.22: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the Benguela marine ecosystem network.

5.3.12 Network 12: US Marine Ecosystem

Similarly we can consider the marine ecosystem of the Northeast US shelf [59]. Once again, nodes represent species and an edge represents an interaction at the approximate trophic level of each species. The network obtained from this ecosystem contains 81 nodes and 1451 undirected edges. In Figures 5.23 and 5.24 we show plots of measures on the US marine ecosystem network.

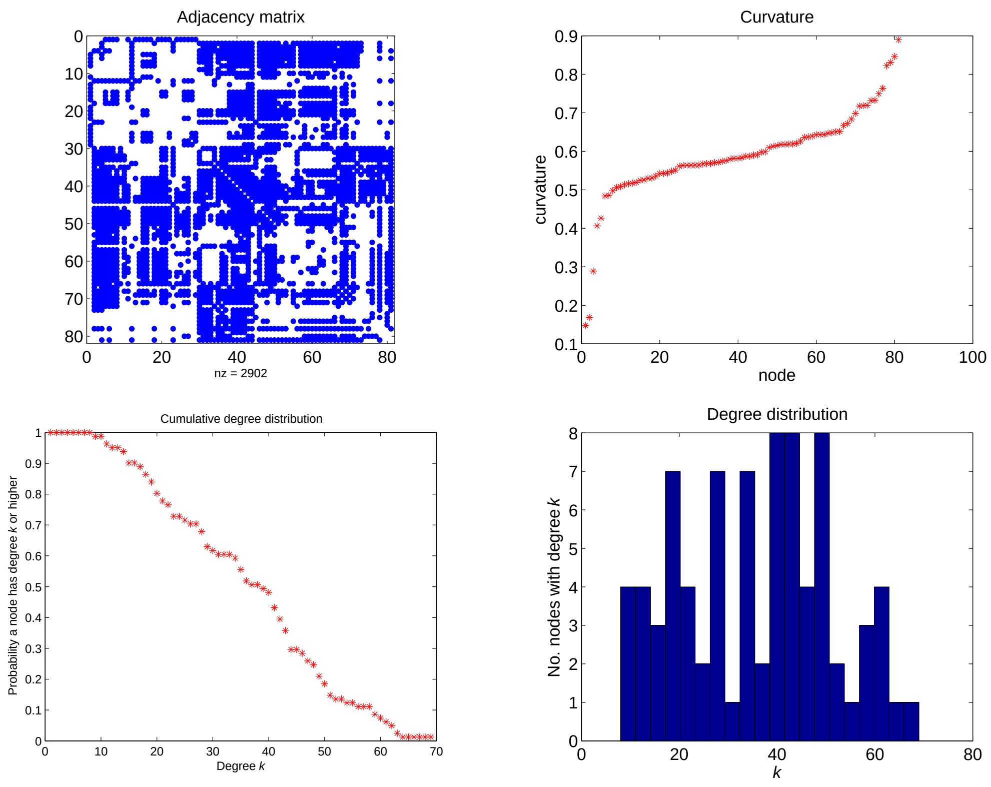


Figure 5.23: Spy-plot, curvature, cumulative degree distribution and degree histogram for the US marine ecosystem network.

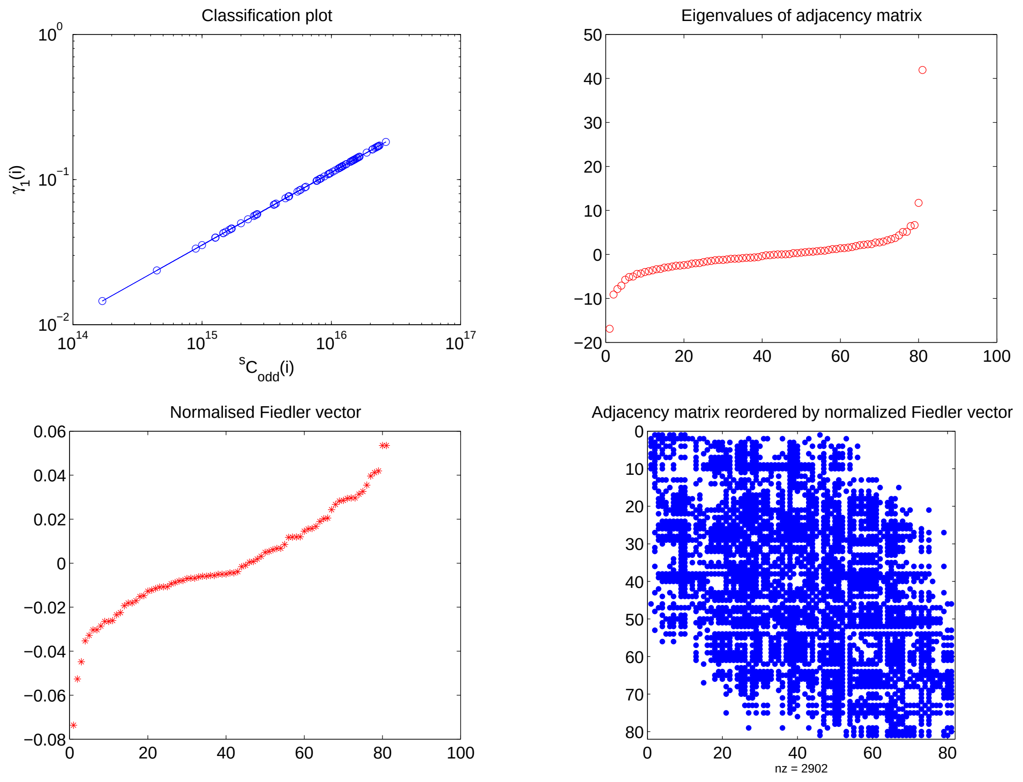


Figure 5.24: Network classification, eigenvalue distribution, normalised Fiedler vector and reordered adjacency matrix for the US marine ecosystem network.

Chapter 6

Discovering Directed Lock and Key Structure

6.1 Motivation

A network, or subnetwork, is said to be bipartite if it can be split into two disjoint groups of nodes such that connections only occur across, but not within, the two groups. Such a structure is obvious in some networks because they consist of objects that naturally fall into two sets. For instance a movie network from a source such as the Internet Movie Database [67] may be constructed where nodes are actors or movies and an edge indicates that an actor appeared in a particular movie. In such a network nodes of one type (e.g. actors) are only connected to other nodes of the same type by walks of even length.

If, however, nodes in a network cannot be easily separated into two such categories, bipartite structure might not be immediately apparent. There has been interest in quantifying the overall level of bipartivity for a network, node or edge [26, 30, 52] but we seek to take a more practical approach of identifying hidden biparite substructures. In reality, perfectly bipartite structures are unlikely to occur in several types of network, since datasets are often contaminated with missing or spurious edges, so we will concern ourselves with attempting to identify approximately bipartite subnetworks.

Existing work in this area has been concerned with protein-protein interaction (PPI) networks where nodes represent proteins and edges denote an observed physical interaction between a pair of proteins [20]. It has been observed that some types of protein-protein interaction occur as a result of complementary binding domains. That is to say that all proteins that possess a particular binding domain should interact with all proteins that possess the complementary domain [83]. In [66], Morrison et al developed an algorithm that aimed to find bipartite substructure in PPI networks relying only on interaction data, i.e. network topology, by taking a spectral approach which was shown to be robust in the presence of noise. A related approach was taken in [29] where the negative matrix exponential was interpreted as a count of odd and even length walks. This approach has the advantage that it allows the whole network to be partitioned into quasi-bipartite communities.

This work differs from previous studies by considering networks with *directed* edges - a connection from node i to node j may not necessarily have a reciprocal connection from node j to node i. Our aim is to develop an approach for identifying *directed bipartite substructure*, that is, groups of nodes S_1 and S_2 such that edges point from nodes in S_1 to nodes in S_2 but not in the opposite direction. We will show that spectral information is still relevant if we generalize from eigenvalues and eigenvectors to singular values and singular vectors. We develop our theoretical arguments in Section 6.2, present meaningful test cases in Sections 6.3 and 6.4 and finally apply the approach to a larger network arising from cancer microarray data in Section 6.5 as well as a smaller one from sociology in Section 6.6.

6.2 Theory

Using standard notation, we denote a directed network with N nodes by the unsymmetric adjacency matrix $A \in \mathbb{R}^{N \times N}$, where $a_{ij} = 1$ if there is a link from node i to node j and $a_{ij} = 0$ otherwise. To characterize directed bipartite subnetworks, we find it useful to borrow the *lock and key* analogy that was introduced in [66]. We suppose that locks and keys are distributed among the nodes in a network. We further suppose that each lock and key has a particular colour (red, blue, green, ...) and that lock-key matches, corresponding to connections in the network, take place only when the colours agree. Suppose now that two sets of nodes, S_1 and S_2, form a directed bipartite subnetwork so that edges between these nodes only point from nodes in S_1 to nodes in S_2. We may relate this to the lock and key analogy by imagining that S_1 consists of all the nodes that possess a key of a particular colour, say red, and that S_2 consists of all the nodes that possess the corresponding red lock.

We note that the reference [66] dealt only with undirected edges, whereas this work is concerned with the directed case. The concept of locks and keys here is slightly different, and perhaps more natural, and we find that the arguments in support of a spectral algorithm are stronger.

We focus for now on this particular red lock-key subnetwork. We may introduce a pair of indicator vectors $\mathbf{u}^{\text{red}}, \mathbf{v}^{\text{red}} \in \mathbb{R}^N$ such that

$$(\mathbf{u}^{\mathrm{red}})_i \quad = \quad \begin{cases} 1 & \text{if node } i \text{ has the red key,} \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

and

$$(\mathbf{v}^{\mathrm{red}})_i \quad = \quad \begin{cases} 1 & \text{if node } i \text{ has the red lock,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

It follows immediately that the edges arising from red lock-key interactions may be characterized through the outer product $\mathbf{u}^{\mathrm{red}}(\mathbf{v}^{\mathrm{red}})^T$. If we let

$$\text{keyred} := ||\mathbf{u}^{\mathrm{red}}||_2^2$$

and

$$\text{lockred} := ||\mathbf{v}^{\mathrm{red}}||_2^2$$

denote the total number of red keys and red locks, respectively, then this outer product may be written

$$\sqrt{\text{keyred} \times \text{lockred}}\, \hat{\mathbf{u}}^{\mathrm{red}}(\hat{\mathbf{v}}^{\mathrm{red}})^T,$$

where $\hat{\mathbf{u}}^{\mathrm{red}} := \mathbf{u}^{\mathrm{red}}/||\mathbf{u}^{\mathrm{red}}||_2$ and $\hat{\mathbf{v}}^{\mathrm{red}} := \mathbf{v}^{\mathrm{red}}/||\mathbf{v}^{\mathrm{red}}||_2$ are unit vectors. More generally, when all edges arise through key-lock interactions, the adjacency matrix for the network may be expanded as

$$
\begin{aligned}
A \;=\; & \operatorname{sign}\Big(\sqrt{\text{keyred} \times \text{lockred}}\,\hat{\mathbf{u}}^{\text{red}}(\hat{\mathbf{v}}^{\text{red}})^T \\
& +\sqrt{\text{keyblue} \times \text{lockblue}}\,\hat{\mathbf{u}}^{\text{blue}}(\hat{\mathbf{v}}^{\text{blue}})^T \\
& +\sqrt{\text{keygreen} \times \text{lockgreen}}\,\hat{\mathbf{u}}^{\text{green}}(\hat{\mathbf{v}}^{\text{green}})^T + \cdots\Big),
\end{aligned} \tag{6.3}
$$

where the sign function deals with the possibility of multiple key-lock matches; node i may have both a red and blue key whilst node j has both a red and blue lock.

The sign function in 6.3 is not needed if we make the following assumption.

Assumption A: each node has at most one key and one lock.

Note that this assumption permits a node to possess a key of one colour and a lock of another colour, or a lock and key of the same colour. A second important consequence of assumption A is that the key indicator vectors $\{\mathbf{u}^{\text{red}}, \mathbf{u}^{\text{blue}}, \mathbf{u}^{\text{green}}, \cdots\}$ form an orthogonal set and the lock indicator vectors $\{\mathbf{v}^{\text{red}}, \mathbf{v}^{\text{blue}}, \mathbf{v}^{\text{green}}, \cdots\}$ form an orthogonal set. In this case, we see that the expansion 6.3 has the same form as the singular value decomposition (SVD) [38]

$$
A = \sum_{k=1}^{N} \sigma_k \mathbf{u}^{[k]} \mathbf{v}^{[k]T}, \tag{6.4}
$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_N \geq 0$ are the singular values of A and $\{\mathbf{u}^{[k]}\}_{k=1}^{N}$ and $\{\mathbf{v}^{[k]}\}_{k=1}^{N}$ are the corresponding left and right singular vectors, respectively. We conclude that under Assumption A the SVD can be used to discover directed bipartite subgraphs - the square of the singular value, σ_k^2, indicates the product of the number of locks and keys of the kth colour, the nonzero entries of $\mathbf{u}^{[k]}$

give the key locations and the nonzero entries of $\mathbf{v}^{[k]}$ give the lock locations.

We show now that there is a complementary way to motivate the use of the SVD. This approach, which is based on the ideas in [66] that were used for undirected networks, also goes some way towards allowing for false negatives among the edges. Under Assumption A, suppose that node i does not possess the red key. Then multiplying the ith row of the adjacency matrix into the red lock indicator vector will give a value zero; there will be no matches in the inner product. On the other hand, if node i possesses the red key then multiplying the ith row of the adjacency matrix into the red lock indicator vector will count the number of red locks in existence - each red lock will take part in one nonzero term. Suppose now that there are some 'errors' in the network in the form of missing edges. More precisely, suppose that only a fixed proportion $\theta \in (0, 1)$ of the red key-lock matches are recorded as edges. Then generalizing the argument above we have

$$(A\mathbf{v}^{[k]})_i = \sum_{j=1}^{N} a_{ij} v_j^{[k]} = \begin{cases} \theta\text{lockred} & \text{if node } i \text{ has the red key,} \\ 0 & \text{otherwise,} \end{cases}$$

which may be written

$$A\hat{\mathbf{v}}^{\text{red}} = \theta\sqrt{\text{lockred} \times \text{keyred}}\,\hat{\mathbf{u}}^{\text{red}}. \tag{6.5}$$

Similarly, we find that

$$A^T \hat{\mathbf{u}}^{\text{red}} = \theta \sqrt{\text{lockred} \times \text{keyred}}\, \hat{\mathbf{v}}^{\text{red}}. \tag{6.6}$$

The relations 6.5 and 6.6 show that $\hat{\mathbf{u}}^{\text{red}}$ and $\hat{\mathbf{v}}^{\text{red}}$ correspond to left and right singular vectors of A, respectively, with singular value $\theta\sqrt{\text{lockred} \times \text{keyred}}$. Of course, when $\theta = 1$, we recover the singular value expression $\sqrt{\text{lockred} \times \text{keyred}}$ that we derived earlier via the argument involving rank one outer products. However, it is worth noting that 6.5 and 6.6 require assumption A to hold only for nodes with red locks or keys. The other nodes in the network could be connected in any way. So the SVD will reveal isolated substructure of this type hidden within any complex network.

In summary, we have shown that if a directed network can be broken down into isolated or non-overlapping bipartite subnetworks, even when a fixed proportion of edges are missing, then the left and right singular vectors of A will reveal which nodes take part in which subnetwork, and the singular values tell us how many nodes are involved.

Eigenvectors and, more generally, singular vectors, enjoy important variational properties, and hence the information they convey tends to be robust to the presence of noise. This has been confirmed experimentally [6, 28, 42, 80]. In particular, for the case of undirected edges, it was shown in [66] that the SVD can find approximate bipartite subgraphs in both synthetic and real networks. In the next section, we test the robustness of the SVD in the directed network setting.

6.3 Testing

6.3.1 Detailed Example

We begin with a detailed investigation of the synthetic network shown in Figure 6.1. By design this network does not completely satisfy Assumption A from Section 6.2. The colour coding of the arrows emphasizes the distribution of keys and locks. Nodes 1-5 have the red key and nodes 6, 11-15 and 17 the corresponding red lock. Nodes 4 and 6-10 have the blue key and nodes 16-20 the corresponding blue lock. Finally, nodes 19 and 20 have the green key, while nodes 9 and 10 have the green lock. We note that this node ordering was chosen simply to make the output easier to interpret - results from the SVD are invariant under symmetric row and column permutations.

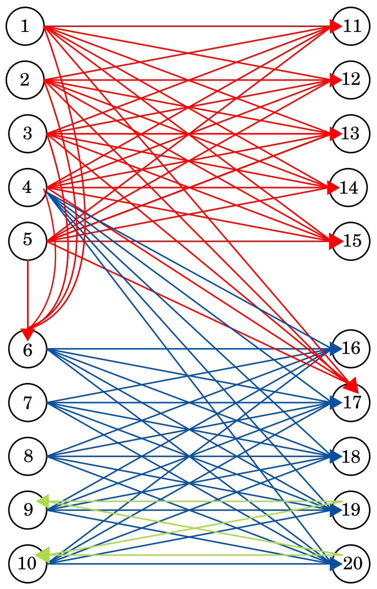


Figure 6.1: Synthetic network with three types of key-lock pairings.

The nontrivial singular values are, to two digits, 6.5, 4.7, 2.0 and 0.7, which is consistent with our expectation that the first, second and third pair of singular vectors should correspond to the red, blue and green groups. We now consider pairs of singular vectors to determine how effectively they reveal the underlying key-lock distribution.

6.3.2 First Left and Right Singular Vectors

Figure 6.2 shows the components of the first left and right singular vectors of the adjacency matrix in increasing order. On the horizontal axis are the indices, so, for example, the most negative left and right singular vector entries correspond to nodes 4 and 17 respectively.

For the first left singular vector, $\mathbf{u}^{[1]}$, there are two main groups of nodes with components away from zero; one with values around -0.34 and another with values around -0.23. There is also an outlying vertex at around -0.5. The group at height -0.34 involves nodes 1, 2, 3 and 5, which, as we see from Figure 6.1, share the red key. The outlier is node 4. This is the only other red key node, but it also has the blue key.

The first right singular vector, $\mathbf{v}^{[1]}$, splits up the network in a similar fashion. Nodes 6, 11, 12, 13 and 14 form a clear group and we see from Figure 6.1 that they share the red lock. The outlier is node 17. This is the only other red lock node, but it also has the blue lock. We note that node 6 differs from its neighbours in Figure 6.2 in that it also has a blue key, but the left singular vector has not been affected by this - node 6 is placed at the same height as the purely red lock nodes. This is consistent with the theory in Section 6.2, where Assumption A does not rule out the case of a node having a key and lock of different colours.

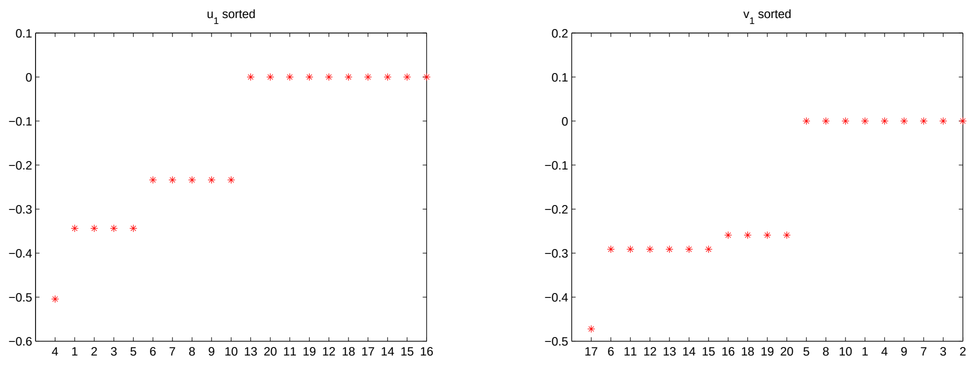


Figure 6.2: First left and right singular vectors for network in Figure 6.1.

In Figure 6.3 we show the adjacency matrix for the subgraph created by nodes 4, 1, 2, 3, 5, 6, taken from the left hand end of $\mathbf{u}^{[1]}$ up to the cut-off from -0.34 to -0.23, and nodes 17, 6, 11, 12, 13, 14, 15, taken from the left hand end of $\mathbf{v}^{[1]}$ up to the corresponding cut-off. We see that there is a clear two-by-two block checkerboard structure, corresponding to a directed bipartite subgraph, with node 6 having an extra link to its red lock colleague (arising from the separate blue lock-key connection).

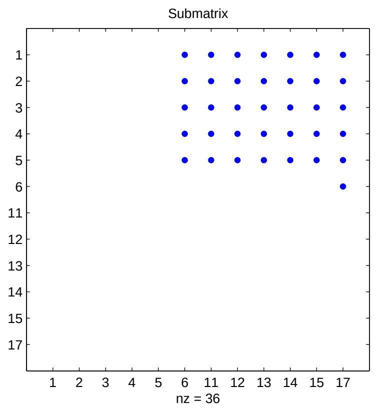


Figure 6.3: Subgraph showing red lock-key interactions arising from nodes 4, 1, 2, 3, 5 and 17, 6, 11, 12, 13, 14, 15 taken from $\mathbf{u}^{[1]}$ and $\mathbf{v}^{[1]}$ in Figure 6.2.

Similarly, Figure 6.4 reveals the blue lock-key interactions by plotting the adjacency matrix arising from nodes 6, 7, 8, 9, 10 and 16, 18, 19, 20 that were grouped together above the main cut-offs in Figure 6.2. Here, node 17 is missing from the blue lock group; this makes sense because it is also a lock member of the larger red lock-key group. In Figure 6.4 we also see that the green lock-key group of 9, 10 and 19, 20 appears as a block in the adjacency matrix. However, from Figure 6.2 this appears to be a coincidental feature caused by the fact that these nodes are also part of the blue subgraph - the components of nodes 9, 10 in $\mathbf{u}^{[1]}$ and 19, 20 in $\mathbf{v}^{[1]}$ are not visually distinguishable from their blue key and lock neighbours, 6, 7, 8 and 16, 18 respectively.

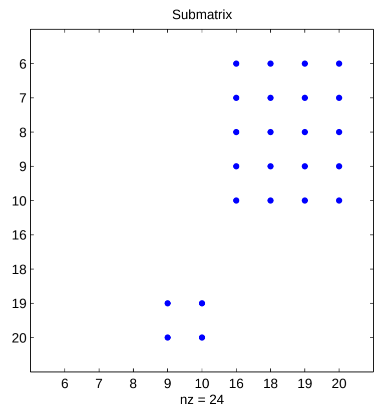


Figure 6.4: Subgraph showing red lock-key interactions arising from nodes 4, 1, 2, 3, 5 and 17, 6, 11, 12, 13, 14, 15 taken from $\mathbf{u}^{[1]}$ and $\mathbf{v}^{[1]}$ in Figure 6.2.

Overall, the first left and right singular vectors have done an excellent job of sorting out the key and lock nodes, respectively, for the red group. They also made a reasonable delineation of the blue group, but the ambiguous nodes that shared red and blue characteristics were placed next to their red colleagues, which form the dominant group.

6.3.3 Second Left and Right Singular Vectors

Figure 6.5 shows the second left and right singular vectors. In $\mathbf{u}^{[2]}$ we see that nodes 1, 2, 3, 5 are grouped together. These are four out of the five red key nodes. Node 4, which also has the blue key, has been given a positive value, in line with the positivity of nodes 6, 7, 8, 9, 10, which complete the blue key group and are classified together. In $\mathbf{v}^{[2]}$ we see nodes 6, 11, 12, 13, 14, 15 grouped together. These are all red lock nodes, with the exception of node 17 which has been given a positive value in line with the other blue lock nodes 16, 18, 19, 20 that are classified together. Figures 6.6 and 6.7 show the subgraphs obtained from the negative and positive extreme values of the second singular vectors, respectively.

So, overall, the second singular vectors also give information about both red and blue groups, but they favour the second-largest, blue group.

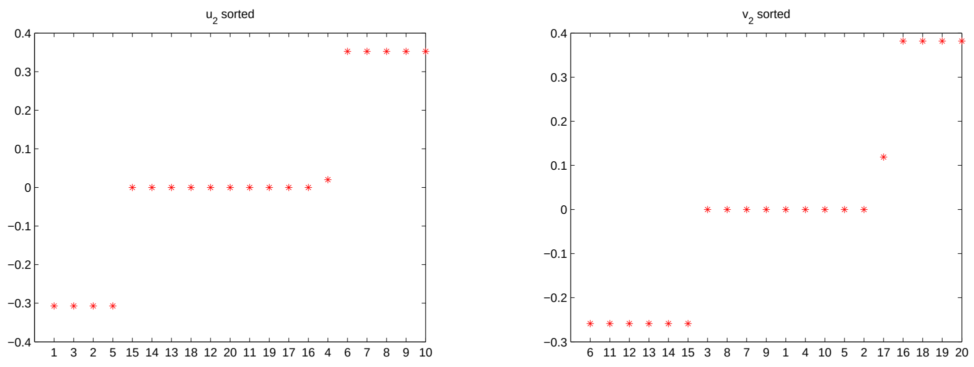


Figure 6.5: Second left and right singular vectors for network in Figure 6.1.

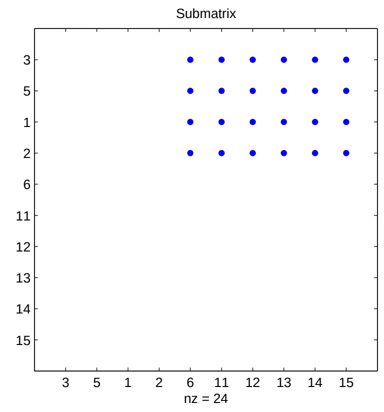


Figure 6.6: Subgraph showing red lock-key interactions arising from nodes $3, 5, 1, 2$ and 6, 11, 12, 13, 14, 15 taken from $\mathbf{u}^{[2]}$ and $\mathbf{v}^{[2]}$ in Figure 6.5.

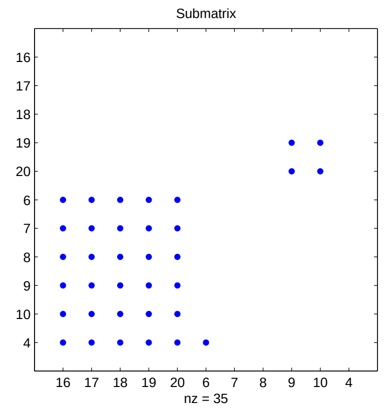


Figure 6.7: Subgraph showing blue lock-key interactions arising from nodes $16, 17, 18, 19, 20$ and $6, 7, 8, 9, 10, 4$ taken from $\mathbf{u}^{[2]}$ and $\mathbf{v}^{[2]}$ in Figure 6.5.

6.3.4 Third Left and Right Singular Vectors

Figure 6.8 shows the third left and right singular vectors. In this case we see that the green keys, 19, 20 and the green locks, 9, 10 have been picked out unambiguously. The resulting subgraph is shown in Figure 6.9. Although this subgraph is clearly symmetric, we note that reciprocal links belong to a different ‘colour’ of key-lock interaction.

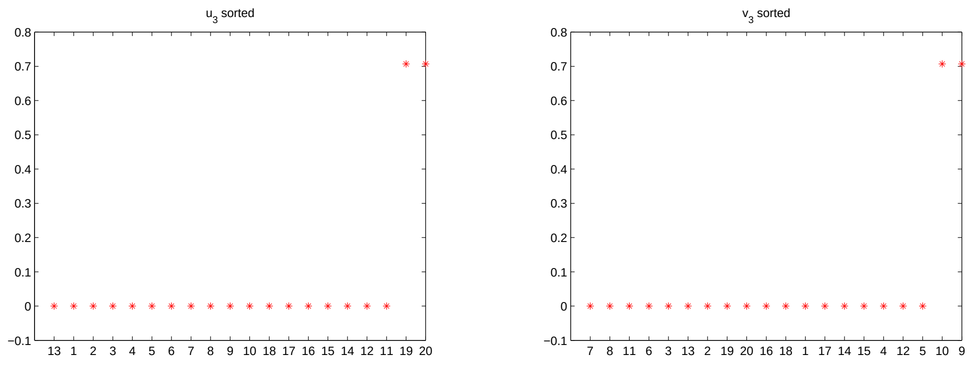


Figure 6.8: Third left and right singular vectors for network in Figure 6.1.

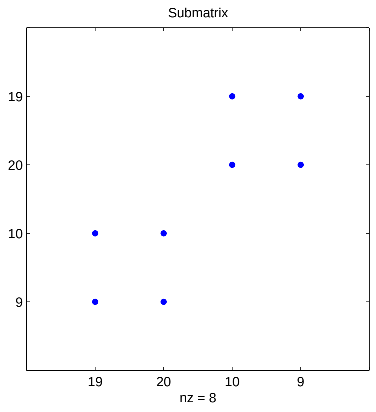


Figure 6.9: Subgraph showing green lock-key interactions arising from nodes 19, 20 and 9, 10 taken from $\mathbf{u}^{[3]}$ and $\mathbf{v}^{[3]}$ in Figure 6.8.

6.3.5 Parameterized Examples

Our second experiment tests the robustness of the SVD approach when spurious and missing edges contaminate the directed bipartivity. We constructed a network of 50 nodes with nodes 1-10 forming group S_1 and nodes 11-25 forming group S_2. We formed links independently at random such that the probability of a link from node i to node j is given by 0.6 if $i \in S_1$ and $j \in S_2$, and p_2 otherwise.

To the left in Figure 6.10 we show the adjacency matrix that arose for $p_2 = 0.1$. We see that the $S_1 \rightarrow S_2$ block forms a dense patch, but there is a significant amount of non-bipartite 'noise'. In fact there are 96 $S_1 \rightarrow S_2$ edges and 214 others. In the centre and right of Figure 6.10 we show the first left and right singular vectors. It is clear that the key group, S_1, is picked out by $\mathbf{u}^{[1]}$ and the lock group, S_2, is picked out by $\mathbf{v}^{[1]}$; in each case, the group members appear sequentially, taking the extreme values in the vector.

In Figure 6.11, we increase p_2 to 0.3. Now there are 691 edges outwith the $S_1 \rightarrow S_2$ block. In this case we are at the extremes of the noise level that the SVD can tolerate. In $\mathbf{u}^{[1]}$, the 10 nodes in group S_1 appear in positions 1, 2, 3, 4, 5, 6, 7, 8, 18, 26 as we search through the components with largest to smallest absolute value. Similarly, in $\mathbf{v}^{[1]}$, the 15 nodes in group S_2 appear in positions 1, 2, 3, 4, 8, 9, 10, 11, 12, 16, 17, 22, 24, 28, 48. So the dominant singular vectors do not reproduce perfectly the key and lock groups; although at this high level of noise it may be argued that the groups are not clearly defined.

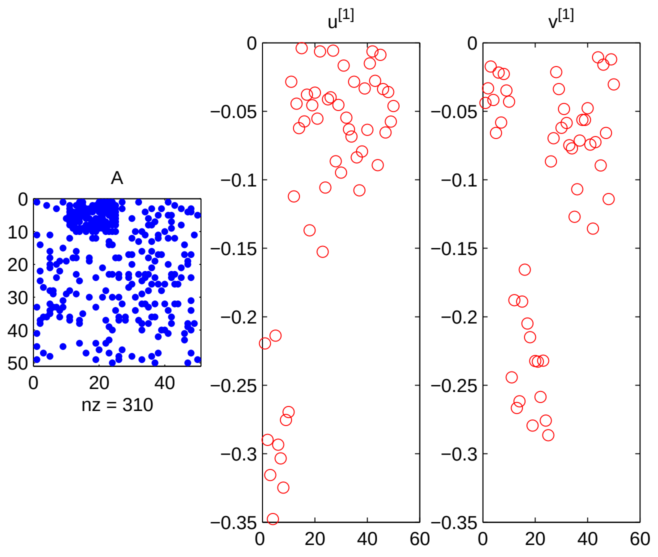


Figure 6.10: Left: adjacency matrix. Middle: first left singular vector. Right: first right singular vector. Here $p_2 = 0.1$ for the non-bipartite connectivity probability.

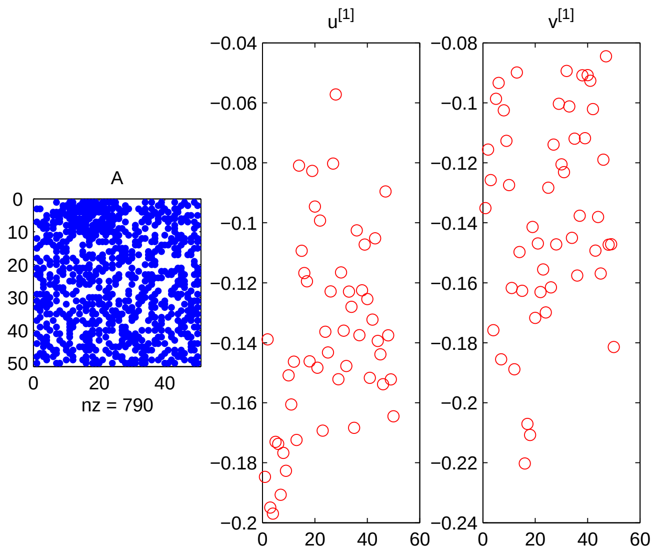


Figure 6.11: Left: adjacency matrix. Middle: first left singular vector. Right: first right singular vector. Here $p_2 = 0.3$ for the non-bipartite connectivity probability.

6.4 Statistical Testing

Having established that the left and right singular vectors may be useful in identifying members of directed bipartite communities, we would like to try and assess the success with which nodes are classified into subgroups. To this end we construct the following synthetic test. We construct an adjacency matrix consisting of 100 nodes with sets S_1 and S_2 comprising nodes 1-10 and 11-20 respectively. We connect pairs of nodes i and j independently at random with probability p_1 if $i \in S_1$ and $j \in S_2$ and with probability p_2 otherwise.

With an adjacency matrix set up in this fashion, we may compute the SVD and examine the components of the first left and right singular vectors. If the SVD perfectly organises nodes into the appropriate subgroups, then nodes 1-10 should correspond to the ten components with highest absolute value in $\mathbf{u}_1$ and nodes 11-20 should correspond to the ten components with highest absolute value in $\mathbf{v}_1$. We fix p_1 and vary p_2 from 0 to p_1, generating several instances of the synthetic network described above for each value of p_2. The SVD of each adjacency matrix is calculated and the proportion of correctly identified nodes in the first ten positions of the relevant singular vector is recorded in each instance.

In Figure 6.12 we show a plot of the mean proportion of correctly identified nodes for $p_1 = 0.9$ and p_2 varying from 0 to 0.9 in increments of 0.01. The blue line shows the mean proportion of “key” nodes correctly identified and the red line corresponds to the mean proportion of “lock” nodes correctly identified. In this case the probability of false negatives in the region of bipartite connectivity is 0.1, and the SVD is reasonably tolerant of false positives elsewhere in the adjacency matrix. The singular vectors correctly identify more than half of the correct nodes until the probability of a false positive in a given position in the adjacency matrix reaches around 0.3.

Figure 6.13 shows a plot of the same type with $p_1 = 0.6$. In this case, the singular vectors successfully identify more than half of the corect nodes until the probability of a false positive in a given position in the adjacency matrix reaches around 0.15.

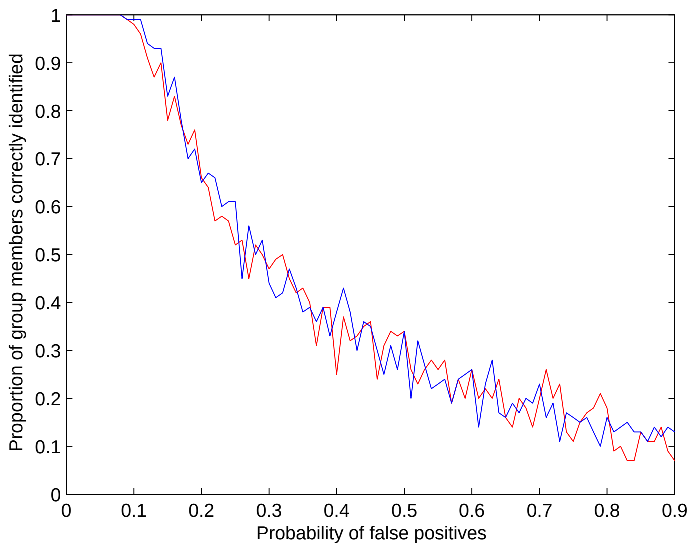


Figure 6.12: Proportion of 'key' nodes (red) and 'lock' nodes (blue) correctly identified by first singular vectors for $p_1 = 0.9$ with varying levels of 'noise'.

By varying p_1 from 0 to 1 and carrying out the procedure described above, we can obtain three dimensional plots of the proportions of locks and keys correctly recovered by the SVD for varying probabilities of false negatives and false positives. The procedure carried out for each value of p_1 is identical to the one described above, with the addition of a random row and column permutation of the adjacency matrix at the beginning of each run. This ensures that the 'keys' do not simply correspond to the first ten nodes and that the 'locks' do not correspond to the next ten.

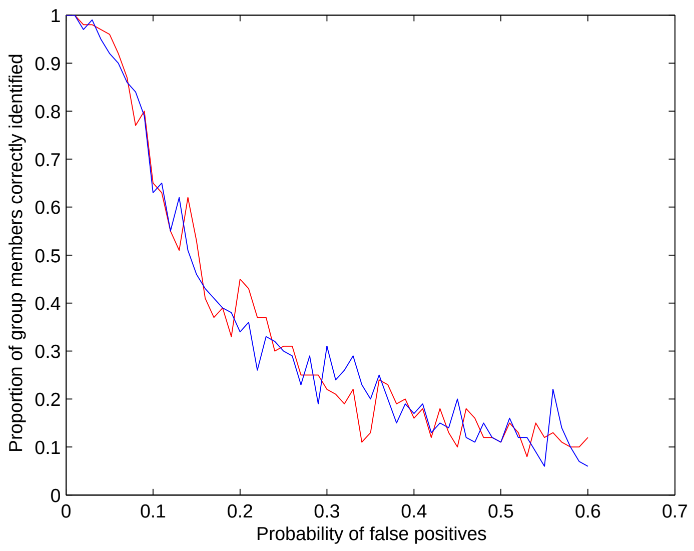


Figure 6.13: Proportion of 'key' nodes (red) and 'lock' nodes (blue) correctly identified by first singular vectors for $p_1 = 0.6$ with varying levels of 'noise'.

Figure 6.14 shows the proportion of 'key' nodes correctly recovered for varying values of p_1 and p_2. The data is also plotted as a heat map in Figure 6.15. Similarly, Figures 6.16 and 6.17 show the proportion of 'lock' nodes correctly recovered as a surface plot and a heat map respectively. By inspection we see that the left and right singular vectors perform very similarly in their identification of 'lock' and 'key' nodes. Approximately half the correct nodes are identified if the probability of a false positive is around a third of the probability of a 'true positive', and all the correct nodes are identified if every true connection is present and the probability of a false positive is less than 0.2.

It is important to stress that the success of the SVD in identifying members of directed bipartite subgroups is dependent upon a number of factors, including the proportion of 'noise' in the data (i.e. the frequency of false positives and false negatives), the number of lock-and-key pairs (including overlapping group membership) and the size of a directed bipartite community relative to the dataset as a whole.

Having attempted to quantify the success of the SVD in a number of test cases, we now apply our method to a dataset from genomics and attempt to uncover meaningful communities.

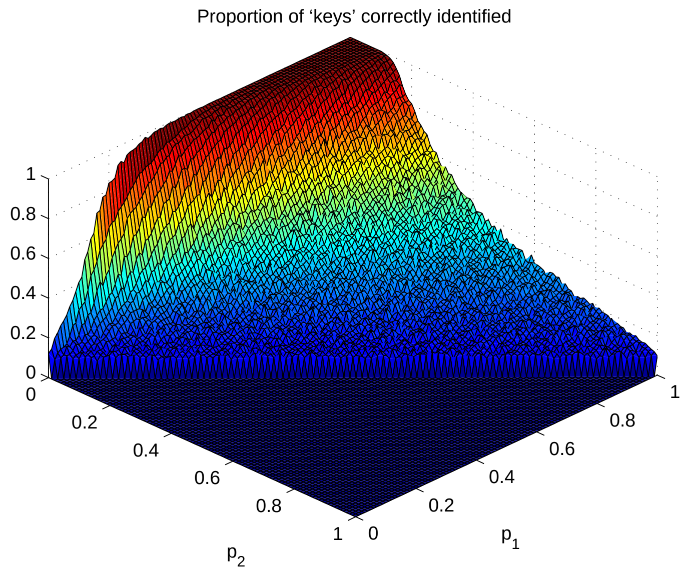


Figure 6.14: Proportion of 'key' nodes correctly identified by first left singular vector for varying levels of 'noise'.

6.5 Application to Cancer Microarray Data

In this section we consider a network arising through gene expression. Cancer microarray data from [87] was treated with the classification method developed in [89]. More precisely, we selected 133 genes related to the oncogene p53, and computed the 'plus-minus' network. Here, an edge between nodes i and j indicates that when gene i expresses significantly above its usual level, gene j

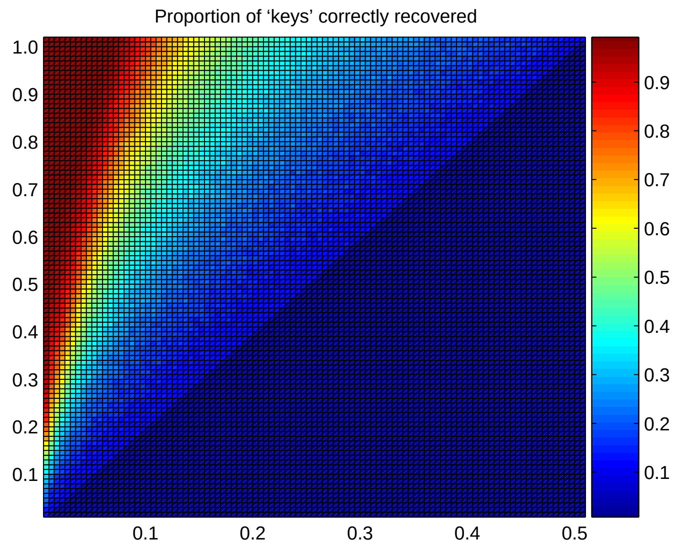


Figure 6.15: Proportion of 'key' nodes correctly identified as a heat map.

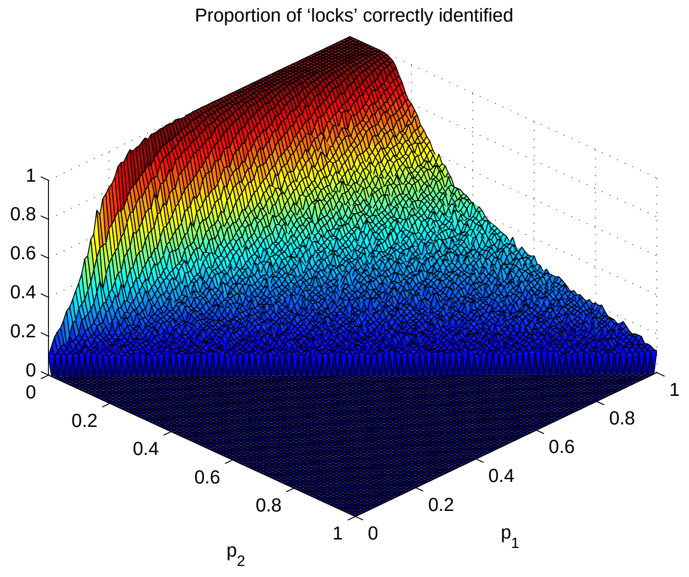


Figure 6.16: Proportion of 'lock' nodes correctly identified by first right singular vector for varying levels of 'noise'.

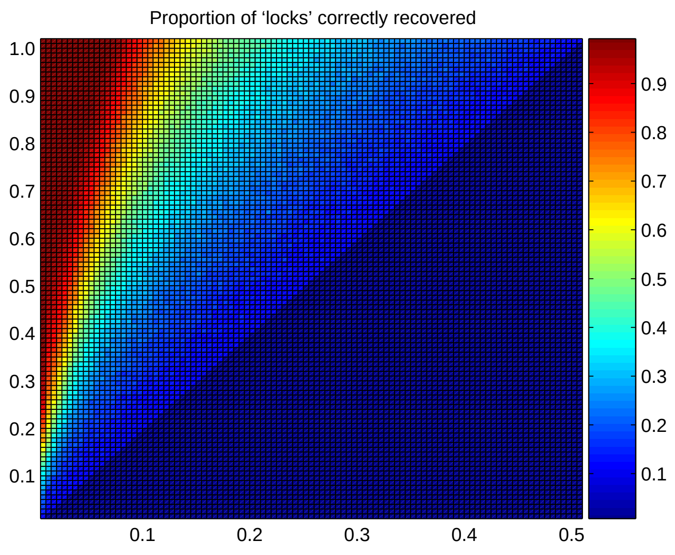


Figure 6.17: Proportion of 'lock' nodes correctly identified as a heat map.

generally expresses significantly below its usual level. This produced a directed network with 133 nodes and 558 edges, whose adjacency matrix is shown in Figure 6.18. Discovering directed bipartite subgraphs in this setting is of major biological interest, as it reveals a pair of gene groups such that over-expression in one group is associated with under-expression in the other.

The singular values for this adjacency matrix are plotted in Figure 6.19. We see that the largest singular values are around 8. The first left and right singular vectors were found to produce an approximately bipartite subnetwork where edges crossed between groups in both directions; a feature that suggests there is a significant element of symmetry in the network. Since we are interested here in directional information, we focus on the second left and right singular vectors, which are shown in Figure 6.20.

Keeping in mind the typical network size suggested by the singular values

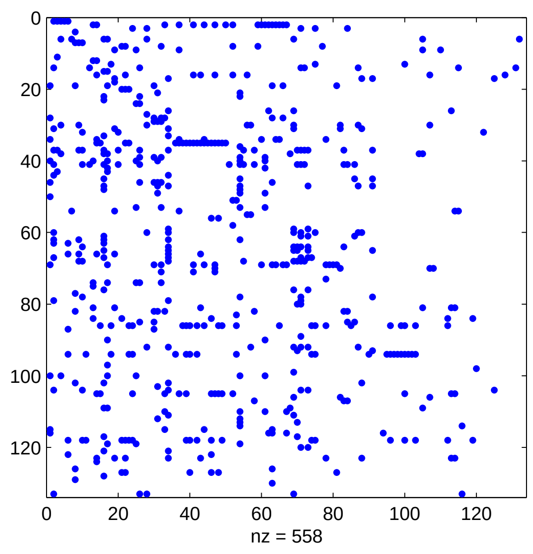


Figure 6.18: Adjacency matrix for a 'plus-minus' network of genes relating to the oncogene p53.

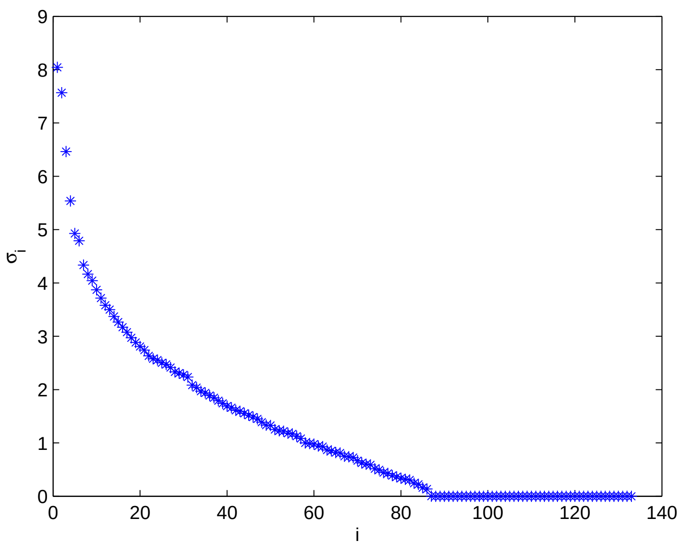


Figure 6.19: Singular values of the adjacency matrix in Figure 6.18.

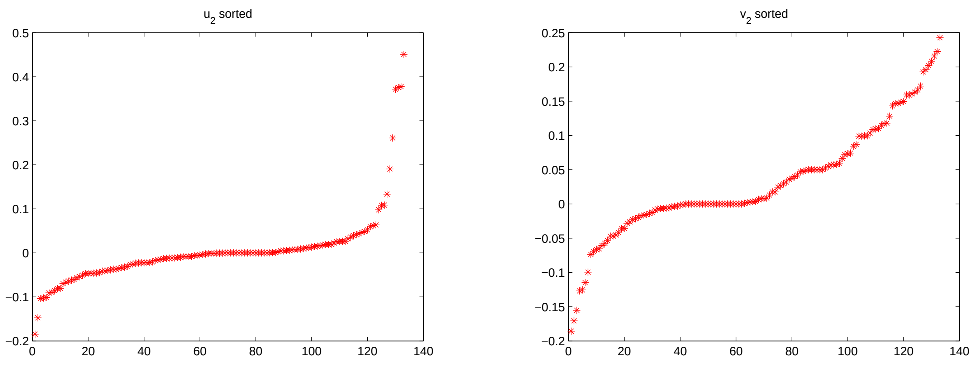


Figure 6.20: Second left and right singular vectors of the adjacency matrix in Figure 6.18.

and considering the natural break-points in the singular vector components, we chose the indices from four extreme components of $\mathbf{u}^{[2]}$ and seven extreme components of $\mathbf{v}^{[2]}$. This produced the subnetwork shown in Figure 6.21. We see that there is a high degree of directed bipartivity. Denoting the first four nodes in this subnetwork as group S_1 and the remaining seven nodes as group S_2, twenty-five out of the possible twenty-eight $S_1 \rightarrow S_2$ connections are present, but none of the other $S_1 \rightarrow S_1$, $S_2 \rightarrow S_2$ or $S_2 \rightarrow S_1$ connections.

The subnetwork in Figure 6.21 was produced using only the network data. Because of the high level of interest in p53, there is extra biological information available, which can be used to justify the relevance of the subnetwork. Details of the genes are given in Table 6.1. Looking at these genes, BTG2, CCNG2 and FHL1 all appear to have inhibitory effects on growth or cell-division, while the role of HLA-F with respect to growth is unclear. KIF15, CDC20, PRC1, CCNB2, KIF20A and NEK2 all seem to be involved in the cell-division process and, in most cases, inhibiting the genes seems to prevent growth. So it appears that we have identified two groups of genes whose products have opposite functions. Whilst we make no detailed biological interpretation here it

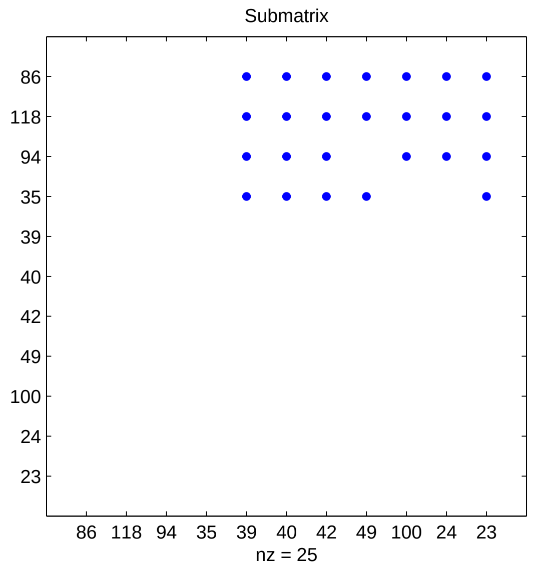


Figure 6.21: Adjacency matrix for the subgraph of 11 nodes discovered via the second left and right singular vectors.

seems reasonable that their mutually exclusive expression patterns are consistent with growth promotion and inhibition being correlated, with one of these groups being switched on while the other is off.

6.6 Application to Other Networks

In the upper left of Figure 6.22 we show the adjacency matrix for a directed network constructed from the `RDHLP` interactions in the `WIRING` data set of `UCINET IV` at

`http://vlado.fmf.uni-lj.si/pub/networks/data/Ucinet/UciData.htm`

This data comes from a study in [77] of workers in a bank wiring room. In the network displayed there are nine nodes, corresponding to wiremen or

Probe Set ID	Gene ID	Description
201235_s_at	BTG2	BTG family, member 2
202769_at	CCNG2	Cyclin G2
201540_at	FHL1	four and a half LIM domains 1
221978_at	HLA-F	major histocompatibility complex, class I, F
219306_at	KIF15	kinesin family member 15
202870_s_at	CDC20	CDC20 cell division cycle 20 homolog (S. cerevisiae)
218009_s_at	PRC1	protein regulator of cytokinesis 1
204822_at	TTK	TTK protein kinase
202705_at	CCNB2	cyclin B2
218755_at	KIF20A	kinesin family member 20A
204641_at	NEK2	NIMA (never in mitosis gene a)-related kinase 2

Table 6.1: Details of genes corresponding to the groups in Figure 6.20. The two groups are separated by a double horizontal line.

assemblers. A link from node i to node j indicates that person i was observed to help person j in the course of their work. The dominant singular value of the adjacency matrix is $\sigma_1 = 2.8$; the next is $\sigma_2 = 1.5$. The first left and right singular vectors, $\mathbf{u}^{[1]}$ and $\mathbf{v}^{[1]}$, are shown as the upper and lower right pictures in Figure 6.22. Taking the two extreme components of $\mathbf{u}^{[1]}$ and the three extreme components of $\mathbf{v}^{[1]}$, we arrive at the subnetwork shown in the lower left of the figure. We see that there is an almost complete directed bipartite structure; with $S_1 = \{6, 1\}$ and $S_2 = \{3, 9, 7\}$ we have one missing link and no spurious links. In this context S_1 represents a group of workers who gave unreciprocated help to the workers in S_2.

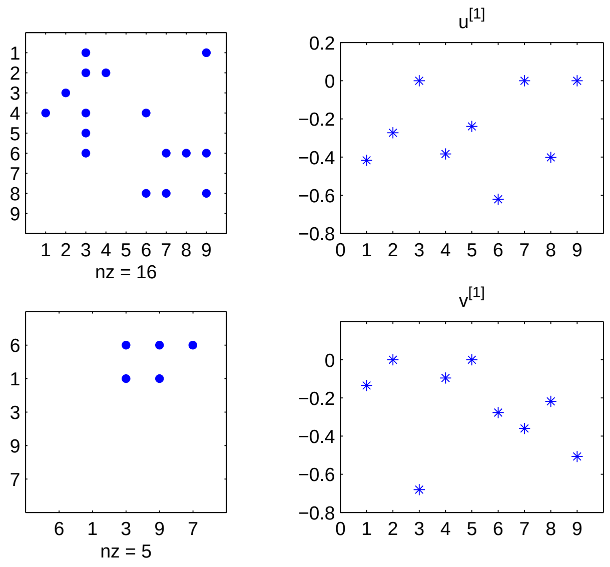


Figure 6.22: Upper left: adjacency matrix for a small social interaction network. Upper and lower right: first left and right singular vectors. Lower left: subnetwork using extremal nodes {6,1} from $\mathbf{u}^{[1]}$ and {3,9,7} from $\mathbf{v}^{[1]}$.

6.7 Conclusions

Overall, we believe that these results show a proof of principle for the SVD as a tool for discovering directed bipartite communities. In the context of analysing high-throughput expression data its main utility, of course, will lie in the case where directed bipartite patterns are found that involve gene groups for which annotational information is missing or only partially known. In this way, the algorithm could suggest putative functional and cause-and-effect relationships that may direct more specific experiments. More generally, for any set of unsymmetric interaction data this new method for detecting directed bipartite community structure offers a useful tool for highlighting meaningful information.

Chapter 7

Directed Stickiness Model

7.1 Motivation

Although random graph models such as the Erdös-Rényi or Gilbert models can be easily extended to produce directed networks, it has been argued that they are unsuitable for use as test matrices or for comparison with real world networks. Looking ahead to Chapter 8, we wish a means of generating realistic random graphs such that their expected degrees match those of the graph we are investigating. This will allow us to conduct more meaningful tests of the statistical significance of our results than would be possible relying on Erdös-Rényi random graphs. The work by Pržulj and Higham [76] established a procedure for generating such graphs in the undirected case and showed that properties of real protein-protein interaction networks can be recovered successfully. We now seek to extend this idea to the directed setting. Our aim is to match two quantities for each node in the graph, the in-degree and out-degree.

7.2 Model

For each node i we define two quantities, $\theta_{out}(i)$ and $\theta_{in}(i)$, such that these are a measure of the likelihood of that node having a connection to/from another node in a particular direction. We define the probability of a connection from node i to node j as the product

$$\theta_{out}(i)\theta_{in}(j).$$

Our aim is that the expected out-degree of node i in the model matches the out-degree of node i in the given data. This requires

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} &= \mathbb{E}(\text{out-degree}(i)) \\ &= \sum_{j=1}^{n} \theta_{out}(i)\theta_{in}(j) \\ &= \theta_{out}(i) \sum_{j=1}^{n} \theta_{in}(j). \end{aligned} \tag{7.1}$$

But since $\sum_{j=1}^{n} \theta_{in}(j)$ does not depend on i, this shows that

$$\theta_{out}(i) \propto \sum_{j=1}^{n} a_{ij}.$$

So let

$$\theta_{out}(i) = \frac{1}{K_1} \sum_{j=1}^{n} a_{ij}. \tag{7.2}$$

Similarly we wish the expected in-degree of node i in the model to match the in-degree of node i in the data, giving

$$\begin{aligned}\sum_{j=1}^{n} a_{ji} &= \mathbb{E}(\text{in-degree}(i)) \\ &= \sum_{j=1}^{n} \theta_{out}(j)\theta_{in}(i) \\ &= \theta_{in}(i)\sum_{j=1}^{n}\theta_{out}(j).\end{aligned}$$

Since $\sum_{j=1}^{n}\theta_{out}(j)$ does not depend on i, we have

$$\theta_{in}(i) \propto \sum_{j=1}^{n} a_{ji},$$

so we let

$$\theta_{in}(i) = \frac{1}{K_2}\sum_{j=1}^{n} a_{ji}. \tag{7.3}$$

Having determined their general form, we now wish to find appropriate constants of proportionality, K_1 and K_2, such that the expected in and out-degrees in the model match those of the initial data. Returning to the out-degree of node i, using Equations 7.1, 7.2 and 7.3, we require

$$\sum_{j=1}^{n} a_{ij} = \left(\frac{1}{K_1}\sum_{j=1}^{n} a_{ij}\right)\left(\frac{1}{K_2}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{kj}\right),$$

which leads to

$$\frac{1}{K_1K_2}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{kj} = 1. \tag{7.4}$$

Likewise, considering the in-degree of node i we require

$$\sum_{j=1}^{n} a_{ji} = \left(\frac{1}{K_2}\sum_{j=1}^{n} a_{ji}\right)\left(\frac{1}{K_1}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{jk}\right),$$

which becomes

$$\frac{1}{K_1 K_2}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{jk} = 1. \tag{7.5}$$

From Equations 7.2, 7.3, 7.4 and 7.5 we have

$$\sum_{j=1}^{n} \theta_{in}(j) = K_1$$

and

$$\sum_{j=1}^{n} \theta_{out}(j) = K_2.$$

Since the sum of in-degrees and out-degrees in a network must be equal, it is reasonable to assume that the sum of expected in-degrees and expected out-degrees should be the same, and we arrive at

$$K_1 = K_2 = \sqrt{\sum_{j=1}^{n}\sum_{k=1}^{n} a_{jk}}. \tag{7.6}$$

We can now write down an algorithm to produce an instance of such a random graph.

- Input deg_{in} and deg_{out}, vectors of in/out-degrees.
- Compute the scaling factor $w = \sqrt{\sum_i deg_{in}(i)}$.
- Let $\theta_{in} = w^{-1} deg_{in}$ and $\theta_{out} = w^{-1} deg_{out}$.
- For each pair of nodes i and j, connect i to j with independent probability

$\theta_{out}(i) \times \theta_{in}(j)$.

We emphasize that this is a natural generalization of the original stickiness model in [76] to the case of directed edges.

7.2.1 Constraints

It should be noted that for the model to be valid we require all probabilities to be bounded above by one; that is,

$$\theta_{out}(i)\theta_{in}(j) \leq 1 \text{ for } i \neq j.$$

Using the expressions for $\theta_{in}(i)$ and $\theta_{out}(i)$ we can obtain constraints on the structure of the target network that ensure this condition. We have

$$\begin{aligned}
\theta_{out}(i)\theta_{in}(j) &= \frac{1}{K_1 K_2} \sum_{k=1}^{n} a_{ik} \sum_{k=1}^{n} a_{kj} \\
&= \frac{deg_{out}(i)deg_{in}(j)}{K_1 K_2} \\
&= \frac{deg_{out}(i)deg_{in}(j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}} \\
&= \frac{deg_{out}(i)deg_{in}(j)}{\text{no. edges}}.
\end{aligned}$$

This quantity is less than 1 for all i and j if the product of the largest in-degree and the largest out-degree is less than the total number of edges in the target network. This happens, for example, when all degrees are bounded by

$\sqrt{n}$. The datasets considered in this work are generally very sparse and conform to this requirement.

7.3 Variations

As well as the algorithm outlined in Section 7.2, two variations of the stickiness model may be suitable in certain situations. We may produce a 'shuffled' stickiness graph by randomly permuting the components of deg_{in} and deg_{out} so that rather than trying to match the in and out-degree of each node, we only aim to match the distribution of in and out-degrees as a whole. We may also wish to produce a stickiness graph with a particular bias built in whereby we wish to force nodes with high in-degree to also have high out-degree. This can be accomplished by simply sorting the components of deg_{in} and deg_{out}. Both variations may be useful if we wish to destroy inherent bipartite substructure in the data. Figure 7.1 shows the degree distributions of a synthetic network generated by the directed stickiness method with a *c. elegans* neural network as input. This particular network will be discussed in more detail in Section 8.6. Similarly Figures 7.2 and 7.3 show the degree distributions of networks generated using the shuffling and biased methods respectively. In each case, the blue circles represent degree counts for one sample from the random network model. In order to quantify the success of the stickiness models at producing synthetic networks with suitable degree distribution, we use `ranksum`, MATLAB's implementation of the Mann-Whitney U test. The test compares two samples and returns a pvalue which is the probability of observing this result, or one more extreme, given that a null-hypothesis is true. In this instance, the null hypothesis is that the two samples (the degree distributions) came from distributions with equal medians. The results are given below in Table 7.1. The pvalues are sufficiently high that we cannot reject the null-hypothesis at the 5% significance level.

Stickiness model	in-degree pvalue	out-degree pvalue
Directed stickiness	0.8337	0.7537
Shuffled stickiness	0.9414	0.5723
Biased stickiness	0.8000	0.6191

Table 7.1: Significance of subgraph in test case 1 for varying test matrix classes.

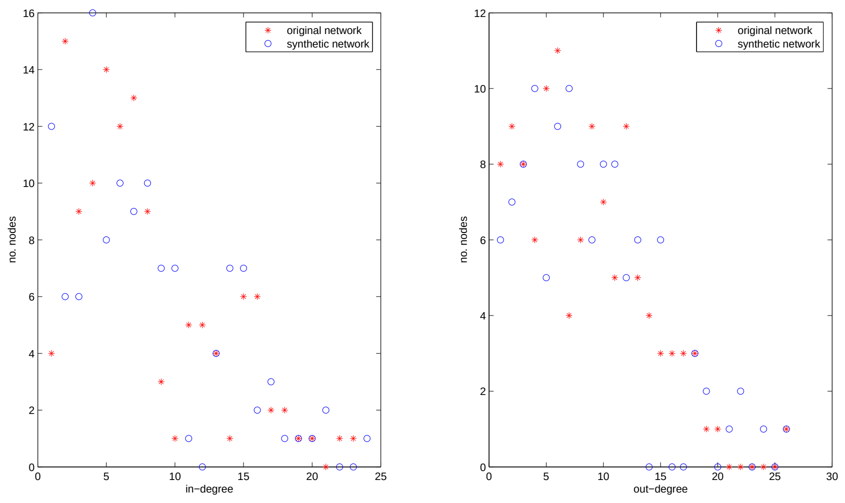


Figure 7.1: Degree distributions of *c. elegans* neural network and one fitted directed stickiness network.

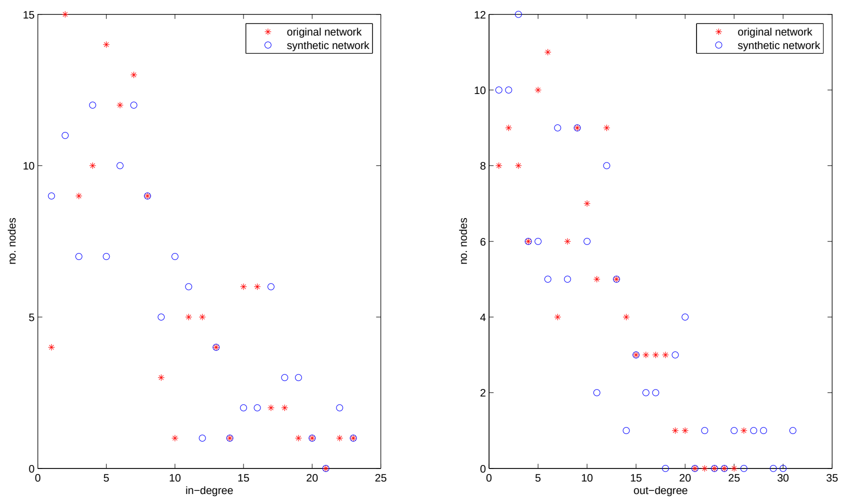


Figure 7.2: Degree distributions of *c. elegans* neural network and one shuffled stickiness network.

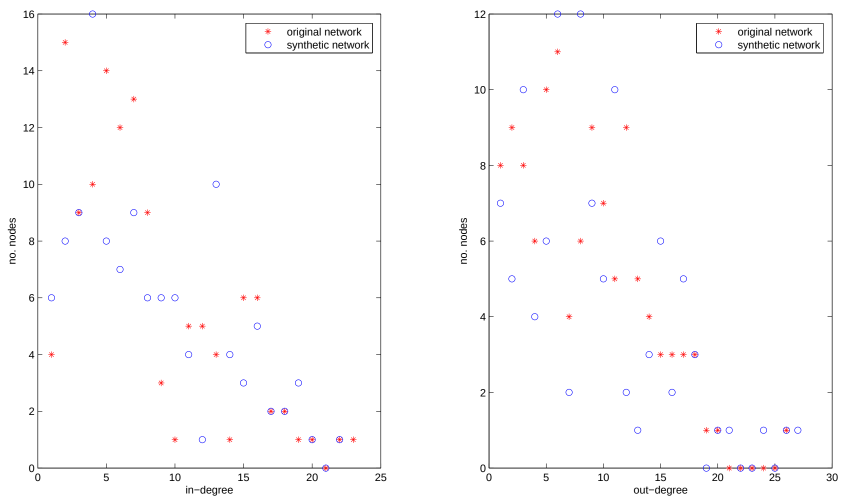


Figure 7.3: Degree distributions of *c. elegans* neural network and one biased stickiness network.

7.4 Monte Carlo Testing

In order to confirm the analytical result in Section 7.2 regarding the expected degrees of nodes generated by the stickiness model, we construct many networks using the degree distributions of the *c. elegans* neural network as input and compute the sample average of the degree of each node. The results are sorted and plotted in Figures 7.4 to 7.6 for 1, 10 and 1000 samples of stickiness networks respectively. Although a single instance of a stickiness network closely approximates the overall degree distribution of the original network, it can be seen that increasing the sample size improves the fit of the expected degree distribution to the target distribution.

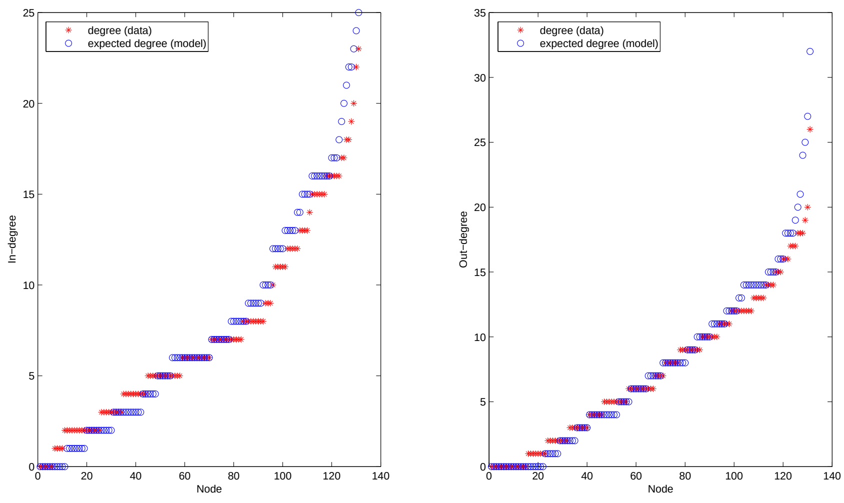


Figure 7.4: In-degrees and out-degrees of nodes for a single stickiness network.

To show the convergence of the expected degree of a node to that in the original distribution, we fix a node and plot the modulus of the difference between expected degree and 'target' degree as the sample size increases. These

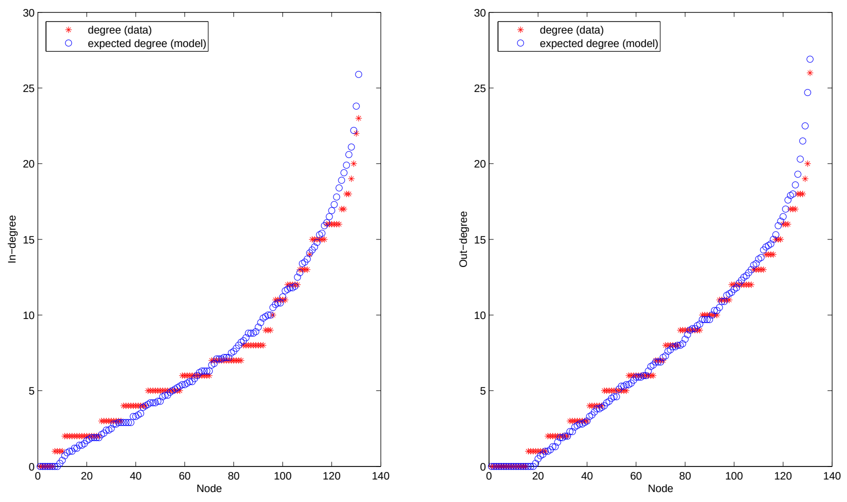


Figure 7.5: Expected in-degrees and out-degrees of nodes over 10 samples of stickiness networks.

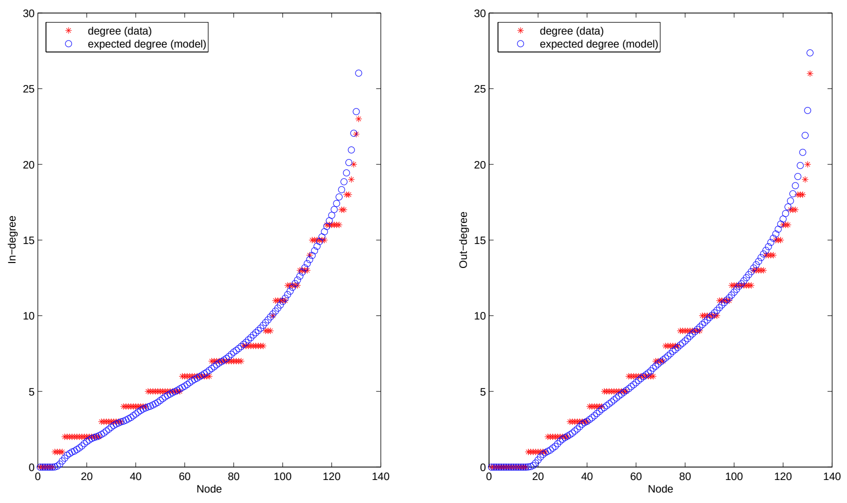


Figure 7.6: Expected in-degrees and out-degrees of nodes over 1000 samples of stickiness networks.

results are shown in Figure 7.7, with sample size increasing to 100,000. The figure clearly shows that the error, that is to say the difference between the target degree and sample average degree in the model, decays like the square root of the sample size. The red line in the figure shows one over the square root of the sample size. This is standard for the sampling error in a Monte Carlo simulation [37]. The node chosen in this instance has in-degree and out-degree close to the mean for the network and so this plot is an indicator of typical behaviour rather than of exceptional circumstances such as high in-degree and low out-degree.

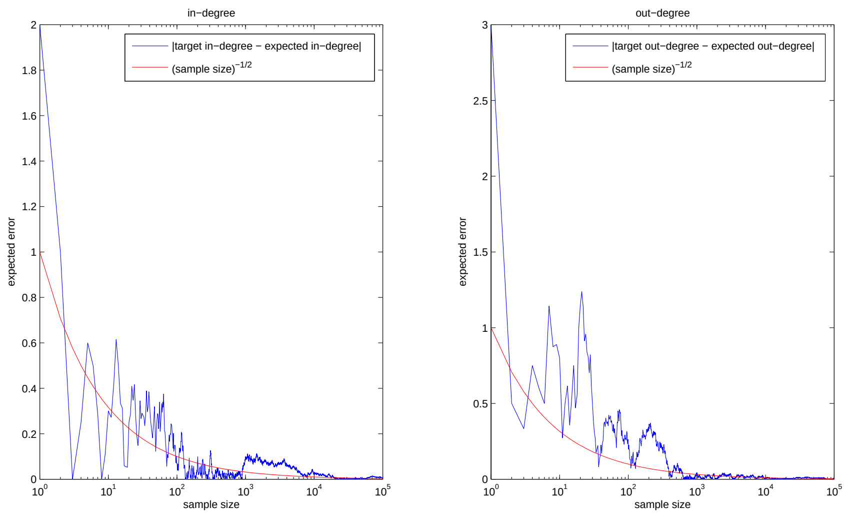


Figure 7.7: Error in expected in-degree and out-degree for a single node with a reference line superimposed.

Chapter 8

Mapping Directed Networks

8.1 Motivation

Although the algorithm presented in Chapter 6 was shown to be useful in identifying bipartite substructures within networks, the reliance on two separate orderings of nodes (one for the 'locks' and one for the 'keys') meant that visualisation could be confusing and results could be potentially misleading. There is a wealth of literature on clustering in symmetric matrices (and therefore on undirected networks) and it is our aim to develop a procedure that will allow us to take advantage of these methods when attempting to identify bipartite communities within directed networks. To this end we propose a matrix mapping that translates an unsymmetric, binary adjacency matrix to a symmetric, real valued matrix which is amenable to standard clustering algorithms.

8.2 Theory

We begin by introducing the new concept of an *alternating walk*. An alternating walk is a traversal of a graph that successively respects then violates edge direction.

Definition 1. An alternating walk of length k from node i_1 to node i_{k+1} is a sequence of (not necessarily distinct) nodes $i_1, i_2, i_3, \ldots, i_{k+1}$ such that $a_{i_s,i_{s+1}} \neq 0$ for s odd and $a_{i_{s+1},i_s} \neq 0$ for s even.

From the definition of a matrix product, it follows that

$$(AA^T AA^T \ldots)_{ij} \tag{8.1}$$

with k factors counts the number of alternating walks of length k from node i to node j.

Suppose now that a network contains subsets S_1 and S_2 that form an approximate directed bipartite community as described in the previous section. Given two nodes i and j drawn from these subsets, it is possible to make statements about the likelihood of connections between these nodes based upon their locations. If both i and j are members of subset S_1, it is unlikely that there will be a link from i to j but it is likely that there will be many ways to traverse from i to j via S_2 by following one edge in the correct direction and then a different edge in the wrong direction. In other words we expect there to be few alternating walks of length one from i to j but many alternating walks of length two. Continuing this argument, for i and j both belonging to subset S_1 we expect many even-length alternating walks but few odd-length alternating walks from i to j.

On the other hand, if node i belongs to S_1 and node j belongs to S_2 then it is likely that there will be a link between them, or that there will be ways to traverse from i to j by following an edge in the correct direction, another in the wrong direction and then a third in the correct direction. So for i and j belonging to S_1 and S_2 respectively, we expect many odd length alternating walks but few even length alternating walks from i to j.

Although taking into account information about longer walks may be an effective way of compensating for noise within a network, short-length walks are in general more informative and so it is reasonable to weight walks by a quantity inversely proportional to their length. In previous work [28, 29], walks of length k are commonly scaled by a factor $1/k!$ and it is this weighting that we shall use. Thus, a measure of similarity between nodes i and j may be defined as the difference between the total number of even and odd length alternating walks (appropriately scaled) and is expressed by the new matrix mapping

$$f(A) = I - A + \frac{AA^T}{2!} - \frac{AA^TA}{3!} + \frac{AA^TAA^T}{4!} - \dots \tag{8.2}$$

The identity matrix I is introduced here for convenience, although it has the intuitive meaning that there is an alternating walk of length zero from a node to itself. Expressing A in terms of its singular value decomposition $A = U\Sigma V^T$, where $U, V \in \mathbb{R}^{N\times N}$ are orthogonal and $\Sigma \in \mathbb{R}^{N\times N}$ is diagonal, we have

$$\begin{aligned} f(A) &= I - U\Sigma V^T + \frac{(U\Sigma V^T)(U\Sigma V^T)^T}{2!} - \frac{(U\Sigma V^T)(U\Sigma V^T)^T(U\Sigma V^T)}{3!} + \dots \\ &= I - U\Sigma V^T + \frac{(U\Sigma^2 U^T)}{2!} - \frac{(U\Sigma^3 V^T)}{3!} + \dots, \end{aligned}$$

which may be written

$$f(A) = U\cosh(\Sigma)U^T - U\sinh(\Sigma)V^T. \tag{8.3}$$

It should be noted that the matrix mapping may be expressed in terms of traditional matrix functions and we now show this.

From Equation 8.2 we have

$$\begin{aligned} f(A) &= I - A + \frac{AA^T}{2!} - \frac{AA^TA}{3!} + \frac{AA^TAA^T}{4!} - \dots \\ &= \left(I + \frac{AA^T}{2!} + \frac{AA^TAA^T}{4!} + \dots\right) - \left(A + \frac{AA^TA}{3!} + \frac{AA^TAA^TA}{5!} + \dots\right). \end{aligned}$$

Now letting $B = AA^T$ and observing that $(B^{\frac{1}{2}})^2 = AA^T$ we have

$$f(A) = \left(I + \frac{(B^{\frac{1}{2}})^2}{2!} + \frac{(B^{\frac{1}{2}})^4}{4!} + \dots\right) - \left(A + \frac{(B^{\frac{1}{2}})^2}{3!} + \frac{(B^{\frac{1}{2}})^4}{5!} + \dots\right)A.$$

Rewriting I as $B^{\frac{1}{2}}B^{-\frac{1}{2}}$ this gives

$$\begin{aligned} f(A) &= \cosh(B^{\frac{1}{2}}) - \left(\left[B^{\frac{1}{2}}B^{-\frac{1}{2}}\right] + \frac{(B^{\frac{1}{2}})^2\left[B^{\frac{1}{2}}B^{-\frac{1}{2}}\right]}{3!} + \dots\right) \\ &= \cosh(B^{\frac{1}{2}}) - \left(B^{\frac{1}{2}} + \frac{(B^{\frac{1}{2}})^3}{3!} + \frac{(B^{\frac{1}{2}})^5}{5!} + \dots\right)B^{-\frac{1}{2}}A \\ &= \cosh(B^{\frac{1}{2}}) - \sinh(B^{\frac{1}{2}})B^{-\frac{1}{2}}A. \end{aligned}$$

Based on our motivating remarks, we would expect $f(A)_{ij}$ to take large positive values when $i, j \in S_1$. We illustrate this idea with an example.

A network of fifty nodes is generated by considering pairs of nodes, i and j, and connecting them with independent probability p_{ij}. Nodes are artificially organised into subsets in such a way that nodes $\{1, 2, \ldots, 10\}$ point to nodes $\{11, 12, \ldots, 25\}$ with independent probability 0.65, nodes $\{30, 31, \ldots, 39\}$ point to nodes $\{41, 42, \ldots, 50\}$ with independent probability 0.8 and all other pairs of nodes are connected with independent probability 0.05. The spy-plot of such a network is shown in Figure 8.1, where the directed interaction between communities can be clearly seen. It is important to note that in practice community structure may not be visible until some reordering of the nodes has been applied. A heat map of the mapped matrix $f(A)$ is shown in Figure 8.2 where 'hot' regions are seen to correspond to $S_1 \rightarrow S_1$ relationships and 'cold' regions to $S_1 \rightarrow S_2$ relationships.

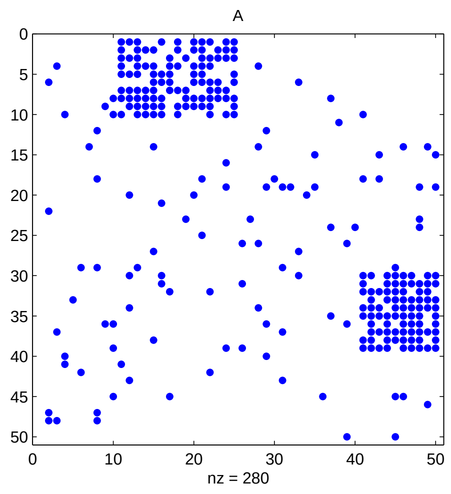


Figure 8.1: Spy-plot of an artificial network containing two directed bipartite subgraphs.

Analagously, we can argue that $f(A^T)$ will have large positive entries for $S_2 \rightarrow S_2$ relationships and large negative entries for $S_2 \rightarrow S_1$ relationships, as shown in Figure 8.3. Therefore the sum $f(A) + f(A^T)$ should be useful in revealing inter-cluster relationships ($S_1 \rightarrow S_1$ and $S_2 \rightarrow S_2$) through positive entries and extra-cluster relationships ($S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$) through negative

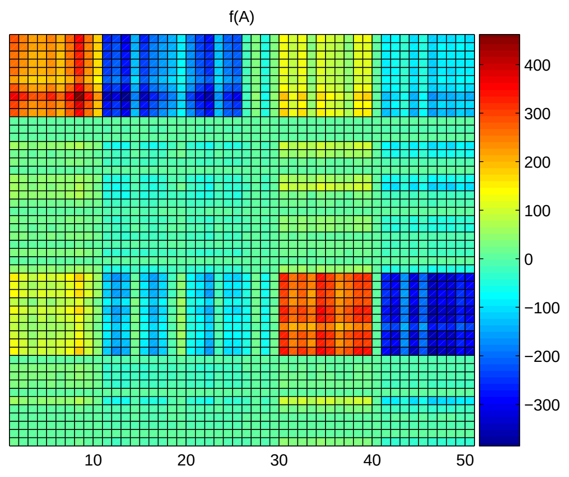


Figure 8.2: Heat map of the mapped matrix $f(A)$ in Equation 8.2 for the network in Figure 8.1.

entries. The heat map in Figure 8.4 illustrates this effect.

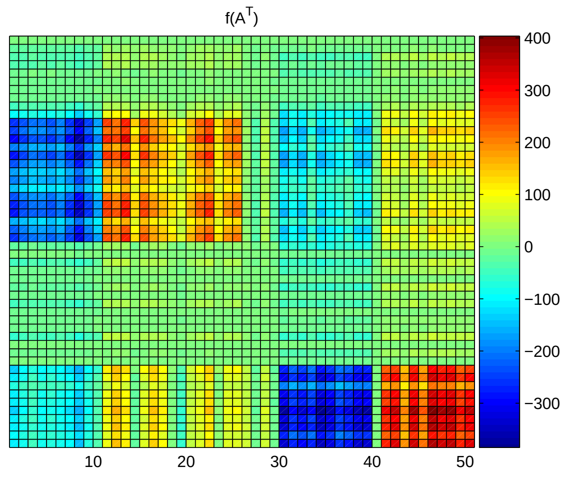


Figure 8.3: Heat map of the mapped adjacency matrix transpose $f(A^T)$

We show now that summing $f(A) + f(A^T)$ results in a symmetric matrix. We have

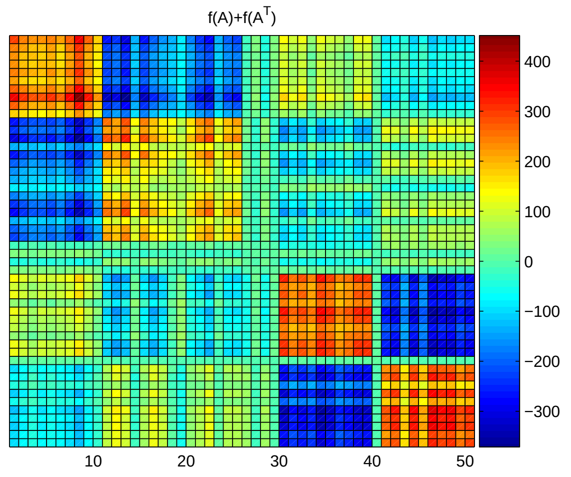


Figure 8.4: Heat map of the symmetric mapped matrix $f(A) + f(A^T)$.

$$\begin{aligned} f(A) + f(A^T) &= U\cosh(\Sigma)U^T - U\sinh(\Sigma)V^T \\ &\quad + V\cosh(\Sigma)V^T - V\sinh(\Sigma)U^T \\ (f(A) + f(A^T))^T &= U\cosh(\Sigma)U^T - V\sinh(\Sigma)U^T \\ &\quad + V\cosh(\Sigma)V^T - U\sinh(\Sigma)V^T \end{aligned}$$

Subtracting gives

$$(f(A) + f(A^T)) - (f(A) + f(A^T))^T = 0$$

and so

$$(f(A) + f(A^T)) \equiv (f(A) + f(A^T))^T.$$

As $f(A) + f(A^T)$ is a symmetric, real valued matrix, standard clustering approaches may be employed to identify the common parts of bipartite communities. We propose using the first eigenvector of $f(A) + f(A^T)$ to obtain a useful reordering of the nodes [50].

8.3 Comparison with Matrix Exponential

The negative matrix exponential has been proposed as a method for identifying bipartite communities within undirected networks [28, 29]. We consider a synthetic example of a directed network comprised of three subsets forming two directed bipartite structures in order to show that the matrix mapping described above gives a more suggestive rendering of the structure within the network than the negative matrix exponential or related functions.

Our synthetic network is comprised of three subsets, S_1, S_2 and S_3, and is generated in such a way that nodes in S_1 tend to point to nodes in S_2 while nodes in S_2 tend to point to nodes in S_3. A spy-plot of the adjacency matrix of this network is shown in Figure 8.5. We then compute the matrix exponential, negative matrix exponential and our matrix mapping for this network and study the heat map and sign pattern of each. By considering how these represent counts of walks around a network, we can argue that the exponentials of A and $-A$ will have 3-by-3 block structure of the form

$$e^A \approx \begin{bmatrix} 0 & + & + \\ 0 & 0 & + \\ 0 & 0 & 0 \end{bmatrix} \text{ and } e^{-A} \approx \begin{bmatrix} 0 & - & + \\ 0 & 0 & - \\ 0 & 0 & 0 \end{bmatrix}$$

whereas $f(A) + f(A^T)$ will take the form

$$f(A) + f(A^T) \approx \begin{bmatrix} + & - & 0 \\ - & + & - \\ 0 & - & + \end{bmatrix}$$

In Figure 8.6 we see that there is no significant correlation between hot or cold regions of the negative matrix exponential and communities in the original adjacency matrix, although inspection of the sign pattern in Figure 8.7 suggests a correspondence between negative entries in e^{-A} and communities in A.

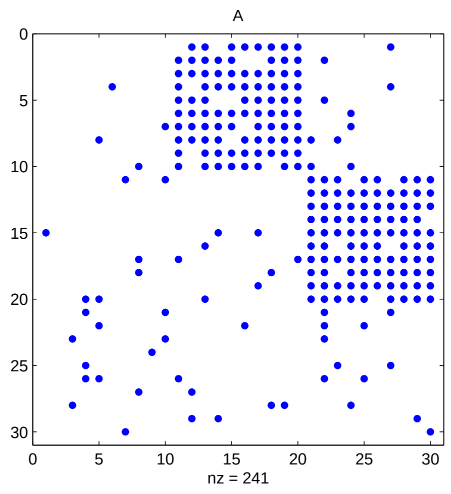


Figure 8.5: Spy-plot of A.

The matrix exponential is less revealing, as can be observed in Figure 8.8. The hottest region of the matrix corresponds to the region of interaction between the two communities with incoming links but fails to highlight membership of any one community. As expected, the sign pattern of e^{A} as seen in Figure 8.9, reveals nothing significant about the network, since there are no negative entries and very few take the value 0.

Finally we compute the matrix mapping $f(A) + f(A^T)$ and consider the heat map and sign pattern. Both clearly identify the three separate communities in the network. The hotter regions in Figure 8.10 indicate community co-membership while the cold regions imply separation between nodes. Observing the sign pattern in Figure 8.11 we note that positive regions almost

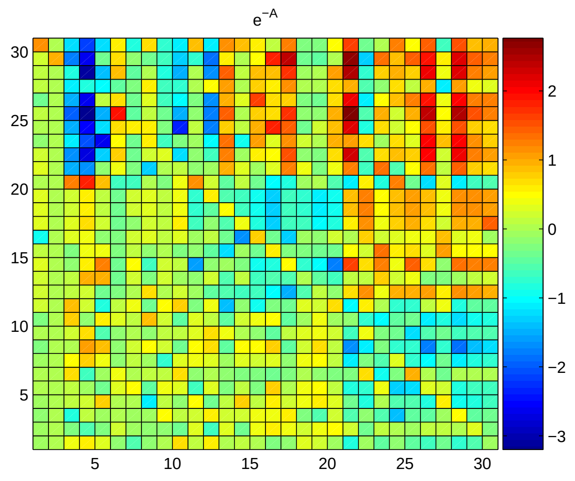


Figure 8.6: Heat map of e^{-A} for A in Figure 8.5.

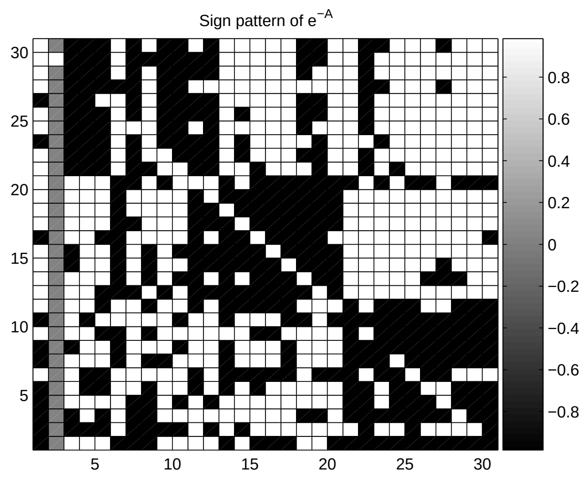


Figure 8.7: Sign pattern of e^{-A} for A in Figure 8.5.

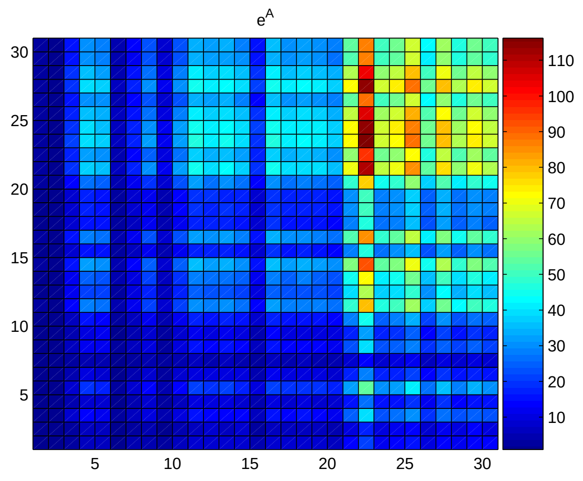


Figure 8.8: Heat map of e^A for A in Figure 8.5.

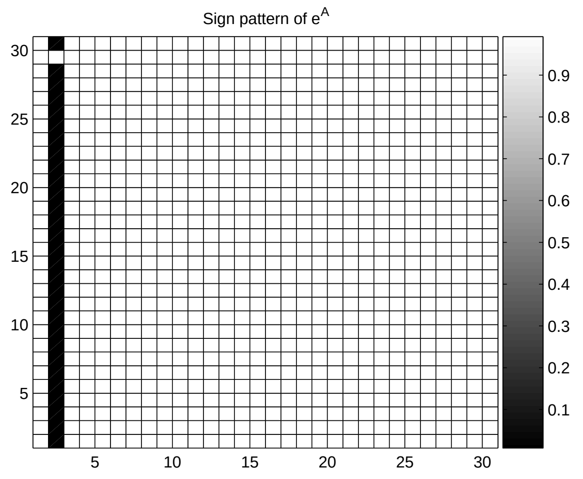


Figure 8.9: Sign pattern of e^A for A in Figure 8.5.

exclusively indicate community co-membership with a few exceptions caused by the built in 'noise' in the matrix.

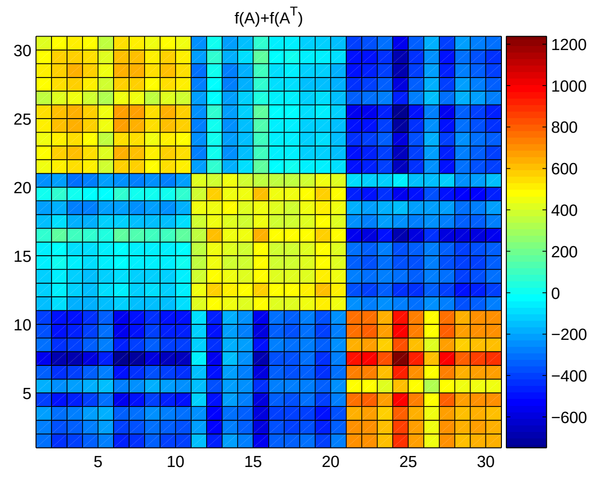


Figure 8.10: Heat map of $f(A) + f(A^T)$ for A in Figure 8.5.

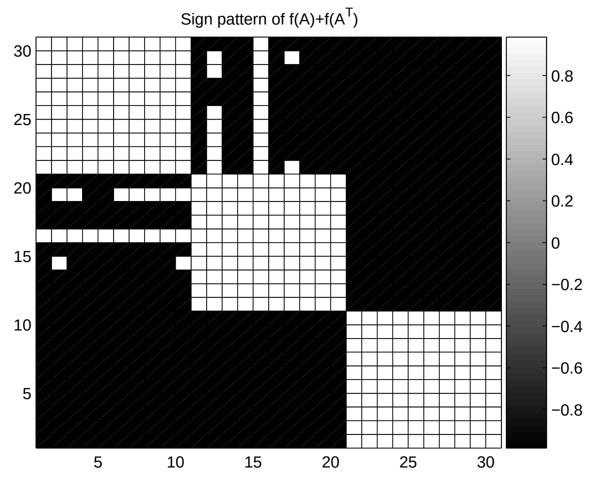


Figure 8.11: Sign pattern of $f(A) + f(A^T)$ for A in Figure 8.5.

8.4 Testing

In practice, nodes comprising directed bipartite communities will rarely be arranged contiguously in the adjacency matrix, so we will rely on a traditional clustering approach to identify such structures. Having calculated the mapped matrix $f(A) + f(A^T)$, we will reorder it according to an appropriate eigenvector. Since the algorithm gives a positive value to pairs of nodes in the same bipartite subgroup, and a negative value to pairs of nodes in different bipartite subgroups, reordering by an appropriate eigenvector should force subgroups to opposite ends of the mapped matrix. We can then extract the adjacency matrix of the subgraph consisting of nodes that are members of bipartite subgroups. By using the same node ordering obtained from the eigenvector of the mapped matrix, the spy-plot of this subgraph should clearly show a directed bipartite connectivity structure.

We consider four synthetic networks with particular connectivity structures. For each example, we distribute nodes to subsets in a random fashion by applying a permutation of the node indices $1 \ldots n$ and assigning contiguous groups of nodes to subsets. For example, if we wish to assign eight nodes to two subsets of equal size, we produce a permutation of the numbers 1 to 8 and assign the first four nodes to S_1 and the remaining four to S_2:

$$1, 2, 3, 4, 5, 6, 7, 8 \rightarrow \underbrace{8, 2, 7, 4}_{S_1}, \underbrace{3, 6, 5, 1}_{S_2}.$$

From this node ordering we construct the adjacency matrix A. We then compute the mapped matrix $f(A) + f(A^T)$ and reorder based on its first eigenvector. We apply this same ordering to A and spy-plot the adjacency matrix to determine whether bipartite communities have been clustered.

8.4.1 Example 1

The first example we consider is a perfect directed bipartite structure, that is to say every node in S_1 is connected to every node in S_2 as shown in Figure 8.12. There are no reciprocal links and there are no inter-set links. Based on the derivation, the matrix mapping should recover community structure perfectly and the first eigenvector of the mapped matrix should yield a permutation which clusters nodes 1-6 and nodes 7-12.

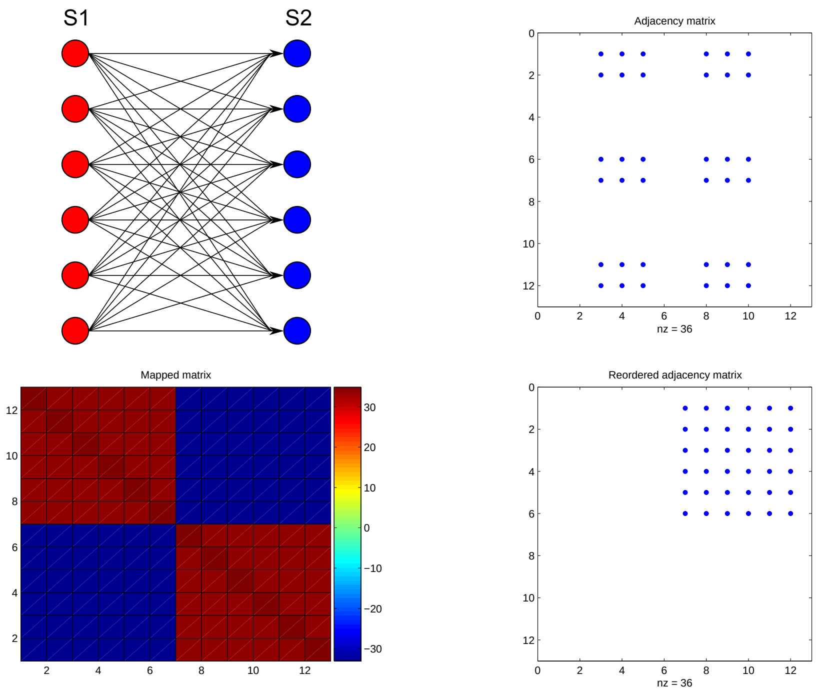


Figure 8.12: Example 1: Directed bipartite network, adjacency matrix, mapped matrix and reordered adjacency matrix.

We label the nodes via the arbitrary permutation

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \rightarrow \underbrace{6, 3, 8, 12, 10, 2}_{S_1}, \underbrace{4, 7, 11, 9, 5, 1}_{S_2}.$$

In Fig. 8.12, a spy-plot of the adjacency matrix for this network is shown together with a pcolor plot of the mapped matrix $f(A) + f(A^T)$ reordered by the first eigenvector. The 'hot' regions have been clustered in blocks along the diagonal, and applying this same reordering to the adjacency matrix, we see that nodes have indeed been clustered into the sets S_1 and S_2. When the node indices are mapped back, the recovered ordering is 3, 2, 4, 5, 1, 6, 8, 12, 10, 7, 11, 9.

8.4.2 Example 2

The second example we consider is a network of three communities, S_1, S_2 and S_3 with directed bipartite connectivity from set S_1 to S_2 and from set S_2 to S_3. This network is shown in Figure 8.13 and, as with the previous example, since there is no inter-set connectivity and there are no reciprocal or missing links, we expect the matrix mapping to distinguish perfectly between the three communities.

We label the nodes via the arbitrary permutation

$$1,2,3,4,5,6,7,8,9,10,11,12 \rightarrow \underbrace{7,2,3,5}_{S_1},\underbrace{1,9,6,12}_{S_2},\underbrace{4,8,11,10}_{S_3}$$

and plot the resulting adjacency matrix in Figure 8.13. The mapped matrix is calculated and reordered according to its first eigenvector. In the pcolor plot of the resulting matrix, we see that the nodes have been separated into three distinct groups and the spy-plot of the reordered adjacency matrix confirms that these groups do comprise nodes of similar connectivity. In fact when the node indices are mapped back, we find that the reordering is 5, 6, 8, 7, 2, 1, 4, 3, 9, 12, 11, 10, so we see that the nodes have been ordered into the three original groupings.

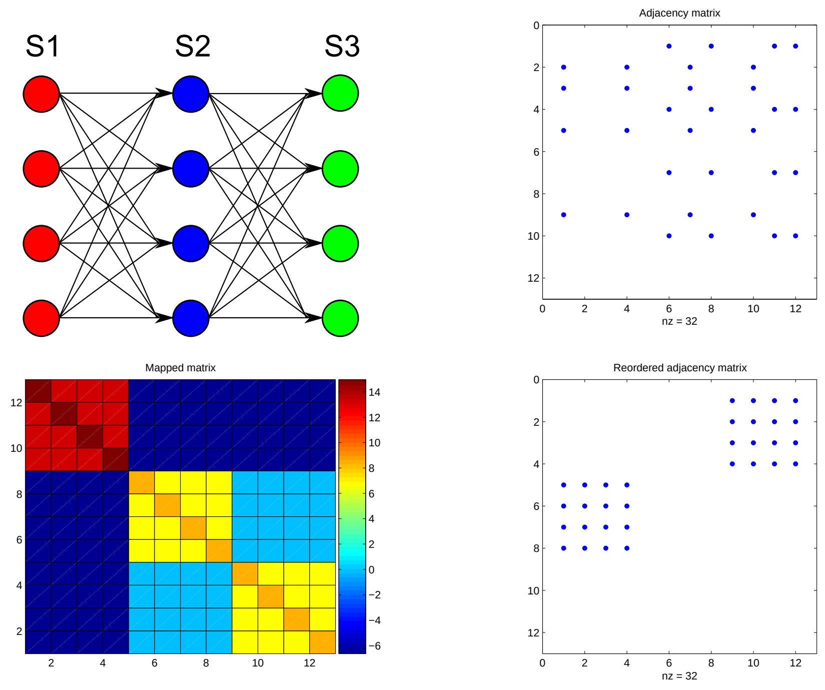


Figure 8.13: Example 2: Two overlapping directed bipartite communities, adjacency matrix, mapped matrix and reordered adjacency matrix.

8.4.3 Example 3

The third synthetic network we consider, shown in Figure 8.14, is comprised of three communities S_1, S_2 and S_3 where there is directed bipartite connectivity from S_1 to S_2 and undirected bipartite connectivity between S_1 and S_3. Although the matrix mapping is designed to reveal directed biparite community structure, it would be beneficial if we can understand whether or not some sort of separation between undirected bipartite subgroups can also be achieved. The arguments used to derive the algorithm can also be used to justify the matrix mapping in this case.

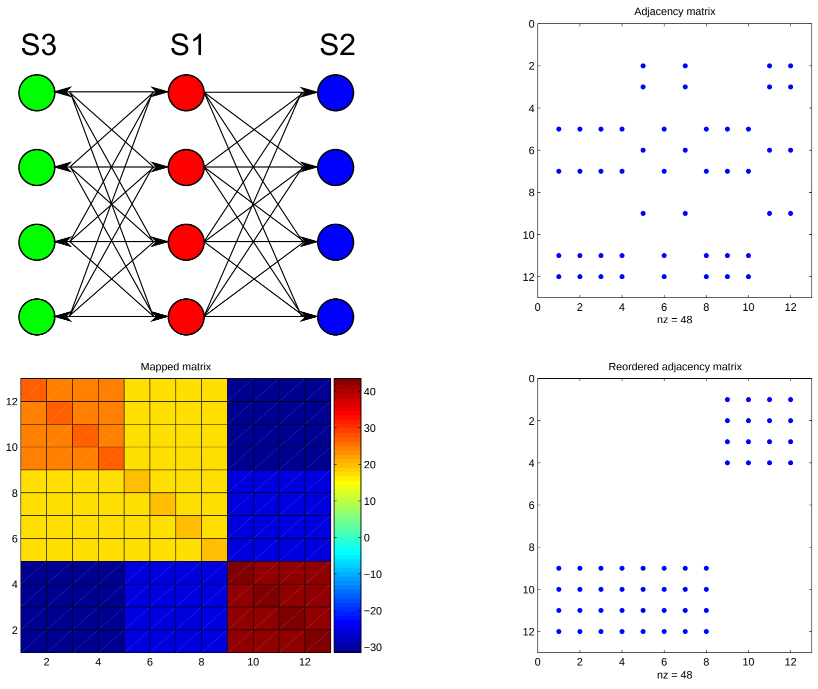


Figure 8.14: Example 3: Communities with directed and undirected bipartite connectivity, adjacency matrix, mapped matrix and reordered adjacency matrix.

We label the nodes via the arbitrary permutation

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \rightarrow \underbrace{8, 12, 9, 7}_{S_1}, \underbrace{1, 10, 4, 6}_{S_2}, \underbrace{11, 5, 3, 2}_{S_3}$$

and plot the resulting adjacency matrix in Figure 8.14. The mapped matrix is calculated and reordered according to the first eigenvector. Once again, this ordering is applied to the adjacency matrix and the resulting spy-plot shows that the nodes have been succesfully separated into three distinct groups. To confirm that these are the groups we wish to identify the node indices are mapped back, revealing the ordering 12, 9, 10, 11, 7, 6, 5, 8, 1, 4, 3, 2. So despite the undirected edges in the network, the matrix mapping has succesfully achieved the separation of nodes that we desired.

8.4.4 Example 4

The final network we consider, as shown in Figure 8.15, consists of three communities which form a cycle under our definition of an alternate walk, that is to say, S_1 points to S_2, and S_3 points to both S_2 and S_1. This means that there are alternating walks of odd length from S_1 to itself via S_2 and S_3, as well as the trivial alternate walks of length 2. Under these conditions, we would expect the matrix mapping to have difficulty achieving separation between the three communities.

We label the nodes via the arbitrary permutation

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \rightarrow \underbrace{3, 6, 11, 5}_{S_1}, \underbrace{2, 8, 7, 4}_{S_2}, \underbrace{9, 1, 12, 10}_{S_3}$$

and plot the spy-plot of the adjacency matrix in Figure 8.15. In this instance, reordering the mapped matrix and adjacency matrix by the first eigenvector of $f(A)+f(A^T)$ reveals a clear separation between groups S_2 and S_3, but the entries pertaining to members of S_1 are close to zero, meaning that they are not clearly identified as taking part in any directed bipartite connections. The recovered

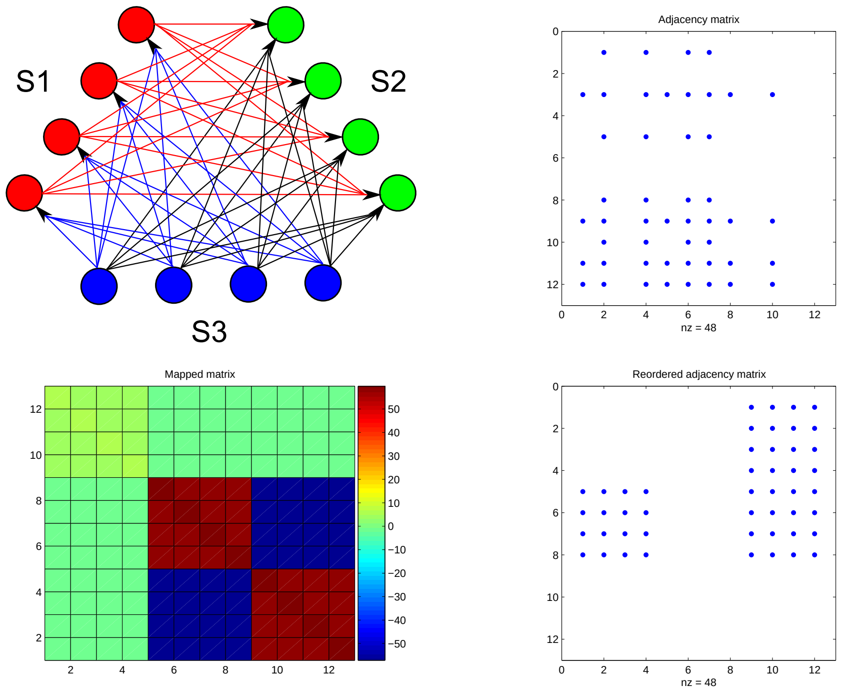


Figure 8.15: Example 4: Three communities with an alternate-walk cycle, adjacency matrix, mapped matrix and reordered adjacency matrix.

node ordering (2, 4, 1, 3, 9, 12, 10, 11, 6, 5, 8, 7), however has separated the nodes into the three original communities, so while the matrix mapping does not identify all the bipartite interactions that are taking place, the subsequent reordering of the adjacency matrix is more illuminating. We emphasise that this network does not fit into the category of approximate directed bipartite communities as defined in Section 6.2 for which our algorithm was designed.

8.5 Statistical Testing

Our aim is, of course, to apply the reordering algorithm to real data sets. In this case there is no guarantee that an approximate directed bipartite community substructure is present in the data. On one hand simply finding a pair of nodes, i and j, such that i points to j but j does not point to i, would not be a convincing result. On the other hand, seeking an equi-partition of all nodes into sets S_1 and S_2, where all possible $S_1 \rightarrow S_2$ links are present but no others, is unreasonable. In practice, we may find a result lying somewhere between these two extremes. In this case it makes sense to quantify the 'significance' of the pattern. We will take the approach of asking how likely we are to observe the same 'level of bipartivity' in an 'arbitrary network of the same type'. We discuss in this section how we chose to formalise this approach.

Consider a perfectly bipartite network consisting of two sets S_1 and S_2 containing m_1 and m_2 nodes respectively. Such a network may be represented by an adjacency matrix with a single off-diagonal nonzero block consisting of the edges from S_1 to S_2. This is illustrated in Figure 8.16. To quantify bipartivity in this subgraph we simply take the ratio of the density of nonzeros in the $S_1 \rightarrow S_2$ block to the density of nonzeros in the remaining L-shaped block plus one (to avoid division by zero) i.e. using nnz to denote "number of nonzeros",

$$\text{bipartivity} = \frac{\dfrac{\text{nnz}(S_1 \to S_2)}{m_1 \times m_2}}{\dfrac{\text{nnz}(S_1 \to S_1 + S_2 \to S_1 + S_2 \to S_2)}{m_1^2 + m_1 m_2 + m_2^2} + 1}.$$

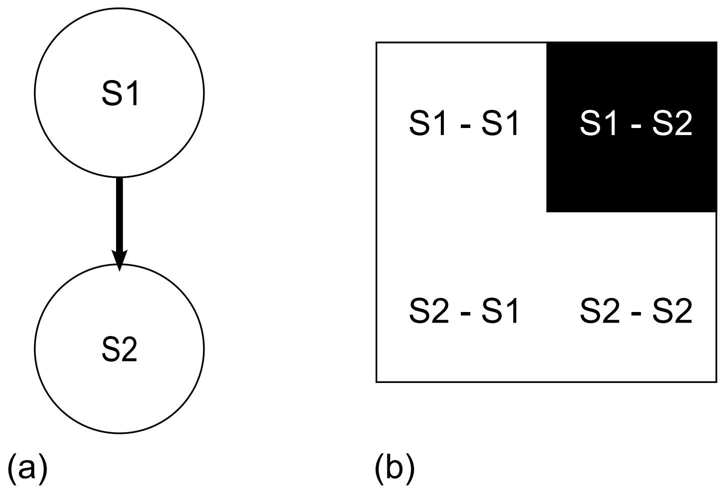


Figure 8.16: Directed bipartite network: (a) edge structure (b) adjacency matrix.

In the case of perfect directed bipartivity, this measure yields a value of 1. The value decreases as nonzeros are added to the L-shaped block or removed from the $S_1 \to S_2$ block. This is analagous to adding edges in the "wrong" direction or edges within subsets. Figure 8.17 shows the value of the bipartivity measure for varying densities of the two relevant regions of the adjacency matrix. As we move to the right along the horizontal axis, the region of bipartite connectivity in the adjacency matrix becomes more dense and the bipartivity value increases. However as we move up the vertical axis, the remaining part of the adjacency matrix becomes more densely packed and the bipartivity value decreases.

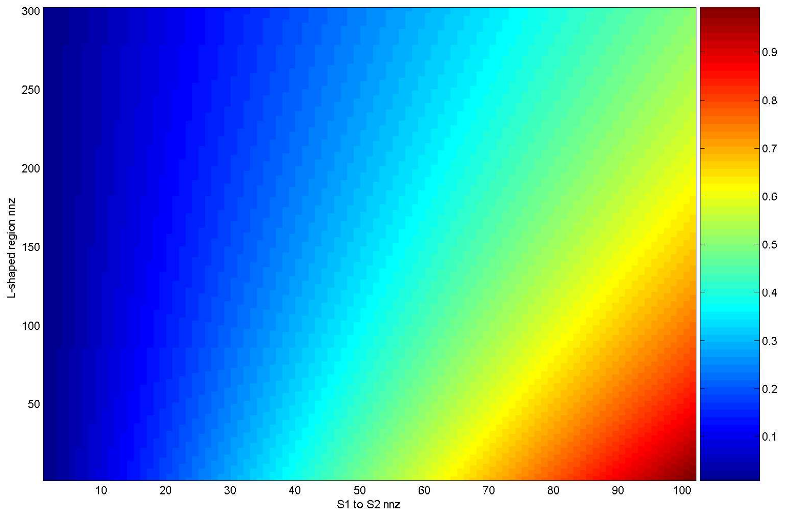


Figure 8.17: Bipartivity measure for varying densities of adjacency matrix regions.

Having obtained a subgraph we wish to quantify the significance of its bipartivity measure. We carry out statistical testing by generating a suitable sample of test matrices using either the Erdös-Rényi model or one of the models discussed in Section 7.3. To each of these we apply the matrix mapping $A \to f(A) + f(A^T)$ and reorder the resulting matrix according to its first eigenvector. A subgraph of the appropriate size is taken and its bipartivity measure is calculated. The proportion of times that the bipartivity value in the randomized data exceeds the bipartivity value for the real data gives us a pvalue [32]. In words, this pvalue answers the question 'How likely are we to observe this level of bipartivity or higher in an arbitrary network of the same type?' In addition to a raw proportion, we also compute a second pvalue by fitting a log-normal distribution to the randomized bipartivity values.

We note that, traditionally, a pvalue below 0.05 is regarded as 'significant' in the sense that values exceeding those on the given network would be observed less than 5% of the time. It is important to note that the pvalue obtained for a given subgraph is dependent upon the class of networks from which the test matrices are drawn. For instance, testing with Erdös-Rényi graphs will often result in a lower pvalue for a particular subgraph (i.e. a higher level of significance) than would directed stickiness graphs. This is because the Erdös-Rényi class tends not to admit large bipartite communities.

Before we apply these ideas to real data, we first test them on synthetic examples where the results can be judged.

8.5.1 Test Case 1

We consider a synthetic network consisting of 100 nodes where a connection between node i and node j occurs with independent probability 0.9 if $i \in [1, 20]$ and $j \in [21, 40]$ and with probability 0.3 elsewhere. This ensures a high level of directed bipartite connectivity from nodes $1 - 20$ to nodes $21 - 40$ with large proportions of false-negatives in this region and of false-positives elsewhere. We then calculate the mapped matrix $f(A) + f(A^T)$ and reorder according to its first eigenvector. This matrix is plotted, together with the adjacency matrix in Figure 8.18. We extract a subgraph consisting of the first and last 20 nodes in the reordering and plot this in Figure 8.19. The bipartivity score for this particular subgraph is calculated to be 0.6138. Upon inspection of the nodes included in the subgraph, we find that 85% of nodes in S_1 were among those we expected to find $(1 - 20)$ and in set S_2, 85% of nodes were those we expected $(21 - 40)$.

We then conduct statistical testing by generating 1000 instances of directed

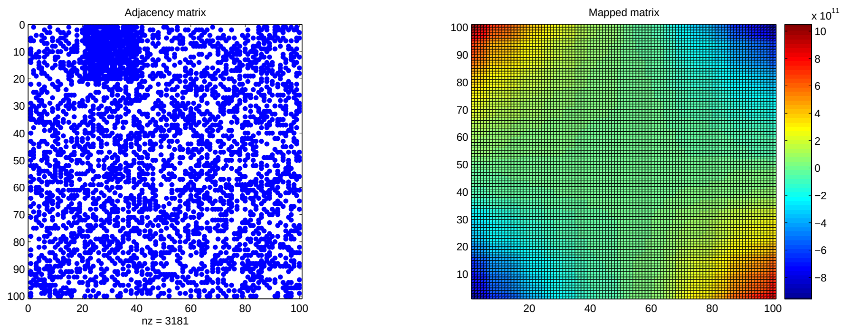


Figure 8.18: Test case 1: Adjacency matrix and mapped matrix.

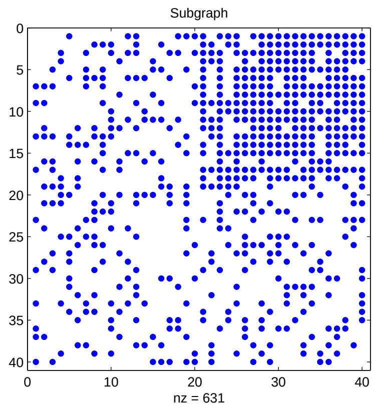


Figure 8.19: Test case 1: Subgraph.

stickiness graphs using our synthetic network as input. Subgraphs of the appropriate size are extracted from each instance according to the procedure described above and a bipartivity score is calculated in each case. The bipartivity values for this sample are plotted in Figure 8.20 with the score for the synthetic network denoted by a green circle. Analysis shows that the distribution of bipartivity scores closely matches a log-normal probability distribution shown in Figure 8.21 together with a quantile-quantile plot [46] of the bipartivity scores together with normalized theoretical quantiles. From this sample we obtain a pvalue of 2.33×10^{-15}. Using the same synthetic network, we carry out testing using all three variants of the directed stickiness model as well as Erdös-Rényi graphs and list the results in Table 8.1. Two measures are listed, pvalue1 which is calculated using a log-normal fit and pvalue2 which is simply the fraction of samples with bipartivity score greater than the original network. For all randomized samples, the low pvalues suggest a significant level of bipartivity in the original network.

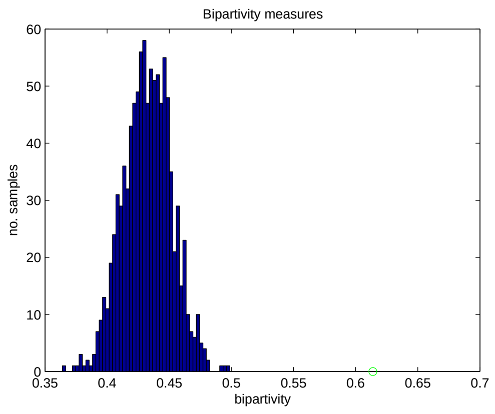


Figure 8.20: Test case 1: Histogram of bipartivity scores.

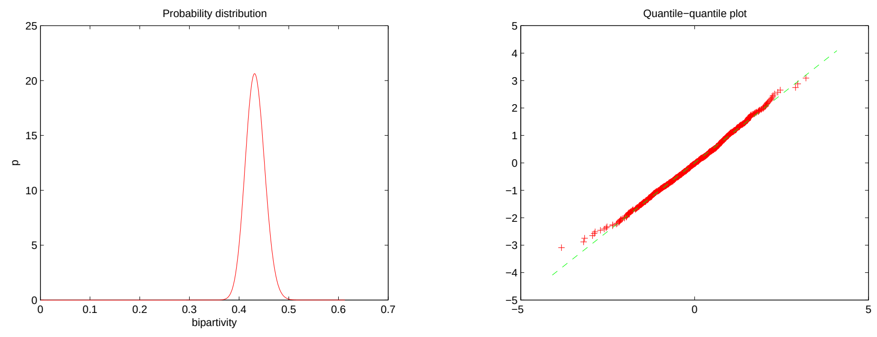


Figure 8.21: Test case 1: Probability distribution and quantile-quantile plot.

	pvalue 1	pvalue 2
Erdös-Rényi	0	0/1000
Directed stickiness	2.33×10^{-15}	0/1000
Shuffled stickiness	5.05×10^{-15}	0/1000
Biased stickiness	0	0/1000

Table 8.1: Significance of subgraph in test case 1 for varying test matrix classes.

8.5.2 Test Case 2

We consider now a synthetic network set up in the same fashion and increase the proportion of false-positive and false-negative connections. In this instance the probability of a connection from i to j is 0.8 for $i \in [1, 20]$ and $j \in [21, 40]$, and 0.4 if i and j are elsewhere. We construct the adjacency matrix and calculate the reordered mapped matrix as shown in Figure 8.22. We take the subgraph consisting of the first and last 20 nodes of the reordering, shown in Figure 8.23, and consider the members of S_1 and S_2. We find that 60% of nodes contained in S_1 and 60% of nodes in S_2 come from the sets we would expect (nodes $1-20$ and $21-40$ respectively) so the matrix mapping has recovered a reasonable proportion of the target nodes despite the high levels of noise.

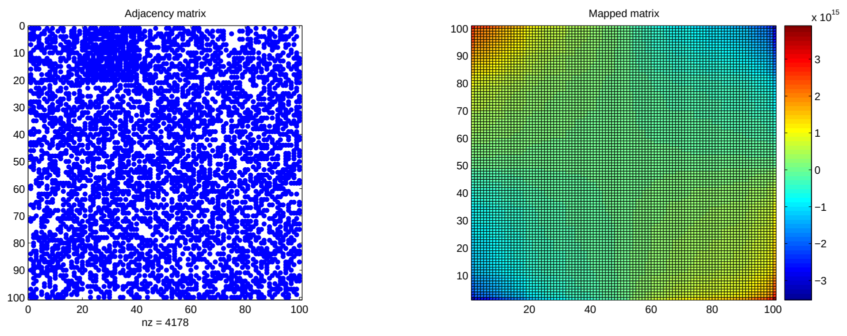


Figure 8.22: Test case 2: Adjacency matrix and mapped matrix.

We conduct statistical tests in the same manner as the previous example. The bipartivity measures for a sample of 1000 directed stickiness networks are plotted in the histogram in Figure 8.24 with the green circle denoting the bipartivity score for the synthetic network. Once again, it was observed that the bipartivity scores appear to follow a log-normal probability distribution, and this is plotted in Figure 8.25 together with a quantile-quantile plot of the bipartivity scores.

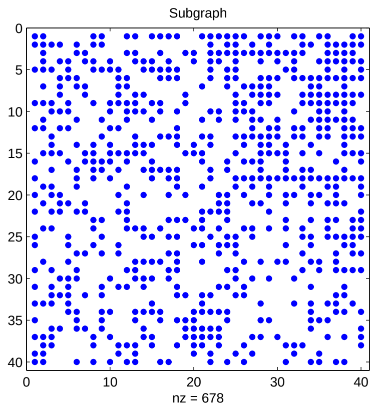


Figure 8.23: Test case 2: Subgraph.

The same procedure was repeated for the variants of the stickiness model and for Erdös-Rényi graphs, with the pvalues for each case listed in Table 8.2. The pvalues are larger (meaning the level of directed bipartivity in the subgraph is less significant) than those in the previous example due to the increased levels of noise in the network, although it should be noted that 60% of correct nodes were recovered. Although the pvalues for the directed stickiness and shuffled stickiness models are much larger than those in the previous example, the pvalues obtained when using the biased stickiness model or Erdös-Rényi model are still reasonably low. In the case of Erdös-Rényi, this may be attributed to the fact that any coherent structure such as directed bipartivity is unlikely to arise by chance, so the synthetic network will most likely have a higher bipartivity score despite the higher level of noise. The biased stickiness model works by assigning high out-degrees to nodes which also have high in-degrees, effectively destroying any structure present in the original network while attempting to preserve the degree distributions. As a result, directed bipartite structures are not preserved as in the stickiness and shuffled stickiness models, and so a lower pvalue is obtained.

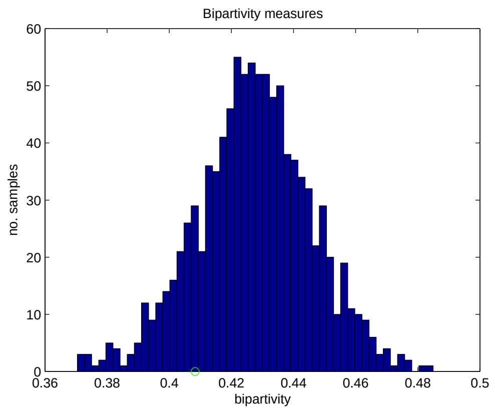


Figure 8.24: Test case 2: Histogram of bipartivity scores.

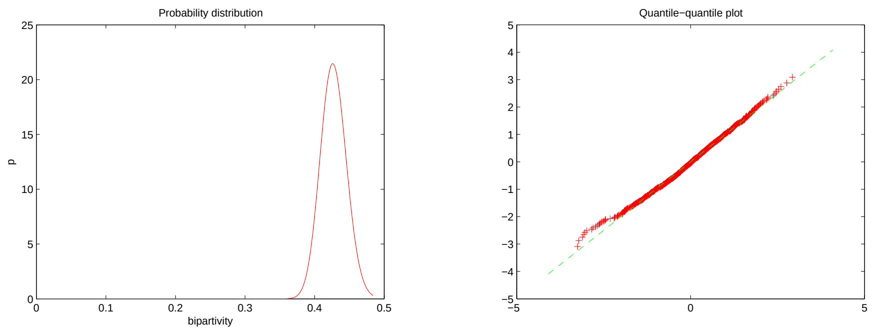


Figure 8.25: Test case 2: Probability distribution and quantile-quantile plot.

	pvalue 1	pvalue 2
Erdös-Rényi	5.91×10^{-2}	55/1000
Directed stickiness	8.46×10^{-1}	844/1000
Shuffled stickiness	9.03×10^{-1}	896/1000
Biased stickiness	7.00×10^{-2}	57/1000

Table 8.2: Significance of subgraph in test case 2 for varying test matrix classes.

8.6 *C. elegans* Neural Data

In order to assess the practical usefulness of the matrix mapping, we consider real world data associated with the nematode (roundworm) *Caenorhabditis elegans*. This organism is well studied and is a popular choice of test subject for applications in the field of network science. We will consider two networks of neural data: (i) the global neuronal network of the *c. elegans*, and (ii) a local subnetwork of 131 frontal neurons of the same organism. Our hope is that the matrix mapping will reveal patterns of directed bipartite connectivity known to exist between neurons without relying upon known biological information, i.e. using only the connectivity data.

In order to obtain a purely directed network, it was necessary to remove gap junctions from the dataset, since experimental techniques used to reconstruct the nervous system of *c. elegans* are unable to infer the directionality of such connections. After non-neuronal cells are removed, this results in a global network of 191 neurons and 1904 chemical synapses, and a local network of 131 neurons and 964 chemical synapses.

Our motivation is work by Durbin [24, Figure 8.1] in which an ad hoc combinatoric algorithm is used to search for the type of directed bipartite structure we consider. We aim to show that this matrix mapping gives an efficient and systematic way of identifying such types of structure.

8.6.1 Global Network

The adjacency matrix and mapped matrix for the global *c. elegans* network are shown in Figure 8.26. In Figure 8.27 we show a heat map of the mapped matrix reordered according to its first eigenvector. We can see that under the reordering, the hot and cold regions of the mapped matrix become tightly clustered at the extremities of the matrix. Based on the dimensions of these regions we make a decision on what size of subgraph to extract.

As we are considering only the network topology and not taking into account any existing biological information, we must consider the spectrum of bipartivity values over a variety of subgraph sizes. This is done by reordering the adjacency matrix based on the first eigenvector of the mapped matrix. Increasingly larger subgraphs are then considered by adding members to S_1 and S_2 until all sizes of subgraphs based on this ordering have been considered. The bipartivity scores are then considered while also taking into account subgraph size. That is to say a larger subgraph may be preferable to a smaller one even though it may have a smaller bipartivity score.

In this instance, the subgraph shown in Figure 8.28 was selected, with S_1 and S_2 having 15 and 8 members respectively. For this subgraph, the $S_1 \to S_2$ submatrix is respectively 3, 80 and 27 times more dense than the $S_1 \to S_1$,

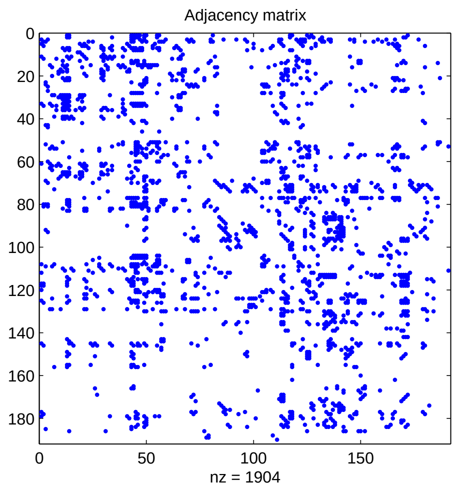


Figure 8.26: *C. elegans* global network: Adjacency matrix.

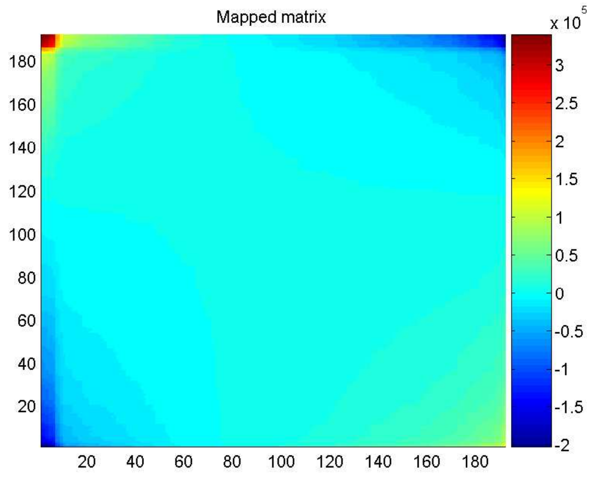


Figure 8.27: *C. elegans* global network: Mapped matrix.

$S_2 \rightarrow S_1$ and $S_2 \rightarrow S_2$ submatrices.

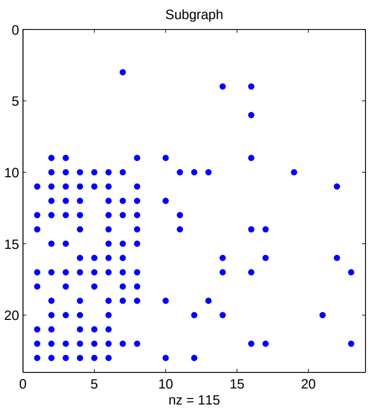


Figure 8.28: *C. elegans* global network: Subgraph.

We obtain a measure of statistical significance for this subgraph by considering the distribution of bipartivity scores for a sample of each type of random graph model described in Section 8.5. A histogram of bipartivity scores for a sample of directed stickiness matrices is shown in Figure 8.29. The bipartivity scores again were found to closely follow a log-normal probability distribution shown in Figure 8.30 together with a quantile-quantile plot confirming the similarity. The subgraph taken from the global network has a bipartivity score of 0.6415 and this level of bipartivity is unlikely to arise by chance in any of the network types considered. The pvalues for each class of synthetic network are listed in Table 8.3.

The subgraph we consider can be deemed significant for all variants of the stickiness model as well as for the Erdös-Rényi model, although the in and out degree distributions of the *c. elegans* neural network are known to differ dramatically from the Poissonian distribution of Erdös-Rényi random graphs, so significance in this test may be less meaningful.

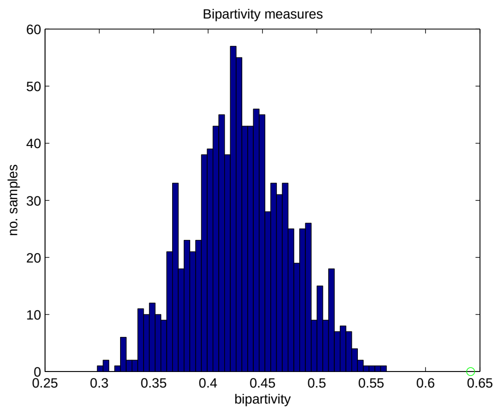


Figure 8.29: *C. elegans* global network: Histogram of bipartivity scores for a sample of directed stickiness networks.

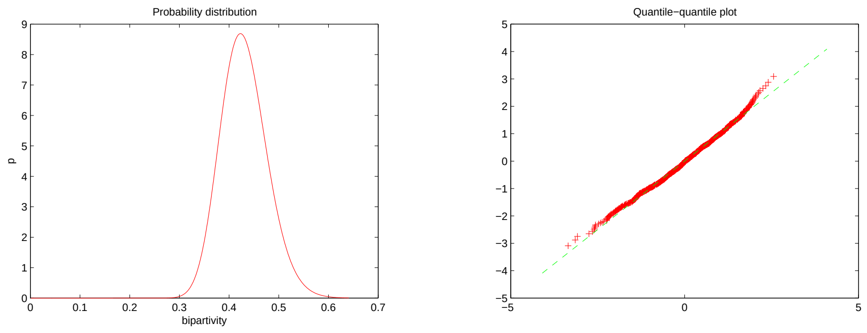


Figure 8.30: *C. elegans* global network: Probability distribution and quantile-quantile plot.

	pvalue 1	pvalue 2
Erdös-Rényi	8.41×10^{-12}	0/1000
Directed stickiness	8.79×10^{-5}	0/1000
Shuffled stickiness	5.95×10^{-5}	0/1000
Biased stickiness	3.82×10^{-6}	0/1000

Table 8.3: Significance of subgraph in *c. elegans* global network for varying test matrix classes.

In Table 8.4 we list the neuronal classes of the members of the subgraph. We see that S_1 consists of sensory neurons and nerve ring interneurons while S_2 consists entirely of command interneurons. This is in good agreement with the hierarchical ordering presented by Durbin.

8.6.2 Local Network

We consider now a local network consisting of 131 neurons and seek to identify a directed bipartite subgraph. The adjacency matrix of this network and the reordered mapped matrix are shown in Figures 8.31 and 8.32. Once again, the hot and cold regions have been clustered in the corners of the mapped matrix and we select nodes from these to form S_1 and S_2. In this instance, a subgraph consisting of 32 nodes (16 in S_1 and 16 in S_2) has been selected and its adjacency matrix is displayed in Figure 8.33.

In this instance, we note that the although the $S_1 \rightarrow S_2$ submatrix is respectively 5, 35 and 9 times more dense than the $S_1 \rightarrow S_1$, $S_2 \rightarrow S_1$ and $S_2 \rightarrow S_2$

	Neuronal Class	Description
S_1	DVA	Interneuron
	FLP	Sensory neuron
	DVC	Ring interneuron
	PVP	Interneuron
	ADL	Amphids; sensory neuron
	AIM	Ring interneuron
	ADE	Anterior deirid; sensory neuron
	ASH	Amphids; sensory neuron
	AQR	Sensory neuron
	ADA	Ring interneuron
S_2	AVA	Command interneuron
	AVB	Command interneuron
	AVD	Command interneuron
	AVE	Command interneuron

Table 8.4: Neuronal class and type for bipartite subgraph found in the global network of 191 neurons of the *C. elegan*.

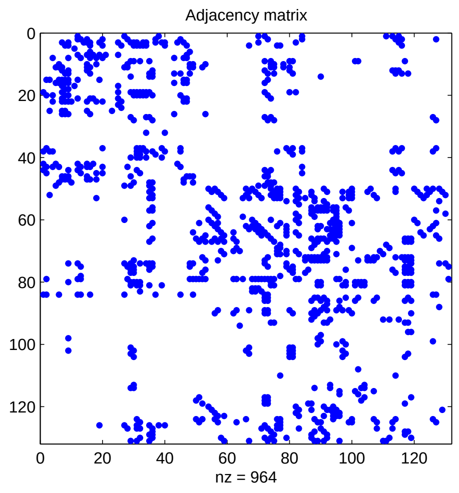


Figure 8.31: *C. elegans* local network: Adjacency matrix.

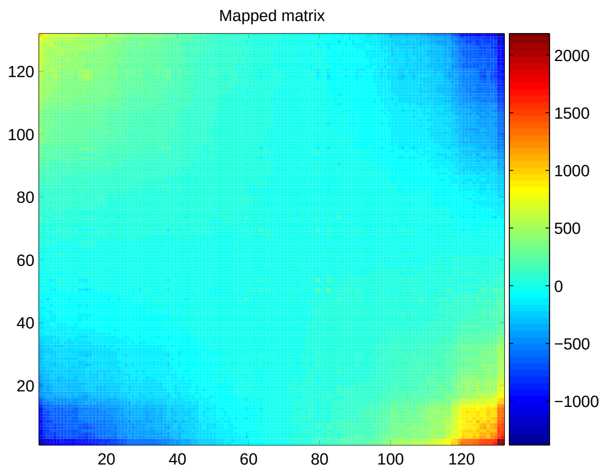


Figure 8.32: *C. elegans* local network: Mapped matrix.

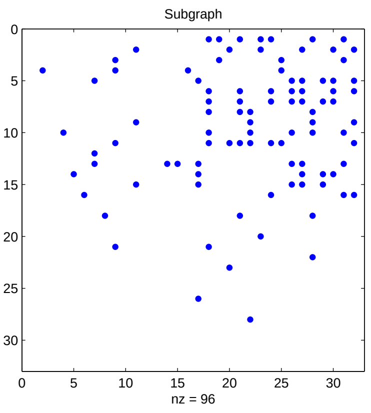


Figure 8.33: *C. elegans* local network: Subgraph.

submatrices, this region is nevertheless relatively sparse and thus the subgraph has a low bipartivity score of 0.2645. As with previous tests, we obtain a measure of statistical significance by plotting the bipartivity scores of a sample of synthetic networks in a histogram, as seen in Figure 8.34 and attempting to match a probability distribution to the distribution of bipartivity scores. A log-normal distribution was again found to be a good fit to the sampled data, and this is plotted in Figure 8.35 together with a quantile-quantile plot confirming that the probability distribution fits the data well.

This procedure was repeated for the variants of the directed stickiness model and for Erdös-Rényi random graphs. The pvalues found in each case are listed in Table 8.5. Unlike the results for the global *c. elegans* network, the subgraph found in this case was only deemed significant when comparisons were made with Erdös-Rényi random graphs and with the biased stickiness model. Despite the low score under our measure of bipartivity, the neurons contained within the subgraph are again of biological interest. The neuronal classes found within S_1 and S_2 are listed in Table 8.6. Approximately 65% of neurons found in S_1

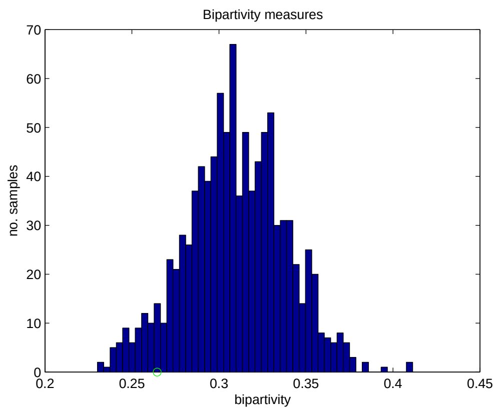


Figure 8.34: *C. elegans* local network: Histogram of bipartivity scores for a sample of directed stickiness networks.

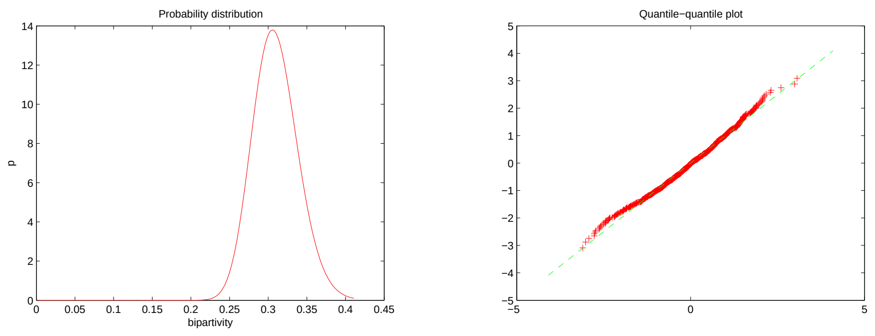


Figure 8.35: *C. elegans* local network: Probability distribution and quantile-quantile plot.

are involved in sensory processes while S_2 is comprised of a mixture of motor neurons and command interneurons. Once again, this is in good agreement with the work by Durbin where an attempt is made to hierarchically display the neurons in such a way that as many synapses as possible point downwards. In this ordering, sensory neurons were placed near the top, motor neurons at the bottom and the remaining interneurons in between.

	pvalue 1	pvalue 2
Erdös-Rényi	9.02×10^{-5}	0/1000
Directed stickiness	9.48×10^{-1}	935/1000
Shuffled stickiness	9.69×10^{-1}	966/1000
Biased stickiness	5.04×10^{-2}	40/1000

Table 8.5: Significance of subgraph in *c. elegans* local network for varying test matrix classes.

On closer inspection of both the local and global *c. elegans* networks, it was found that an important set of neurons were identified in both cases. Around 60% of neurons belonging to S_2 in the local network, and all neurons belonging to S_2 in the global network were found to be members of a group of neurons known as the *lateral ganglion* which are known to be highly interconnected with both sensory and motor neurons. It has been suggested that the lateral ganglion is in fact the principal pathway between sensory and motor components of the nematode *c. elegans* [14].

In addition, the neuronal classes AVA, AVB, AVD and AVE were picked

	Neuronal Class	Description
S_1	OLL	Head sensory neuron
	URY	Head sensory neuron
	IL2	Head sensory neuron
	RIH	Ring interneuron
	ASH	Amphids; sensory neuron
	RIM	Ring motor neuron
	RIV	Ring motor/interneuron
	CEP	Head sensory neuron
	AVH	Interneuron
	ADL	Amphids; sensory neuron
S_2	SMD	Ring motor neuron
	RME	Ring motor neuron
	RMD	Ring motor neuron
	AVB	Command interneuron
	AVA	Command interneuron
	AVE	Command interneuron
	AVD	Command interneuron

Table 8.6: Neuronal class and type for bipartite subgraph found in the local network of 131 frontal neurons of the *C. elegan.*

out both in the local and global networks. These have been identified as ‘hub’ neurons that are crucial to normal biological function. For instance, it is well known that both AVA and AVB neurons are necessary for normal coordinated movement [64].

8.7 Application to Other Networks

In addition to the *c. elegans* neural network, the matrix mapping was applied to the networks of Scottish football transfers from NESSIE as described in Section 5.3.5. In each of these networks, each node represents either a Scottish football club or a club involved in a transfer with one, and an edge from node i to node j indicates that a player was transferred from club i to club j in a particular season. The network is directed but has been made unweighted (so that edges represent only the existence of transfers between two clubs, rather than counting the number of transfers between them) and may also be incomplete since only edges involving at least one Scottish club are included. So, for instance, transfers from Parma or Coventry City to Kilmarnock F.C. are included but any transfers between Parma and Coventry City are not.

8.7.1 Scottish Football Transfers 2008-2009

We first consider the network obtained by recording transfers to and from Scottish football clubs in the summer transfer period of the 2008-2009 season. Figure 8.36 shows a spy-plot of this network, together with a heat map of the mapped matrix $f(A) + f(A^T)$. The adjacency matrix itself is very sparse and as a result we do not expect to find a single large bipartite structure but perhaps many small ones. The mapped matrix has been reordered according to its first eigenvector and small ‘hot’ regions appear to be clustered on the leading diagonal. By taking the first and last three nodes of this reordering we obtain the subgraph shown in Figure 8.37 which has a bipartivity score of 0.7500.

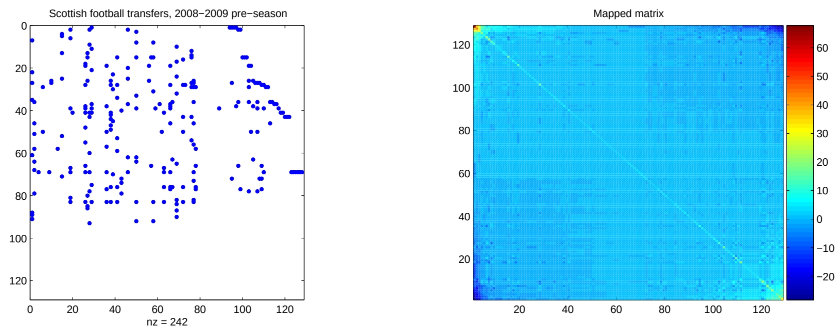


Figure 8.36: Network of Scottish football transfers in the 2008-2009 pre-season and heat map of mapped matrix.

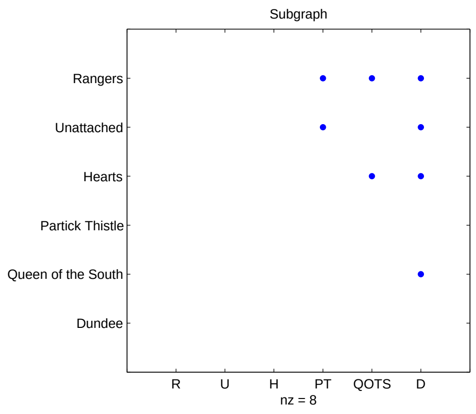


Figure 8.37: Subgraph obtained from the first and last 3 nodes in the heat map in Figure 8.36.

This shows a small directed bipartite structure comprised of movements of unattached players and players from SPL clubs to First Division teams. A common career pattern in players for top-division teams involves loan-spells at lower division clubs, and on further investigation of the players involved, we find that all but one of the players in the subgraph moving from SPL sides did so on loan deals. In Table 8.7 we show the pvalues given by applying the form of testing described in Section 8.5. The results are qualitatively similar in the sense that the subgraph scores well when compared with subgraphs drawn from Erdös-Rényi graphs or the biased stickiness model but less well when compared with the stickiness or shuffled stickiness models. However, in a network of this type we expect few reciprocal links (since it is unusual for transfers in both directions between two clubs) and so there is an underlying 'flow' in the network and we expect the stickiness model to replicate this. We now consider the networks of both transfer periods in the 2007-2008 season and attempt to discover a similar pattern.

	pvalue 1	pvalue 2
Erdös-Rényi	6.70×10^{-2}	22/1000
Directed stickiness	1.57×10^{-1}	95/1000
Shuffled stickiness	3.92×10^{-1}	444/1000
Biased stickiness	9.77×10^{-2}	31/1000

Table 8.7: Significance of subgraph in Figure 8.37 for varying test matrix classes.

8.7.2 Scottish Football Transfers 2007-2008

In addition to the network of transfers in the summer before the 2008-2009 football season, we have the corresponding network for the previous year as well as the network of transfers made during the mid-season transfer window. We expect to find a similar subgraph to the one obtained in the previous section for the summer transfer window. Mid-season transfers, however, tend to follow different patterns as clubs typically attempt to sign players that will improve their current standing in their respective league (i.e. a short-term fix rather than a long-term investment). We first consider the transfers made during the summer. Figure 8.38 shows the network of transfers made during the pre-season window, as well as a heat map of the mapped matrix. Taking the first and last three nodes of the reordered mapped matrix, we obtain the subgraph shown in Figure 8.39 which has a bipartivity score of 0.6667.

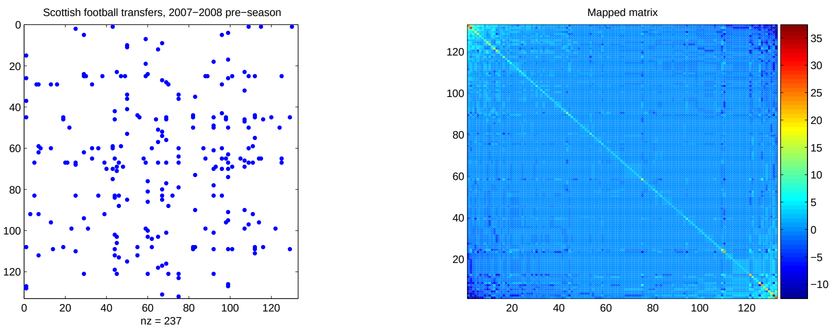


Figure 8.38: Network of Scottish football transfers in the 2007-2008 pre-season and heat map of mapped matrix.

This subgraph consists of transfers from three Premier League clubs to three clubs in various divisions. Three of the transfers were free (players arriving at the end of their contracts), three were loan deals and one involved a transfer

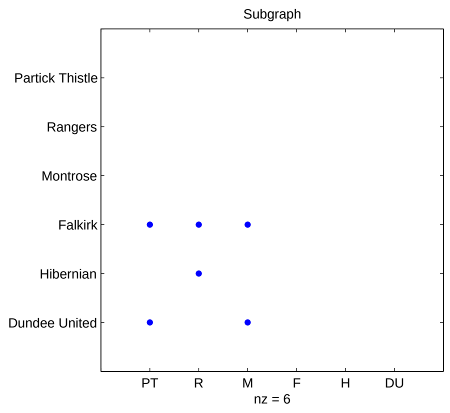


Figure 8.39: Subgraph obtained from the first and last 3 nodes in the heat map in Figure 8.38.

fee. (Although there are only six edges in the subgraph, one edge represents the transfer of two players.) The pvalues given by comparing the bipartivity score of this subgraph with the scores of subgraphs drawn from varying classes of test network are shown in Table 8.8.

	pvalue 1	pvalue 2
Erdös-Rényi	1.37×10^{-1}	33/1000
Directed stickiness	3.74×10^{-1}	387/1000
Shuffled stickiness	6.33×10^{-1}	611/1000
Biased stickiness	1.84×10^{-1}	135/1000

Table 8.8: Significance of subgraph in Figure 8.39 for varying test matrix classes.

Finally we consider the mid-season transfers in 2007-2008. As contracts tend to terminate at the end of a season and clubs have other financial concerns, fewer transfers tend to take place in this period. Figure 8.40 shows the network of transfers during this period together with a heat map of the reordered, mapped matrix. Note that there are fewer nodes than in previous examples. By taking the first two and last three nodes of the reordering, we obtain the subgraph shown in Figure 8.41.

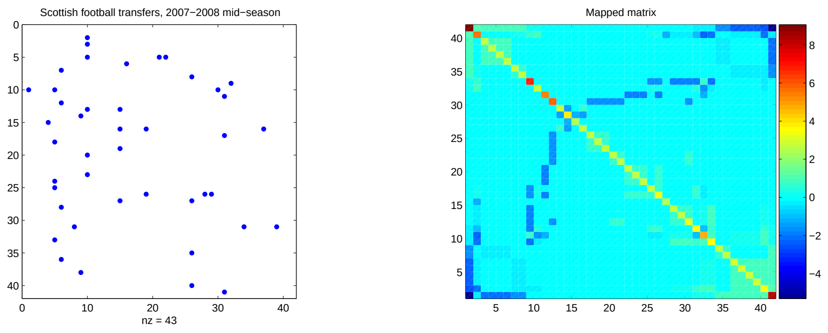


Figure 8.40: Network of Scottish football transfers in the 2007-2008 mid-season and heat map of mapped matrix.

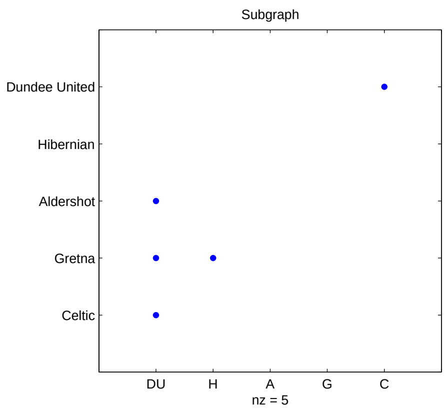


Figure 8.41: Subgraph obtained from the first 2 and last 3 nodes in the heat map in Figure 8.40.

In this instance we find an approximate directed bipartite subgraph consisting of transfers from several clubs to Dundee United (with an additional transfer from Dundee United to Celtic) and one transfer to Hibernian. In contrast to previous examples, two transfers involved fees, two were free transfers and one was a loan deal. Additionally, the reciprocal link between Celtic and Dundee United is of interest: a player was released from one club to the other on loan as part of the agreement to secure the transfer of another player in the opposite direction. This network is smaller than other examples (since fewer transfers take place in the mid-season window) and is very sparse. This is reflected in Table 8.9 where for each test model, many of the random matrices contained subgraphs with higher bipartivity scores.

	pvalue 1	pvalue 2
Erdös-Rényi	3.07×10^{-1}	383/1000
Directed stickiness	4.17×10^{-1}	475/1000
Shuffled stickiness	6.40×10^{-1}	746/1000
Biased stickiness	4.24×10^{-1}	415/1000

Table 8.9: Significance of subgraph in Figure 8.41 for varying test matrix classes.

Although we have identified small approximately bipartite subgraphs for each of the football networks, we must conclude that none of the patterns of connectivity are statistically significant when compared with Erdös-Rényi graphs or the directed stickiness model. In each instance we have a pvalue greater than 0.05, suggesting that directed bipartite structures of this size occur too frequently within random directed networks to be deemed significant. A possible direction further investigations could take would be to consider the evolving network where both clubs and players are nodes, and a player and club are connected if the player is currently under contract with the club. It might also be possible that a significant 'directed partite' structure exists if we consider weighted edges that reflect financial transactions.

8.8 Conclusions

We have addressed the problem of discovering directed bipartite structures within complex networks via a new matrix mapping. Initial tests on networks from neuroscience show that the new mapping can be used to infer biologically relevant information using only network topology. Investigation of a set of networks consisting of players transferring between football clubs provided a novel example where the patterns in a subgraph could be explained in part by considering additional information about the network. We have emphasised the impact of the choice of random graph model by comparing statistics for several such models. In particular, we see that the 'significance' or 'non-significance' of the determined connectivity patterns can be extremely sensitive to the class of random matrices chosen.

Chapter 9

Comparison Between SVD and Mapping Approaches

Having established in Chapters 6 and 8 two procedures for identifying directed bipartite structures within complex networks, we wish to make a brief comparison of their performances under synthetic conditions. Although the SVD algorithm and the matrix mapping both attempt to highlight members of directed bipartite communities, we stress that they are applicable in different general circumstances. The SVD algorithm focuses more on the 'lock and key' interpretation of connectivity, with each pair of singular vectors potentially containing information about the distribution of a different type (or 'colour') of lock and key. The matrix mapping, on the other hand, provides a useful reordering of the entire adjacency matrix, and the weighted entries, as interpreted as a measure of 'similarity' of nodes in terms of bipartite community membership, may be useful in showing overlapping community structure, particularly when plotted as a heat map.

9.1 Resilience to Noise

We compare the methods' resilience to increasing levels of 'noise' in a synthetic matrix by applying the same sort of testing used in Section 6.4. We construct a synthetic matrix consisting of 100 nodes, where there is an underlying bipartite structure involving sets S_1 and S_2, both consisting of 10 nodes. Adjacency matrices are generated randomly such that the independent probability of a connection between a node in S_1 to a node in S_2 is p_1 and the independent probability of a connection elsewhere is p_2.

We fix p_1 and allow p_2 to vary from 0 to p_1. For each value of p_2, we compute the proportion of nodes correctly identified by the SVD and the matrix mapping, and average over ten random graph instances. Figures 9.1 and 9.2 show the proportion of nodes correctly identified by the SVD and the matrix mapping as 'keys' and 'locks' respectively for p_1 fixed at 0.3. We observe that both methods have qualitatively and quantatively similar performances. Similarly, Figures 9.3 and 9.4 show the proportions of 'keys' and 'locks' correctly identified for p_1 fixed at 0.8. Again, both methods have comparable levels of accuracy.

9.2 Run-times

We consider the processing times for the SVD compared with our matrix mapping by computing random matrices of increasing size with a fixed level of sparsity. Figure 9.5 shows a log-log plot of the processing times for the SVD (blue asterisks) and the matrix mapping (red asterisks) together with the matrix generation times (green asterisks). The sparsity for each matrix is fixed at 0.01. By inspection, we see that the processing times follow a power law (with positive

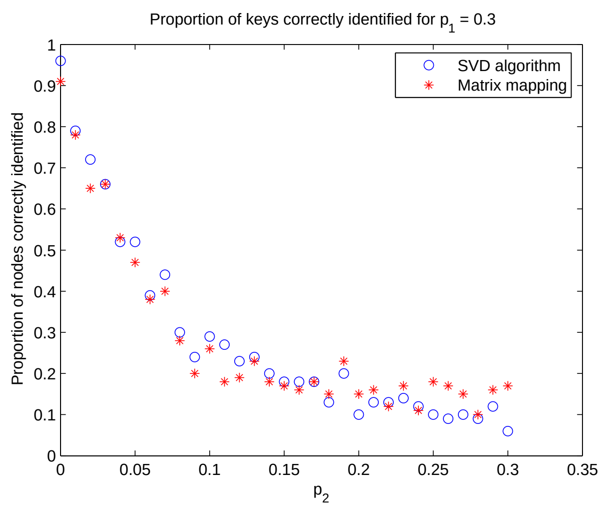


Figure 9.1: Proportion of key nodes correctly identified by first left singular vector of the adjacency matrix and first eigenvector of the mapped matrix for $p_1 = 0.3$.

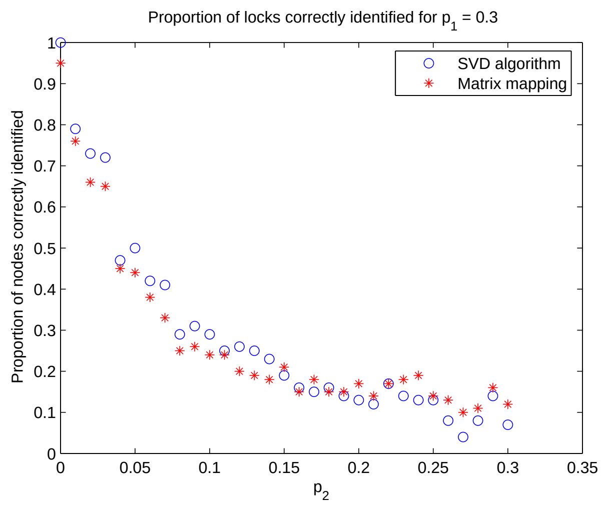


Figure 9.2: Proportion of lock nodes correctly identified by first left singular vector of the adjacency matrix and first eigenvector of the mapped matrix for $p_1 = 0.3$.

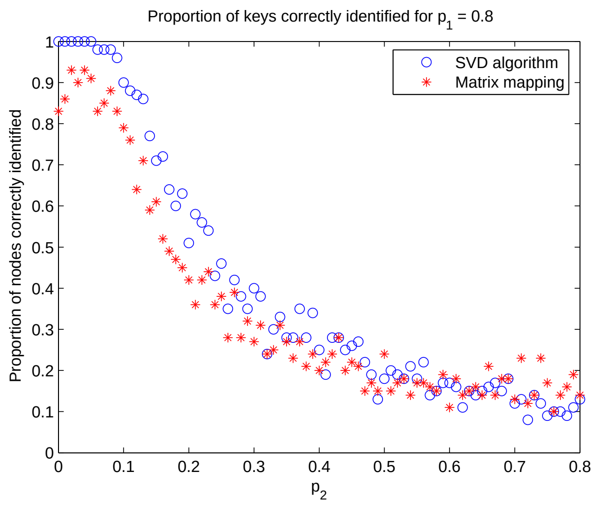


Figure 9.3: Proportion of key nodes correctly identified by first left singular vector of the adjacency matrix and first eigenvector of the mapped matrix for $p_1 = 0.8$.

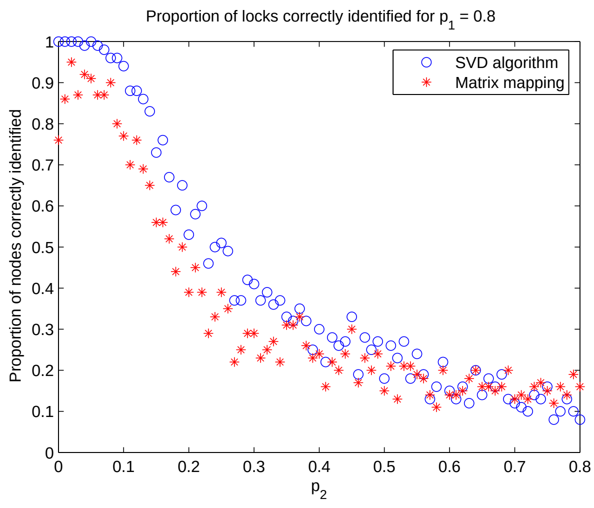


Figure 9.4: Proportion of lock nodes correctly identified by first right singular vector of the adjacency matrix and first eigenvector of the mapped matrix for $p_1 = 0.8$.

exponent) and that the matrix mapping has a consistently longer run-time than the SVD. Such a result is predictable since each matrix mapping involves the computation of two singular value decompositions, one for the adjacency matrix A and one for A^T. Increasing the sparsity to 0.5, we observe in Figure 9.6 that the results are both quantatively and qualitatively similar.

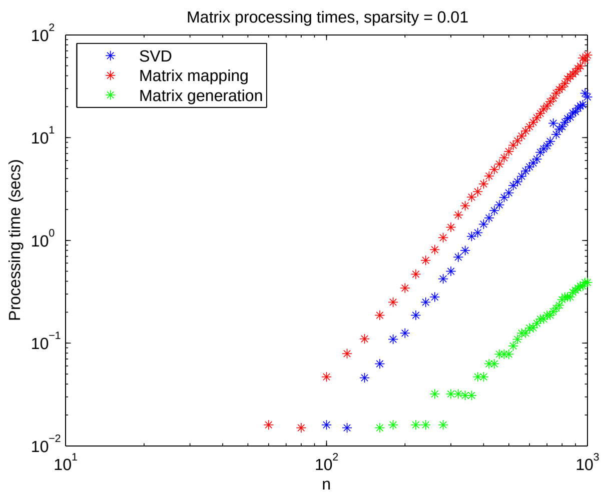


Figure 9.5: Matrix processing times with sparsity = 0.01.

9.3 Application to Real Data

Although both methods perform comparably in the test described in Section 9.1, the SVD takes typically half the time needed to compute the matrix mapping. The post-processing for both methods may differ, however. Having computed the matrix mapping, we obtain a symmetric, real valued matrix which may be used as input for a clustering method selected by the user. The values in the mapped matrix represent a measure of similarity of a sort between two nodes, and as such conventional clustering algorithms may be exploited to search for

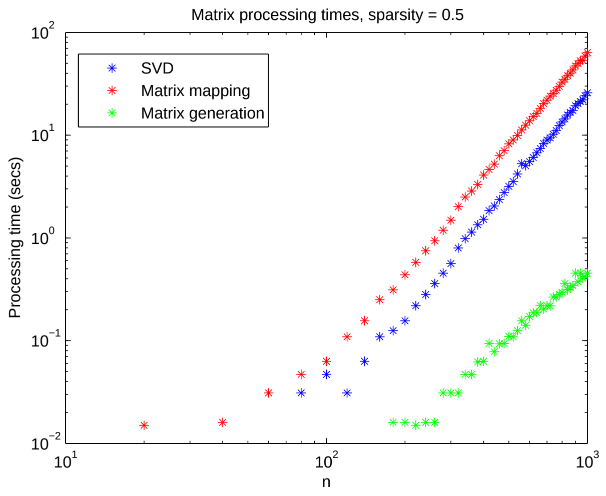


Figure 9.6: Matrix processing times with sparsity = 0.5.

bipartite structures within the data. For instance computing the first eigenvector of the mapped matrix may provide a reordering of the nodes that reveals an underlying bipartite structure.

The SVD algorithm, on the other hand, is more applicable if we believe there to be a number of different (possibly overlapping) bipartite communities. In this case, several singular vector pairs may contain useful information and a more exhaustive search may be required. There are however visualization issues when using the SVD to investigate bipartite communities. Since the left and right singular vectors may contain information about 'keys' and 'locks' respectively, separate row and column reorderings may be the clearest way to show bipartite communities via a spy-plot, although this raises questions about the relative positions of individual nodes in both reorderings. Several spy-plots or subgraphs may have to be investigated to discover all patterns of bipartite connectivity within a given network. In this respect, the matrix mapping has the advantage of producing a single, symmetric matrix which may then be clustered conventionally.

Chapter 10

Conclusions and Future Work

In this thesis we have presented two main topics concerning the search for a specific form of substructure within complex networks, i.e. directed bipartite connectivity. Additionally we have presented two packages to aid in the analysis of complex networks: CONTEST, a controllable test matrix toolbox for MATLAB, and NESSIE, a repository of real world network data. We summarize each of these topics, discuss the conclusions we can draw in each case and discuss the directions in which each topic could be extended in the future.

10.1 CONTEST: A Controllable Test Matrix Toolbox for MATLAB

In recognition of the fact that recent random network models make excellent candidates for sparse test matrices, we have implemented a number of these models in MATLAB. The CONTEST toolbox allows the generation of large sparse matrices as instances of random graphs drawn from one of nine models. The models are controlled by a number of parameters allowing matrices of varying size and sparsity to be tailored to the end-user's particular application. Additionally, since the implementation of each model uses MATLAB's own random number generators `rand` and `randn`, a particular test matrix may be exactly reproduced by re-setting their states. Naturally occuring computational tasks in network science present challenging problems for general (symmetric and unsymmetric) linear systems solvers and symmetric eigenvalue routines.

As an example of the application of these models, a directed version of the Erdös-Rényi code and an extension of the stickiness model have been used extensively in Chapters 6 and 8 this work as test matrices for the statistical analysis of the success of two methods for discovering directed bipartite substructure.

There are a number of ways in which the MATLAB codes in the toolbox may be extended. As previously stated, we focussed on the challenge of generating instances of large matrices than optimizing the speed or efficiency of the algorithms. A potential improvement to the models would be to implement multiple algorithms within a single code and to select the appropriate one based upon the dimension n of matrix to be generated, or to allow the user to select the appropriate algorithm. There are also a number of measures on networks that could potentially be added as utilities in CONTEST, including various measures of centrality or graphlet frequencies.

The models in CONTEST generate adjacency matrices corresponding to undirected networks. We have stated that in some cases a directed variation can be produced by simply combining the upper and lower triangles of separate instances, though in some modelling scenarios this may be inappropriate (as well as inefficient). A potential improvement could be to include an input parameter that allows the user to toggle between directed and undirected versions of the model. In Chapter 7, for instance, we showed how the stickiness model can be extended to the directed case (subject to certain constraints).

A final and perhaps more challenging improvement to the codes in CONTEST would be their implementation using MATLAB's parallel processing capabilities. Applications in network science typically involve lengthy computational processes, and harnessing the potential of parallel processing could allow for faster and more thorough testing. Similarly, the processing time taken to generate large instances of random matrices in CONTEST could be greatly improved by these methods.

10.2 NESSIE: Network Example Source Supporting Innovative Experimentation

We have presented a repository of network data from a variety of sources that may prove useful as a test set for algorithms in network science. The datasets span a variety of application areas, and include directed, undirected, weighted and unweighted networks. At the moment NESSIE consists of 12 networks but this could easily be extended. Network data is available from a variety of sources and may be easily converted into adjacency matrix format. For instance, the website

`http://www.trafficscotland.org`

currently contains regularly updated tables of journey times on particular roads in Scotland. This represents a frequently evolving network that is both weighted and directed and as such may form a particularly interesting dataset. This would allow investigation of a variety of interesting problems, for instance can we predict the effect a delay on a particular road will have on journey times using other nearby roads?

10.3 Discovering Directed Lock and Key Structure

In Chapter 6 we showed that if a network has an underlying directed bipartite structure, the left and right singular vectors of the adjacency matrix A give useful information when interpreted as the distribution of 'locks' and 'keys' among the nodes. Furthermore, we observed that if there are overlapping patterns of directed bipartite connectivity (which we interpreted as different 'colours' of locks and keys), then additional singular vector pairs may give information corresponding to these.

We tested the effectiveness of subgraph reorderings based on components of the singular vectors by applying the SVD to a variety of synthetic test cases in which multiple lock and key structures were embedded. We also investigated the effectiveness of the SVD at identifying nodes belonging to bipartite communities under varying levels of artificial 'noise' (false negatives and false positives in the adjacency matrix), and found the SVD to be tolerant up to modest levels of noise.

Finally we applied the SVD to two directed networks from real world sources; a network relating to co-expression levels of genes related to p53 and another representing a sociological study of coworkers giving and receiving help in an office environment. In the case of the p53 dataset, we identified an approximate directed bipartite subgroup consisting of two sets of genes. The first set was found to be mainly comprised of genes that have an inhibitory effect on cell growth and the second was found to comprise genes related to cell division and cell growth. Discovering this subgroup from the connectivity information alone supports the idea that the concept of directed bipartivity is relevant for gene co-expression networks. In this instance, a nonzero in position $A_{i,j}$ means that when gene i is expressing above the normal level, gene j is simultaneously expressing below the normal level. In terms of the specific subgroup we have identified, this means that when the set of genes inhibiting cell growth are expressing more than usual, the set of genes concerned with promoting cell growth and cell division are expressing less than usual. Biologically, this adds useful information as the subgroup consists largely of genes where such a pattern of connectivity is expected and can be easily explained, but we also pulled out a small number of genes who's role is currently unclear. We are therefore able to assign likely functional roles to these genes, and highlight them as candidates for experimental studies.

We believe that further improvements to the method of identifying approximate directed bipartite subgraphs using the SVD will be algorithmic rather than mathematical. As has been stated, multiple singular vector pairs may provide information about connectivity that can be interpreted as overlapping substructures. An exhaustive search of all potential reorderings of the adjacency matrix based on the components of the singular vectors is unfeasable for networks of even modest size. A possible direction for future development would be to produce a program that examines the subgraphs obtained by extracting nodes based on their corresponding components in several left and right singular vectors, perhaps applying a qualitative measure of bipartivity such as the one described in Section 8.5. By examining several such sugraphs, the 'most bipartite' ones could be output.

When testing the SVD algorithm, we also found that on some occasions, linear combinations of singular vectors provided more insightful reorderings of the adjacency matrix, particularly in synthetic examples consisting of a number of different lock and key pairs. A possible direction of future research could be to attempt to justify the 'refinement' of information in a single reordering vector by updating based on the entries of several singular vectors. This could potentially allow reorderings of the entire adjacency matrix (rather than focussing on subgraphs) that highlight multiple approximate directed bipartite structures.

10.4 Mapping Directed Networks

In Chapter 8, we introduced the concept of an *alternating walk* as a means of assessing the 'similarity' of a pair of nodes if an approximate directed bipartite connectivity structure is assumed. By counting alternating walks of increasing length and scaling appropriately to give less weight to longer walks, we arrived at an expression for the similarity of two nodes as the difference between the total number of even and odd length alternating walks. Furthermore, we showed that such a count of alternating walks on the transpose of the adjacency matrix gives information of a similar type, and that the sum of these two expressions results in a mapping from an unsymmetric binary matrix to a symmetric real-valued matrix where each entry expresses the similarity of two nodes in terms of a count of alternating walks.

We tested this matrix mapping by applying it to a number of synthetic matrices with varying degrees of bipartite connectivity built in and observed that the mapped matrix was effective in identifying members of directed bipartite communities and could form the basis for a process that aims to uncover this type of structure in a network. We also showed that the matrix mapping was more effective than the matrix exponential or negative matrix exponential at highlighting patterns of directed bipartite connectivity within a network.

The matrix mapping was applied to networks from two sources; neural networks of the *c. elegans* organism and networks of transfers between football clubs taken from NESSIE. Taking as our motivation the work by Durbin [24] in which a combinatoric algorithm is used to reorder neurons hierarchically, we attempted to find approximate directed bipartite subgraphs which could be verified by existing knowledge of the connectivity of neurons in *c. elegans*. The results were encouraging. They involved neurons belonging to the *lateral ganglion* which is thought to be the principal pathway between sensory and motor neurons in *c. elegans*. We conducted statistical tests of the subgraphs identified by formulating a simple measure of bipartivity and comparing with the bipartivity of subgraphs drawn from Erdös-Rényi graphs and variations of a directed extension of the stickiness model in [76]. We found in the majority of cases that subgraphs of comparable bipartivity as those extracted from the neuronal networks were significantly unlikely to appear in random networks. We also considered as an additional example three networks consisting of player transfers between football clubs. These networks were particularly sparse and only small approximately bipartite subgraphs could be extracted. Although these subnetworks could be interpreted as picking out sensible patterns, the results were not found to be statistically significant.

As with the SVD algorithm, the work on the matrix mapping could be improved by further automating the process. As it stands, a significant level of 'eyeballing' must take place when selecting an appropriate number of nodes from the mapped matrix. We have experimented with programs which compute many potential subgraphs, assess their level of bipartivity and select the largest of them. Such a method, however, is computationally expensive and may result in lengthy processing times even though a significantly bipartite subgraph may not exist within the network. A method that approximates the change in bipartivity of a subgraph by the addition or removal of a particular node would be a useful extension of this work.

Both the SVD and matrix mapping approaches have been applied primarily in the search for bipartite substructures with non-overlapping communities. A problem that often occurs in practice is that such partitions in networks are not necessarily disjoint. This is analagous to a group of nodes having multiple "locks" or "keys" in the definition in Chapter 6. One of the most important extensions to this work would be the further consideration of methods for detecting overlapping community structure within directed networks.

Appendix A

CONTEST Code

We present here the MATLAB codes for each random graph model and utility implemented in CONTEST. An introductory section in each code describes the input arguments and the default values they take, the format of the output and a typical call of each program.

A.1 baitsample.m

```
function B = baitsample(A,bait,prey)

%BAITSAMPLE     bait and prey subsampling
%
%   Input   A: n by n adjacency matrix
%           bait: proportion of nodes to sample. Defaults to 0.5.
%           prey: proportion of edges to retain from each "bait" node.
%                 Defaults to 0.5.
%
%   Output  B: adjacency matrix with the attribute sparse.
%              Dimension of B cannot be predicted.
%
%   Description:    The adjacency matrix A is considered, and a
%                   proportion, bait, of its rows/columns are retained.
%                   Of these, a proportion, prey, of the outgoing edges
%                   are retained. All other entries are set to zero.
%                   Disconnected nodes are then removed.
%
%   Reference:  J. Han, D. Dupuy, N. Bertin, M. Cusick, M. Vidal,
%               Effect of sampling on topology predictions of
%               protein-protein interaction networks,
%               Nature Biotechnology 23 (2005), pp. 839-844.
%
%   Example: B = baitsample(A,0.1,0.4);
```

```
if nargin <= 2
    prey = 0.5;
    if nargin == 1
        bait = 0.5;
    end
end

n = length(A);
B = sparse(n,n);

rp = randperm(n);
ibait = rp(1:ceil(bait*n));     % bait rows/columns to be retained

for i = 1:length(ibait)

    B(ibait(i),:) = A(ibait(i),:);
    B(:,ibait(i)) = A(:,ibait(i));
    iprey = find(B(ibait(i),:));
    psum = ceil(prey*length(iprey)); % number of prey nodes for ith bait

    if length(iprey) ~= 0
        dist = (1/length(iprey)) : (1/length(iprey)) : 1;

        while length(find(B(ibait(i),:))) > psum

            r = rand;
            pos = min(find(r<=dist));
            B(ibait(i),iprey(pos))=0;
            B(iprey(pos),ibait(i))=0;

        end

    end

end

for i = 1 : n               % trim isolated nodes
if ~any(B(:,(n-i+1)))
B(:,(n-i+1)) = [];
B((n-i+1),:) = [];
end
end
```

A.2 curvature.m

```
function C = curvature(A,ind)

%CURVATURE  Compute the curvatures (clustering coefficients) for a given
%           adjacency matrix.
%
%   Input   A:   n by n adjacency matrix.
%           ind: node index (optional). Can also take string values
%                'max' for maximum and 'ave' for average, ignoring those
%                that are undefined.
%
%   Output  C: n by 1 vector of curvatures (scalar if ind is provided).
%              Undefined values are returned as NaN.
```

```
%
%   Description:    Calculates the curvatures of the nodes in a graph.
%                   Defined for each node by the frequency with which
%                   its neighbours are themselves connected.
%
%   Examples: C = curvature(A)           vector of curvatures
%             C = curvature(A,4)         curvature of 4th node
%             C = curvature(A,'max')     largest curvature
%             C = curvature(A,'ave')      average curvature

n = length(A);

v = sum(A);

b = diag(A^3);
d = v.*(v-1);

ccoefs = zeros(1,n);

for i = 1:n
    if d(i) <= 1
        ccoefs(i) = NaN;
    else
        ccoefs(i) = b(i)/d(i);
    end
end

if nargin==1
    C = ccoefs;
else
    if strcmp(ind,'max')
        C = max(ccoefs);
    else

        if strcmp(ind,'ave')
            C = mean(ccoefs(find(~isnan(ccoefs))));
        else

            C = ccoefs(ind);
        end
    end
end
```

A.3 erdrey.m

```
function A = erdrey(n,m)

%ERDREY     Generate adjacency matrix for a G(n,m) type random graph.
%
%   Input   n: dimension of matrix (number of nodes in graph).
%           m: 2*m is the number of 1's in matrix (number of edges in graph).
%           Defaults to the smallest integer larger than n*log(n)/2.
%
%   Output  A: n by n symmetric matrix with the attribute sparse.
%
%
%   Description:    An undirected graph is chosen uniformly at random from
%                   the set of all symmetric graphs with n nodes and m
%                   edges.
%
%   Reference:  P. Erdos, A. Renyi,
%               On Random Graphs,
%               Publ. Math. Debrecen, 6 1959, pp. 290-297.
%
%   Example: A = erdrey(100,10);

if nargin == 1
    m = ceil(n*log(n)/2);
end

nonzeros = ceil(0.5*n*(n-1)*rand(m,1));
v = zeros(n,1);
for count = 1:n
    v(count) = count*(count-1)/2;
end

I = zeros(m,1);
J = zeros(m,1);
S = ones(m,1);

for count = 1:m
    i = min(find(v >= nonzeros(count)));
    j = nonzeros(count) - (i-1)*(i-2)/2;
    I(count) = i;
    J(count) = j;
end

A = sign(sparse([I;J],[J;I],[S;S],n,n));

while nnz(A) ~= 2*m

    difference = m-nnz(A)/2;
    Inew = zeros(difference,1);
    Jnew = zeros(difference,1);
    for count = 1:difference
        index = ceil(0.5*n*(n-1)*rand);
        Inew(count) = min(find(v>=index));
        Jnew(count) = index - (Inew(count)-1)*(Inew(count)-2)/2;
    end
    I = cat(1,I,Inew);
    J = cat(1,J,Jnew);
    S = ones(length(I),1);
    A = sign(sparse([I;J],[J;I],[S;S],n,n));

end
```

A.4 geo.m

```
function A = geo(n,r,m,per,pnorm)

%GEO      Generate adjacency matrix for a geometric random graph.
%
%   Input   n: dimension of matrix (number of nodes in graph)
%           r: radius used to defined entries (edges). Defaults to the
%           square root of 1.44/n.
%           m: dimension of coordinate system. Defaults to 2.
%        per: periodicity of coordinate system. Periodic if per == 1,
%             not periodic if per == 0. Defaults to 0.
%      pnorm: norm to measure distance between nodes. Defaults to 2.
%
%   Output  A: n by n symmetric matrix with the attribute sparse
%
%
%   Description: nodes are placed randomly in the unit m-cube.
%                An edge is created if two nodes are within distance r.
%   Reference:   M. Penrose, Geometric Random Graphs,
%                Oxford Univeristy Press, 2003.
%
%   Example: A = geo(100,0.01,3,1,2);

if nargin <= 4
    pnorm = 2;
    if nargin <= 3
        per = 0;
        if nargin <= 2
            m = 2;
            if nargin == 1
                r = sqrt(1.44/n);
            end
        end
    end
end

coords = rand(n,m);

I = [];
J = [];

if per == 0

    for i = 2:n
        for j = 1:(i-1)
            diff = abs(coords(i,:) - coords(j,:));
            if norm(diff,pnorm)<=r
                J = cat(1,J,j);
                I = cat(1,I,i);
            end
        end
    end

end

if per == 1

    for i = 2:n
        for j = 1:(i-1)
            diff = min( abs( coords(i,:) - coords(j,:) ), abs( 1 - abs( coords(i,:) - coords(j,:) ) ) );
            if norm(diff,pnorm)<=r
                J = cat(1,J,j);
                I = cat(1,I,i);
            end
        end

    end

end

S = ones(length(I),1);
A = sparse([I;J],[J;I],[S;S],n,n);
```

A.5 gilbert.m

```
function A = gilbert(n,p);

%GILBERT     Generate adjacency matrix for a G(n,p) type random graph.
%
%   Input    n: dimension of matrix (number of nodes in graph).
%            p: probability that any two nodes are neighbours. Defaults to
%               log(n)/n.
%
%   Output   A: n by n symmetric matrix with the attribute sparse.
%
%
%   Description:    An undirected graph is created by considering pairs of
%                   nodes and connecting them with independent probability p.
%
%   Reference:      E.N. Gilbert,
%                   Random Graphs,
%                   Ann. Math. Statist.,30, (1959) pp. 1141-1144.
%
%   This code is a direct translation of Algorithm 1 in
%                   V. Batagelj, U. Brandes,
%                   Efficient generation of large random networks,
%                   Phys. Rev. E, 71 (2005).
%   This algorithm uses a geometric method to skip over potential edges,
%   and has optimal complexity.
%
%   Example: A = gilbert(100,0.1);

if nargin == 1
    p = log(n)/n;
end

v = zeros(n,1);        % Think of lower triangle of n-by-n array ordered
for k = 1:n,           % lexographically, row-wise. So v(k) is the biggest
    v(k) = k*(k-1)/2;  % index appearing in row k.
end

I = zeros(ceil(0.5*p*n^2),1);  % We don't know exact length
J = zeros(ceil(0.5*p*n^2),1);  % Expected length is 0.5*p*n(n-1)

count = 0;
w = 0;

w = w + 1 + floor(log(1-rand)/log(1-p));  %Lexographical index of next nonzero
while w < n*(n-1)/2;
   i = min(find(v >= w));    % Recover i and j from
   j = w - (i-1)*(i-2)/2;    % lexographical index w
   I(count+1) = i;
   J(count+1) = j;
   count = count + 1;
   w = w + 1 + floor(log(1-rand)/log(1-p)); %Lexographical index of next nonzero
end

Ifind = find(I>0);   % trim any left-over zeros
I = I(Ifind);        % from I and J
Jfind = find(J>0);
J = J(Jfind);

S = ones(length(I),1);
A = sparse([I;J],[J;I],[S;S],n,n);
```

A.6 kleinberg.m

```
function A = kleinberg(n,p,q,alpha)

%KLEINBERG      Generate adjacency matrix for a range dependent graph.
%
%   Input   n: dimension of matrix (number of nodes in lattice)
%           p: maximum distance of short-range connections. Defaults to 1.
%           q: number of random connections to add per node. Defaults to 1.
%           alpha: clustering exponent. Defaults to 2.
%
%   Output  A: n by n symmetric matrix with the attribute sparse
%
%
%   Description:    An alternative small world graph is created by
%                   taking a toroidal lattice that connects nodes within
%                   Manhattan distance p of each other and adding q long
%                   range short cuts to each node. The probability of
%                   a short cut between two nodes is inversely proportional
%                   to their Manhattan distance.
%
%   Reference:      J. Kleinberg
%                   Navigation in a small world
%                   Nature 406 (2000), p.845.
%
%   Example: A = kleinberg(100,2,3,1.5);

m = round(sqrt(n));
n = m^2;

if nargin <= 3
    alpha = 2;
    if nargin <= 2
        q = 1;
        if nargin ==1
            p = 1;
        end
    end
end

%First create the toroidal lattice

%Create A1, the diagonal repeating block

I1 = zeros(m*2*p,1);
J1 = zeros(m*2*p,1);
S1 = ones(m*2*p,1);
for i = 1:m
    I1((i-1)*2*p+1:i*2*p) = i*ones(2*p,1);
    J1((i-1)*2*p+1:i*2*p) = mod([i-p:i-1 i+1:i+p],m);
end
J1(find(J1==0)) = m;

%Create A2:A(p+1), the off diagonal blocks

I2 = zeros(m*p^2,1);
J2 = []; J3 = [];
S2 = ones(m*p^2,1);

for i=1:m
    I2((i-1)*p^2+1:i*p^2) = i*ones(p^2,1);
    for j=1:p
        Jnew = mod([i-(p-j):i+(p-j)],m);
        Jnew(find(Jnew==0)) = m;
        Jnew = Jnew + j*m;
        J3new = Jnew+m*(m-2*j);
        J2 = cat(2,J2,Jnew);
        J3 = cat(2,J3,J3new);
    end
end
J2 = J2'; J3 = J3';

%Construct block Toeplitz matrix using first m rows

Ifirst = cat(1,I1,I2,I2);
Jfirst = cat(1,J1,J2,J3);
Sfirst = cat(1,S1,S2,S2);

Irest = [];
Jrest = [];

for i = 1:m-1
    Irest = cat(1,Irest,i*m+Ifirst);
    Jrest = cat(1,Jrest,mod(Jfirst+i*m,n));
    Jrest(find(Jrest==0)) = n;
end

Srest = ones(length(Irest),1);

%Add shortcuts

pdist = [1:n].^(-alpha);
pdist = cumsum(pdist./sum(pdist));

Ihat = zeros(q*n,1);
Jhat = zeros(q*n,1);
Shat = ones(q*n,1);

for i = 1:n
    Ihat((i-1)*q+1:i*q) = i*ones(q,1);
    for j = 1:q
        r = rand;
        index = 1;
        while r >pdist(index)
            index = index+1;
        end
        r = index;
        rowdist = [ones(1,r) 0.5];
        rowdist = cumsum(rowdist./sum(rowdist));
        dist = rand;
        index = 1;
        while dist > rowdist(index)
            index = index + 1;
        end
        dist = index;
        Jhat((i-1)*q+j) = mod(i + ((-1)^floor(rand*2))*dist*m + ((-1)^floor(rand*2))*(r-dist),n)+1;
    end
end

A = sparse([Ifirst;Irest;Ihat;Jhat],[Jfirst;Jrest;Jhat;Ihat],[Sfirst;Srest;Shat;Shat],n,n);
A = sign(A);
```

A.7 lap.m

```
function L = lap(A,nl)

%LAP            Calculate the Laplacian of an adjacency matrix.
%
%   Input   A: n by n adjacency matrix (symmetric).
%          nl: flag for normalization.
%              nl = 0 means unnormalized, nl = 1 means normalized.
%              Defaults to nl = 1.
%
%   Output  L: n by n Laplacian of A.
%
%
%   Description:    Calculates the normalized or unnormalized Laplacian
%                   of a symmetric matrix.
%                   Unnormalized form is diag(sum(A)) - A.
%                   Normalized form has a row/column scaling applied.

if nargin == 1
    nl = 1;
end

L = sparse(diag(sum(A))) - A; % unnormalized

if nl == 1
    deg = max(sum(A),1);
    Dhf = sparse(diag(deg.^(-1/2)));
    L = Dhf*L*Dhf;            % normalized
end
```

A.8 lockandkey.m

```
function A = lockandkey(n,m,p)

%LOCKANDKEY Generate adjacency matrix for a lock & key random graph
%
%   Input   n: dimension of matrix
%           m: number of domains (lock/key pairs). Defaults to the smallest
%           integer bigger than n*log(n).
%           p: probability that a node is assigned each domain type.
%           Defaults to 1/n.
%
%   Output  A: n by n symmetric matrix with attribute sparse
%
%   Description:    Locks and keys are distributed randomly amongst nodes.
%                   Interactions (edges) occur between nodes that share
%                   lock/key pairs. Although a node may be assigned both a
%                   particular lock and the corresponding key, self links
%                   are disallowed.
%
%   Reference:  J.L. Morrison, R. Breitling, D.J. Higham, D.R. Gilbert,
%               A Lock-and-key model for protein-protein interactions,
%               Bioinformatics, 22 (2006), pp. 2012-2019.
%
%   Example: A = lockandkey(100,5,0.4);

if nargin <= 2
```

```
    p = 1/n;
    if nargin == 1
        m = ceil(n*log(n));
    end
end

I = [];
J = [];

for domains = 1:m

    locks = find(rand(n,1)<p);
    keys = find(rand(n,1)<p);

    if length(locks)>=1 && length(keys)>=1
        khat = keys*ones(1,length(locks));
        I = cat(1,I,sort(khat(:)));

        lhat = repmat(locks,length(keys),1);
        J = cat(1,J,lhat);
    end

end

S = ones(length(I),1);

A = sign(sparse([I;J],[J;I],[S;S],n,n));
A = A - diag(diag(A));
```

A.9 mht.m

```
function M = mht(A,i)

%MHT        Mean hitting times
%
%   Input   A: n by n adjacency matrix
%           i: integer representing starting state.
%              Defaults to i = 1.
%
%   Output  M: n-1 by n-1 matrix for the mean hitting time system
%              corresponding to a random walk on the graph A starting
%              at node i.
%
%   Description:    Computes the mean hitting time matrix, M, given by
%                   forming I - D^(-1)*A, where D=diag(sum(A)) is the
%                   diagonal degree matrix (required to be nonsingular) and
%                   then removing the ith row and column.
%                   Solving M*x = ones(n-1,1) then gives the hitting time
%                   vector, x.
%
%   Example: M = mht(A,2)   mean hitting time matrix for state 2

if nargin == 1
    i = 1;
end

M = sparse(diag(1./sum(A))*A);
```

```
M(i,:) = [];
M(:,i) = [];

m = length(M);
M = speye(m,m) - M;
```

A.10 pagerank.m

```
function P = pagerank(A,d)

%PAGERANK   Calculate the PageRank matrix corresponding to a given
%           adjacency matrix.
%
%   Input   A: n by n adjacency matrix.
%           d: multiplicative constant. Defaults to 0.15.
%
%   Output  P: n by n PageRank matrix.
%
%   Description:    Calculates the matrix P, defined as
%                   P = I - d*(A'/max(deg(A),1)).
%
%   Reference:  L. Page, S. Brin,
%               The anatomy of a large-scale hypertextual web search engine
%               Proceedings of the Seventh International Web Conference,
%               (1998).
%
%   Example: P = pagerank(A,0.3);

if nargin == 1
    d = 0.85;
end

n = length(A);

if any(find(A-A'))

    P = sparse(speye(n) - d*(A'*diag(1./max(sum(A,2),1))));

else

    P = sparse(speye(n) - d*(A'*diag(1./max(sum(A),1))));

end
```

A.11 pathlength.m

```
function Plength = pathlength(A)

% PATHLENGTH Calculate minimum pathlengths for a given adjacency
%           matrix.
%
%   Input   A: n by n adjacency matrix (symmetric).
```

```
%
%   Output  Plength: n by n matrix of pathlengths. Element in
%                     position (i,j) is pathlength from node i to node j.
%                     If no path exists, inf is returned.
%
%   Description:    Powers up the adjacency matrix until either there are
%                   no elements equal to zero or the (n-1)st power has been
%                   reached. Records the first power at which (i,j) element
%                   became nonzero.
%
%   Example: Plength = pathlength(A);

Anew = A;
n = length(A);
power = 1;
Plength = sign(A + eye(n,n)); % record all paths of length one, including diagonal

while any(any(Anew==0)) && power <= (n-1)
    power = power + 1;
    Anew = Anew*A;
    Plength = Plength + ( (Plength == 0) & (Anew > 0) )*power;
end

pzero = find(Plength == 0);
Plength(pzero) = inf;                  % reset zeros to inf

Plength = Plength - diag(diag(Plength));     % reset diagonal to zero
```

A.12 pref.m

```
function A = pref(n,d)

%PREF       Generate adjacency matrix for a scale free random graph.
%
%   Input   n: dimension of matrix (number of nodes in graph).
%           d: minimum row sum (minimum node degree). Defaults to 2.
%
%   Output  A: n by n symmetric matrix with the attribute sparse
%
%
%   Description:    Nodes are added successively. For each node, d edges
%                   are generated. The endpoints are selected from the
%                   nodes whose edges have already been created, with bias
%                   towards high degree nodes. This is a MATLAB
%                   implementation of Algorithm 5 in [2].
%
%   References: [1] A.L. Barabasi, R. Albert,
%                   Emergence of scaling in random networks,
%                   Science Vol. 286, 15 (1999).
%
%               [2] V. Batagelj, U. Brandes,
%                   Efficient generation of large random networks,
%                   Phys. Rev. E, 71 (2005).
%
%   Example: A = pref(100,2);

if nargin == 1
    d = 2;
```

```
end

I = [];
J = [];

for v = 1:n
    for i = 1:d
        M(2*((v-1)*d+i)-1) = v;
        I = cat(1,I,v);
        r = ceil(rand*(2*((v-1)*d+i)-1));
        M(2*((v-1)*d+i))=M(r);
        J = cat(1,J,M(r));
    end
end

S = ones(length(I),1);
A = sign(sparse([I;J],[J;I],[S;S],n,n));
```

A.13 renga.m

```
function A = renga(n,lambda,alpha)

%RENGA              Generate adjacency matrix for a range dependent random
%                   graph.
%
%   Input        n: dimension of matrix (number of nodes in graph).
%           lambda: fixed base for the edge probability. Defaults to 0.9.
%            alpha: multiplicative constant for the edge probability.
%                   Defaults to 1.
%
%   Output       A: n by n symmetric matrix with the attribute sparse
%
%   Description:    nodes are considered to lie at unit intervals on a
%                   line. The probability of connecting two nodes is given
%                   by the function p = alpha*lambda^(d-1) where d is the
%                   distance between the nodes.
%
%   Reference:  P. Grindrod,
%               Range-dependent random graphs and their application to
%               modelling large small-world proteome datasets,
%               Phys. Rev E. 66, 066702 (2002).
%
%   Example: A = renga(100,0.9,0.3);

if nargin <= 2
    alpha = 1;
    if nargin == 1
        lambda = 0.9;
    end
end

I = zeros(ceil(1.25*alpha*n/(1-lambda)),1);
J = zeros(ceil(1.25*alpha*n/(1-lambda)),1);

count = 1;

for i = 2:n
    for j = 1:i-1
```

```
        if rand <= alpha*lambda^abs(i-j-1)
            I(count) = i;
            J(count) = j;

            count = count+1;

        end
    end
end

I(find(I==0)) = [];
J(find(J==0)) = [];

A = sparse([I;J],[J;I],ones(2*length(I),1));
```

A.14 rewire.m

```
function R = rewire(A,p)

%REWIRE      Watts-Strogatz rewiring
%
%   Input   A: n by n adjacency matrix
%           p: probability of rewiring a given edge. Defaults to log(n)/n.
%
%   Output  R: rewired n by n adjacency matrix
%
%   Description:    Takes an adjacency matrix A and redirects each edge with
%                   probability p. Rewiring to an existing neighbour (i.e.
%                   overlapping edges) is not allowed. Rewiring is done in a
%                   symmetric fashion.
%
%   Reference:  D.J. Watts, S. H. Strogatz,
%               Collective Dynamics of Small World Networks,
%               Nature 393 (1998), pp. 440-442.
%
%   Example: R = rewire(A,0.1);

n = length(A);
[I,J] = find(tril(A));
edges = length(I);

for i = 1:edges
    if rand <= p
        newneighbour = ceil(rand*n);
        while A(I(i),newneighbour) ~= 0 | newneighbour == I(i)
            newneighbour = ceil(rand*n);
        end
        A(I(i),J(i)) = 0;
        A(J(i),I(i)) = 0;
        A(I(i),newneighbour) = 1;
        A(newneighbour,I(i)) = 1;
    end
end

R = A;
```

A.15 short.m

```
function S = short(A,p)

%SHORT      Randomly add entries (shortcuts) to a matrix
%
%   Input   A: n by n adjacency matrix
%           p: probability that an entry is added to a given row
%
%   Output  S: n by n adjacency matrix with the attribute sparse.
%
%   Description:    A symmetric matrix of shortcuts is created which has
%                   an entry in each row with independent probability p.
%                   This is added to the matrix A.
%
%   Example: S = short(A,0.3);

n = length(A);

if nargin == 1
    p = log(n)/n;
end

Ihat = find(rand(n,1)<=p);
Jhat = ceil(n*rand(size(Ihat)));
Ehat = ones(size(Ihat));

self = find(Ihat==Jhat);
Ihat(self) = [];
Jhat(self) = [];
Ehat(self) = [];


[I,J,E] = find(A);

S = sparse([I;Ihat;Jhat],[J;Jhat;Ihat],[E;Ehat;Ehat],n,n);
S = sign(S);
```

A.16 smallw.m

```
function A = smallw(n,k,p)

%SMALLW     Generate adjacency matrix for a small world network.
%
%   Input   n: dimension of matrix (number of nodes in graph).
%           k: number of nearest-neighbours to connect. Defaults to 1.
%           p: probability of adding a shortcut in a given row. Defaults to
%           0.1.
%
%   Output  A: n by n symmetric matrix with the attribute sparse.
%
%   Description:    Shortcuts are added to a kth nearest neighbour ring
%                   network with n nodes by calling the utility function
%                   short.m.
%
%   Reference:  D.J. Watts, S. H. Strogatz,
%               Collective Dynamics of Small World Networks,
%               Nature 393 (1998), pp. 440-442.
```

```
%
%   Example:    A = smallw(100,1,0.2);

if nargin <= 2
    p = 0.1;
    if nargin == 1
        k = 2;
    end
end

twok = 2*k;

I = zeros(2*k*n,1);
J = zeros(2*k*n,1);
S = zeros(2*k*n,1);

for count = 1:n

    I( (count-1)*twok+1 : count*twok ) = count.*ones(twok,1);
    J( (count-1)*twok+1 : count*twok ) = mod([count:count+k-1 n-k+count-1:n+count-2],n)+1;
    S( (count-1)*twok+1 : count*twok ) = ones(twok,1);

end

A = sparse(I,J,S,n,n);
A = short(A,p);
```

A.17 sticky.m

```
function A = sticky(n,gamma)

%STICKY     Stickiness model random graph
%
%   Input   n:  dimension of matrix.
%           gamma: exponent in scale-free target degree distribution. Probability
%                  of a node having degree i is proportional to i^(-gamma).
%                  Defaults to 2.5.
%
%            *Exception* A = sticky(deg), where deg is a 1D array with length(deg) > 1.
%             Here deg has integer entries between 0 and n and deg(i) is the degree
%             for node i in the target network.
%
%   Output  A:  n by n symmetric matrix with the attribute sparse.
%
%   Description:    A graph is chosen uniformly from a class of random graphs
%                   whose expected degrees match the given target distribution.
%
%   Reference:      N. Przulj, D.J. Higham,
%                   Modelling protein-protein interaction networks via a
%                   stickiness index
%                   J. Royal Society Interface, 3 (2006), pp 711-716.
%
%  Example: A = sticky(100,2);                  100 nodes, scale-free with exponent -2
%           A = sticky(100);                    100 nodes, scale-free with default exponent -2.5
%           A = sticky(sum(gilbert(100,0.02))); 100 nodes, expected degrees from gilbert(100,0.02))

if nargin == 1 && length(n) > 1
        % user has specified target degrees
        deg = n(:);
        n = length(deg);
else
        % scale-free case
        if nargin == 1
            % user wants scale-free with default gamma
            gamma = 2.5;
        end
        %% compute target degrees from a scale-free distribution with exponent -gamma %%
        d = cumsum([1:n].^(-gamma));
        d = d'/d(end);
        d = [0;d(1:end-1)];
        deg = zeros(n,1);
        for i = 1:n
            deg(i) = max(find(rand>d));  %degree of node i
        end
end

%Now compute sticky graph using these degrees
I = [];
J = [];
S = [];

root = sqrt(sum(deg));

for i = 1:n
    for j = 1:i-1
        if rand < deg(i)*deg(j)/(root^2)
            I = cat(1,I,i);
            J = cat(1,J,j);
            S = cat(1,S,1);
        end
    end
end

diagonal = find(I==J);
I(diagonal) = [];
J(diagonal) = [];
S(diagonal) = [];

A = sparse([I;J],[J;I],[S;S],n,n);
```

A.18 unisample.m

```
function U = unisample(A,p)

%UNISAMPLE  subsampling a graph
%
%   Input   A: n by n adjacency matrix
%           p: probability of retaining each node. Defaults to 0.5.
%
%   Output  U: adjacency matrix with the attribute sparse.
%              Dimension of U cannot be predicted.
%
%   Description:    Nodes in a graph are retained with independent
%                   probability p. Nodes that are "discarded" are removed
%                   from the adjacency matrix.
%
```

```
%   Example:    U = unisample(A,0.25);

if nargin == 1
    p = 0.5;
end

U=A;
remove = find(rand(length(A),1) > p); % remove these rows/columns
U(remove,:) = [];
U(:,remove) = [];
```

Appendix B

CONTEST Testing

Here we present the bulk of the results for the testing procedures described in Section 4.4. The `amd` run times for each random graph model are plotted and we list the iteration counts, relative residuals and run-times for a variety of numerical schemes on test matrices drawn from both symmetric and unsymmetric instances of the random graph models. We note that these tests were performed on an earlier version of the codes in CONTEST and acknowledge that the current version of the codes may produce slightly different results.

B.1 `amd` Tests

We display the figures generated by the `amd` testing described in Section 4.4 for the remaining seven models implemented in CONTEST. A best-fit line has been added to the plots and, as before, the run times are typically below $O(|L|)$. There are some outlying data points in all figures, but most notably in Figures B.1, B.4 and B.6 where they form straight lines.

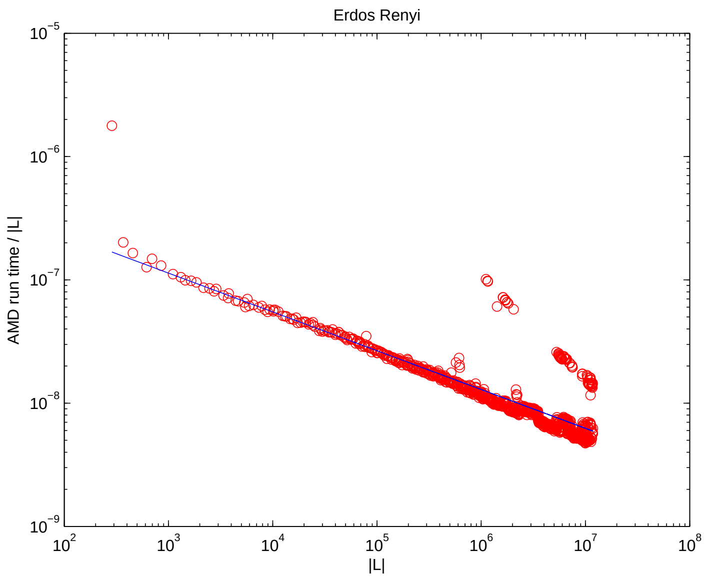


Figure B.1: `amd` run times for Erdös-Rényi model.

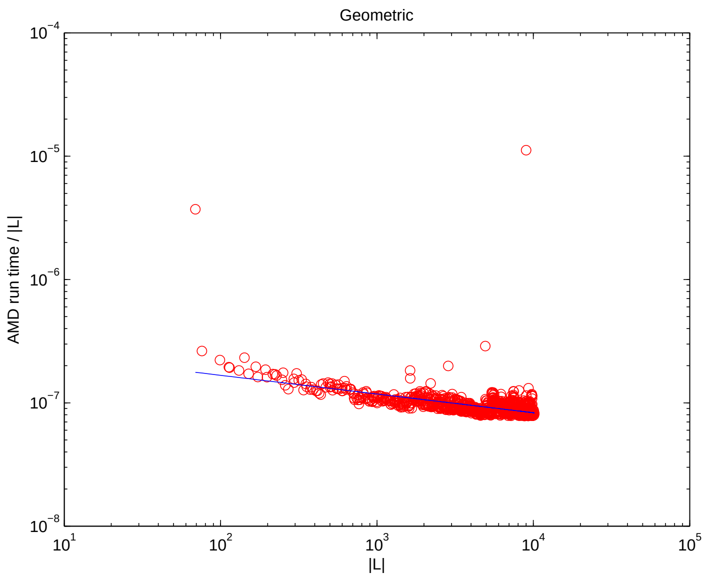


Figure B.2: `amd` run times for geometric model.

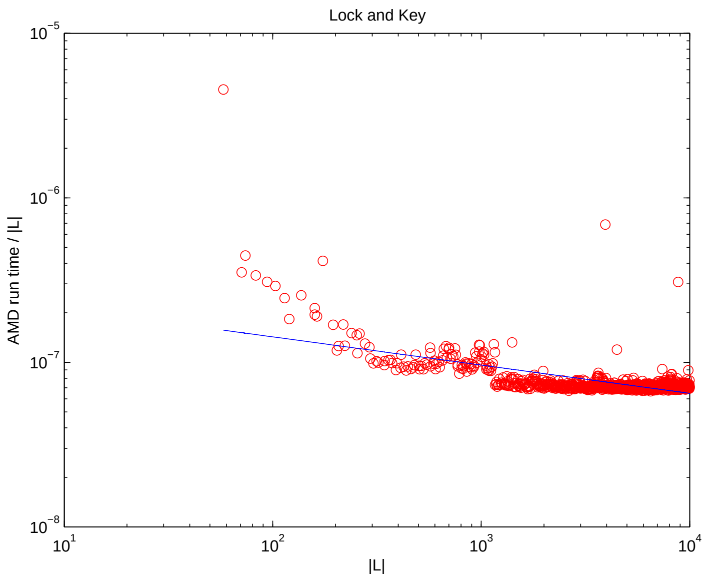


Figure B.3: `amd` run times for lock and key model.

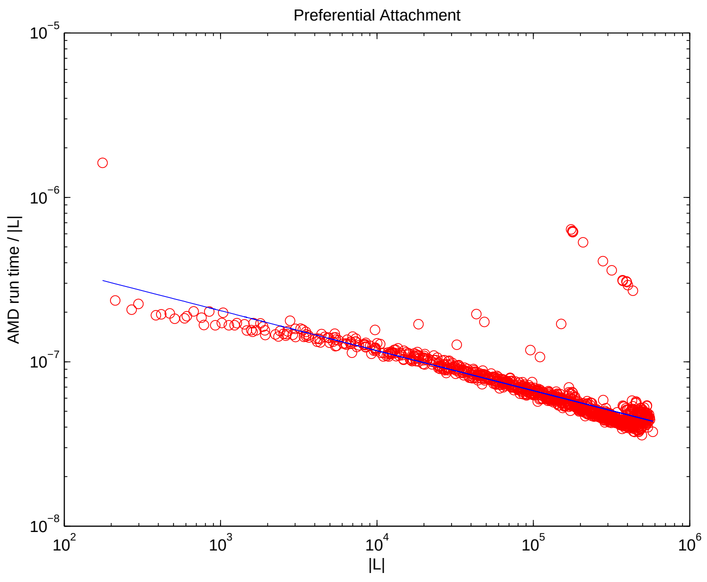


Figure B.4: `amd` run times for preferential attachment model.

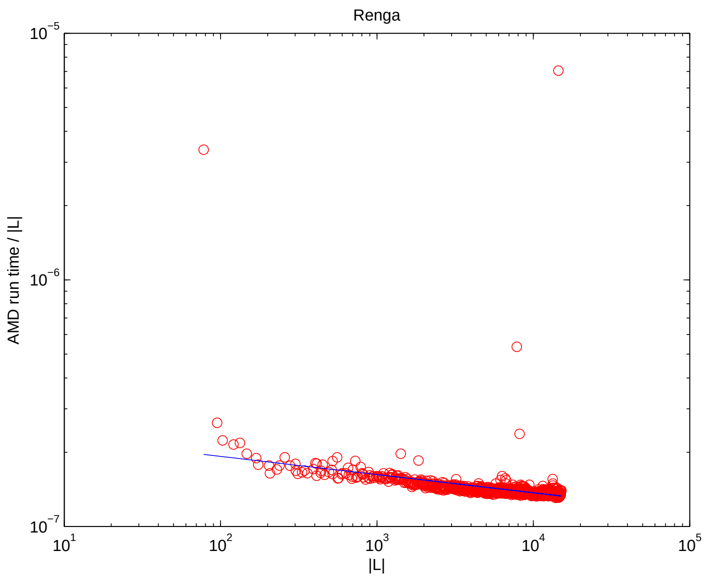


Figure B.5: `amd` run times for renga model.

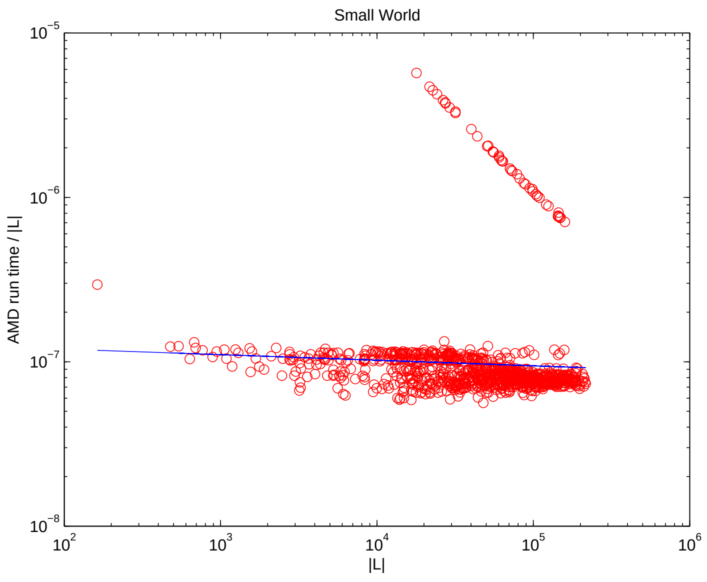


Figure B.6: `amd` run times for small world model.

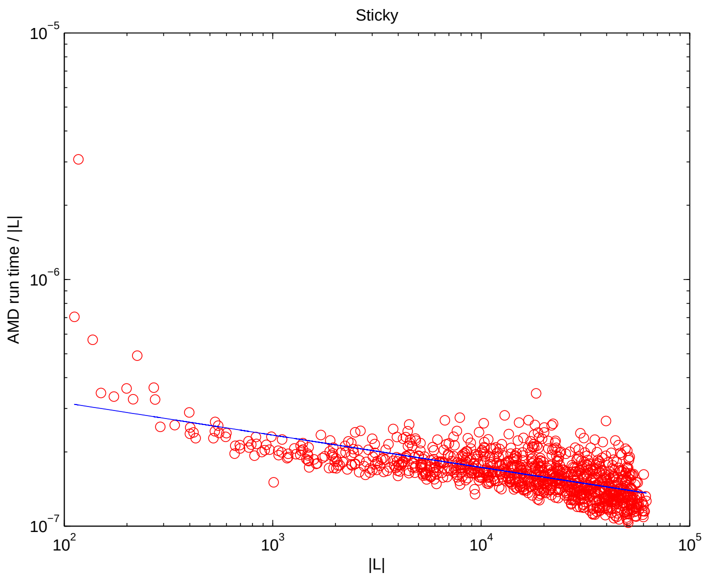


Figure B.7: `amd` run times for stickiness model.

B.2 Linear System Tests

We present here tables of running times, iteration counts and relative residuals for symmetric and unsymmetric matrices generated by the remaining eight models in CONTEST applied to the testing described in Section 4.4. Footnotes indicate what percentage of cases ended in a given state. We describe the meanings of each state for reference.

State 0 : the process coverged to the specified tolerance within the maximum number of iterations.

State 1 : the maximum number of iterations was reached without convergence to a solution.

State 2 : the preconditioner was ill-conditioned.

State 3 : the process stagnated, i.e. successive iterations were the same.

State 4 : a scalar quantity in the computation became too large or too small for MATLAB to process.

We observe that for several of the models, using the Cholesky preconditioner resulted in all cases finishing in state 1 when running `symmlq` or `minres`. We also emphasise that this testing does not comprise the main part of the work in Chapter 4 and that these tests are designed to show that the models implemented in CONTEST can produce adjacency matrices which can be seen as challenging test problems for general linear system solvers.

B.2.1 Erdös Rényi Symmetric

Matrices generated by `erdrey(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$7.3985\,e-2$ $(1.2\,e-3)$	$1.2112\,e-1$ $(5.1\,e-4)$	$5.2146\,e-1$ $(2.9\,e-3)$
qmr	$7.9524\,e-2$ $(6.4\,e-4)$	$2.3985\,e-1$ $(4.4\,e-4)$	$2.2148\,e-1$ $(8.4\,e-4)$
symmlq	$5.3171\,e-2$ $(4.9\,e-4)$	$1.3050\,e-1$ $(5.5\,e-4)$	$1.0508\,e+0$ $(4.2\,e-3)$ [6]
lsqr	$1.9053\,e-1$ $(7.3\,e-4)$	$3.5934\,e-1$ $(7.5\,e-4)$	$3.7559\,e-1$ $(1.1\,e-3)$
minres	$5.2023\,e-2$ $(6.2\,e-4)$	$1.4226\,e-1$ $(5.1\,e-4)$	$1.2865\,e+0$ $(6.3\,e-3)$ [6]
cgs	$7.1681\,e-2$ $(6.9\,e-3)$	$1.2984\,e-1$ $(4.0\,e-4)$	$9.0465\,e-2$ $(5.3\,e-4)$
gmres	$1.1054\,e-1$ $(1.5\,e-3)$	$1.7561\,e-1$ $(4.2\,e-4)$	$1.5740\,e-1$ $(5.2\,e-4)$
bicg	$7.1005\,e-2$ $(4.3\,e-4)$	$2.1146\,e-1$ $(4.2\,e-4)$	$2.0481\,e-1$ $(8.1\,e-4)$
bicgstab	$6.0545\,e-2$ $(7.4\,e-4)$	$1.0107\,e-1$ $(3.8\,e-4)$	$9.5148\,e-2$ $(5.2\,e-4)$

Table B.1: Mean running times and standard errors for `erdrey`.

B.2.2 Erdös Rényi Unsymmetric

Matrices generated by `erdrey(10000)`.

[6] All cases finished in state 1
[7] 32% of cases finished in state 0, 66% in state 1 and 2% in state 4
[8] All cases finished in state 1

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$1.1010\,e+1$ $(1.0e-4)$	$5\,e+0$ (0)	$4.2290\,e+1$ $(9.1\,e-2)$
qmr	$8.0300\,e+0$ $(1.7e-2)$	$4\,e+0$ (0)	$7.0200\,e+0$ $(1.4\,e-2)$
symmlq	$10\,e+0$ (0)	$4\,e+0$ (0)	$14\,e+0$ (0) [6]
lsqr	$1.0030\,e+1$ $(1.7e-2)$	$5\,e+0$ (0)	$9.0300\,e+0$ $(1.7\,e-2)$
minres	$10\,e+0$ (0)	$4\,e+0$ (0)	$3.2680\,e+1$ $(3.1\,e-1)$ [6]
cgs	$8.9375\,e+0$ $(8.8\,e-1)$	$3\,e+0$ (0)	$4\,e+0$ (0)
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$8\,e+0$ (0)	$4\,e+0$ (0)	$7.0200\,e+0$ $(1.4\,e-2)$
bicgstab	$4.4950\,e+0$ $(3.7\,e-2)$	$2\,e+0$ (0)	$3.500\,e+0$ (0)

Table B.2: Mean iteration counts and standard errors for `erdrey`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$7.4049\,e-11$ $(1.0\,e-12)$	$8.6882\,e-13$ $(4.6\,e-15)$	$9.4770\,e-11$ $(2.8\,e-13)$
qmr	$3.6701\,e-11$ $(1.1\,e-12)$	$1.8703\,e-11$ $(7.0\,e-14)$	$1.4977\,e-11$ $(1.5\,e-12)$
symmlq	$6.4903\,e-11$ $(7.9\,e-13)$	$8.6838\,e-13$ $(4.5\,e-15)$	$3.3252\,e-6$ $(2.9\,e-8)$ [6]
lsqr	$6.6793\,e-11$ $(1.3\,e-12)$	$8.1383\,e-12$ $(3.6\,e-13)$	$4.0550\,e-11$ $(1.4\,e-12)$
minres	$6.3164\,e-11$ $(7.7\,e-13)$	$8.6829\,e-13$ $(4.5\,e-15)$	$1.6429\,e-7$ $(1.1\,e-9)$ [6]
cgs	$3.3615\,e-11$ $(4.6\,e-12)$	$6.8250\,e-16$ $(4.9\,e-18)$	$5.6089\,e-12$ $(5.4\,e-13)$
gmres	$3.1022\,e-11$ $(1.1\,e-13)$	$1.7518\,e-11$ $(5.7\,e-14)$	$6.4137\,e-12$ $(2.6\,e-14)$
bicg	$3.4861\,e-11$ $(2.4\,e-13)$	$1.8713\,e-11$ $(7.0\,e-14)$	$1.5054\,e-11$ $(1.5\,e-12)$
bicgstab	$4.3697\,e-11$ $(3.0\,e-12)$	$3.4251\,e-11$ $(9.5\,e-14)$	$1.1100\,e-11$ $(4.0\,e-13)$

Table B.3: Mean relative residuals and standard errors for `erdrey`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$1.1326\,\mathrm{e}-1$ $(6.0\,\mathrm{e}-3)$	$1.4865\,\mathrm{e}-1$ $(2.2\,\mathrm{e}-4)$	$6.1355\,\mathrm{e}-1$ $(9.3\,\mathrm{e}-4)$
qmr	$1.1928\,\mathrm{e}-1$ $(4.0\,\mathrm{e}-3)$	$3.5846\,\mathrm{e}-1$ $(4.1\,\mathrm{e}-4)$	$3.3815\,\mathrm{e}-1$ $(5.8\,\mathrm{e}-4)$
symmlq	$7.7118\,\mathrm{e}-2$ $(6.5\,\mathrm{e}-4)$	$1.5973\,\mathrm{e}-1$ $(3.6\,\mathrm{e}-4)$	N/A[8]
lsqr	$3.3258\,\mathrm{e}-1$ $(5.4\,\mathrm{e}-4)$	$5.4199\,\mathrm{e}-1$ $(4.4\,\mathrm{e}-3)$	$5.1392\,\mathrm{e}-1$ $(1.6\,\mathrm{e}-3)$
minres	$7.3983\,\mathrm{e}-2$ $(3.9\,\mathrm{e}-4)$	$1.7285\,\mathrm{e}-1$ $(4.0\,\mathrm{e}-4)$	N/A[8]
cgs	$2.7740\,\mathrm{e}-1$ $(4.9\,\mathrm{e}-2)$[7]	$1.3264\,\mathrm{e}-1$ $(2.2\,\mathrm{e}-4)$	$1.4312\,\mathrm{e}-1$ $(2.2\,\mathrm{e}-4)$
gmres	$1.2925\,\mathrm{e}-1$ $(2.4\,\mathrm{e}-3)$	$2.2439\,\mathrm{e}-1$ $(3.4\,\mathrm{e}-4)$	$2.0660\,\mathrm{e}-1$ $(2.9\,\mathrm{e}-4)$
bicg	$1.0927\,\mathrm{e}-1$ $(5.0\,\mathrm{e}-4)$	$3.207\,\mathrm{e}-1$ $(3.4\,\mathrm{e}-4)$	$3.2287\,\mathrm{e}-1$ $(6.7\,\mathrm{e}-4)$
bicgstab	$9.8519\,\mathrm{e}-2$ $(7.6\,\mathrm{e}-4)$	$1.4883\,\mathrm{e}-1$ $(2.3\,\mathrm{e}-4)$	$1.5133\,\mathrm{e}-1$ $(2.5\,\mathrm{e}-4)$

Table B.4: Mean running times and standard errors for unsymmetric `erdrey`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$9\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$3.2110\,\mathrm{e}+1$ $(3.1\,\mathrm{e}-2)$
qmr	$7.0100\,\mathrm{e}+0$ $(1.0\,\mathrm{e}-4)$	$4\,\mathrm{e}+0$ (0)	$7.0100\,\mathrm{e}+0$ $(1.0\,\mathrm{e}-2)$
symmlq	$8\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[8]
lsqr	$9\,\mathrm{e}+0$ (0)	$4.6400\,\mathrm{e}+0$ $(4.8\,\mathrm{e}-2)$	$7.0600\,\mathrm{e}+0$ $(2.4\,\mathrm{e}-2)$
minres	$8\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	N/A[8]
cgs	$1.7438\,\mathrm{e}+1$ $(3.2\,\mathrm{e}+0)$[7]	$2\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)
gmres	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)
bicg	$7.0800\,\mathrm{e}+0$ $(3.1\,\mathrm{e}-2)$	$4\,\mathrm{e}+0$ (0)	$7.0200\,\mathrm{e}+0$ $(1.4\,\mathrm{e}-2)$
bicgstab	$4.0850\,\mathrm{e}+0$ $(3.2\,\mathrm{e}-2)$	$2\,\mathrm{e}+0$ (0)	$3.5000\,\mathrm{e}+0$ (0)

Table B.5: Mean iteration counts and standard errors for unsymmetric `erdrey`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	8.0513 e − 11 (4.2 e − 13)	5.9831 e − 11 (1.5 e − 13)	9.1233 e − 11 (4.2 e − 13)
qmr	6.9670 e − 11 (6.8 e − 13)	2.8533 e − 12 (5.9 e − 15)	6.3255 e − 12 (1.1 e − 12)
symmlq	7.0229 e − 11 (3.7 e − 13)	5.9540 e − 11 (1.5 e − 13)	N/A[8]
lsqr	2.0231 e − 11 (6.1 e − 14)	3.5707 e − 11 (4.8 e − 12)	6.3266 e − 11 (1.5 e − 12)
minres	6.9613 e − 11 (3.6 e − 13)	5.9540 e − 11 (1.5 e − 13)	N/A[8]
cgs	4.4752 e − 11 (4.8 e − 12)[7]	4.3145 e − 11 (8.1 e − 14)	1.4285 e − 12 (3.2 e − 14)
gmres	6.4559 e − 11 (1.1 e − 13)	2.7688 e − 12 (5.6 e − 15)	1.4240 e − 12 (4.3 e − 15)
bicg	6.6245 e − 11 (1.6 e − 12)	2.8538 e − 12 (5.9 e − 15)	6.8636 e − 12 (1.3 e − 12)
bicgstab	2.0113 e − 11 (1.8 e − 12)	9.8870 e − 12 (2.4 e − 14)	2.1076 e − 12 (2.4 e − 14)

Table B.6: Mean relative residuals and standard errors for unsymmetric `erdrey`.

B.2.3 Gilbert Symmetric

Matrices generated by `gilbert(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	7.4577 e − 2 (5.3 e − 4)	1.2325 e − 1 (2.4 e − 4)	5.3552 e − 1 (2.9 e − 3)
qmr	8.0914 e − 2 (4.6 e − 4)	2.4058 e − 1 (2.1 e − 4)	2.2521 e − 1 (8.0 e − 4)
symmlq	5.3318 e − 2 (3.3 e − 4)	1.3225 e − 1 (2.0 e − 4)	N/A[10]
lsqr	1.9258 e − 1 (5.8 e − 4)	3.6162 e − 1 (4.1 e − 4)	3.8004 e − 1 (1.0 e − 5)
minres	5.2439 e − 2 (3.2 e − 4)	1.4335 e − 1 (2.0 e − 4)	N/A[10]
cgs	1.6591 e − 1 (4.0 e − 2)[9]	1.3110 e − 1 (1.8 e − 4)	9.3332 e − 2 (4.3 e − 4)
gmres	1.1007 e − 1 (2.6 e − 4)	1.7652 e − 1 (1.8 e − 4)	1.6073 e − 1 (3.7 e − 4)
bicg	7.1842 e − 2 (4.3 e − 4)	2.1292 e − 1 (2.2 e − 4)	2.1079 e − 1 (9.1 e − 4)
bicgstab	6.2060 e − 2 (6.5 e − 4)	1.0217 e − 1 (1.6 e − 4)	9.7965 e − 2 (4.6 e − 4)

Table B.7: Mean running times and standard errors for `gilbert`.

[9] 24% of cases finished in state 0, 73% in state 1 and 3% in state 4

[10] All cases finished in state 1

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$11\,\mathrm{e}+0$ (0)	$5\,\mathrm{e}+0$ (0)	$4.2310\,\mathrm{e}+1$ $(9.4\,\mathrm{e}-2)$
qmr	$8.0900\,\mathrm{e}+0$ $(2.9\,\mathrm{e}-2)$	$4\,\mathrm{e}+0$ (0)	$7.0600\,\mathrm{e}+0$ $(2.4\,\mathrm{e}-2)$
symmlq	$10\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	N/A[10]
lsqr	$1.0020\,\mathrm{e}+1$ $(1.4\,\mathrm{e}-2)$	$5\,\mathrm{e}+0$ (0)	$9.0300\,\mathrm{e}+0$ $(1.7\,\mathrm{e}-2)$
minres	$10\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	N/A[10]
cgs	$1.8208\,\mathrm{e}+1$ $(4.6\,\mathrm{e}+0)$[9]	$3\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)
gmres	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)
bicg	$8.0100\,\mathrm{e}+0$ $(1.0\,\mathrm{e}-4)$	$4\,\mathrm{e}+0$ (0)	$7.0700\,\mathrm{e}+0$ $(2.6\,\mathrm{e}-2)$
bicgstab	$4.5500\,\mathrm{e}+0$ $(4.7\,\mathrm{e}-2)$	$2\,\mathrm{e}+0$ (0)	$3.500\,\mathrm{e}+0$ (0)

Table B.8: Mean iteration counts and standard errors for `gilbert`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$7.4024\,\mathrm{e}-11$ $(9.8\,\mathrm{e}-13)$	$8.6697\,\mathrm{e}-13$ $(4.9\,\mathrm{e}-15)$	$9.5492\,\mathrm{e}-11$ $(3.0\,\mathrm{e}-13)$
qmr	$3.7143\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-12)$	$1.8632\,\mathrm{e}-11$ $(7.7\,\mathrm{e}-14)$	$1.1500\,\mathrm{e}-11$ $(1.2\,\mathrm{e}-12)$
symmlq	$6.4040\,\mathrm{e}-11$ $(8.6\,\mathrm{e}-13)$	$8.6656\,\mathrm{e}-13$ $(4.9\,\mathrm{e}-15)$	N/A[10]
lsqr	$6.7935\,\mathrm{e}-11$ $(1.4\,\mathrm{e}-12)$	$8.4239\,\mathrm{e}-12$ $(5.8\,\mathrm{e}-13)$	$4.0903\,\mathrm{e}-11$ $(1.5\,\mathrm{e}-12)$
minres	$6.2327\,\mathrm{e}-11$ $(8.3\,\mathrm{e}-13)$	$8.6648\,\mathrm{e}-13$ $(4.9\,\mathrm{e}-15)$	N/A[10]
cgs	$5.2562\,\mathrm{e}-11$ $(5.8\,\mathrm{e}-12)$[9]	$6.7930\,\mathrm{e}-16$ $(3.5\,\mathrm{e}-18)$	$4.7020\,\mathrm{e}-12$ $(3.1\,\mathrm{e}-13)$
gmres	$3.1000\,\mathrm{e}-11$ $(1.4\,\mathrm{e}-13)$	$1.7439\,\mathrm{e}-11$ $(5.9\,\mathrm{e}-14)$	$6.3347\,\mathrm{e}-12$ $(2.4\,\mathrm{e}-14)$
bicg	$3.5904\,\mathrm{e}-11$ $(7.1\,\mathrm{e}-13)$	$1.8642\,\mathrm{e}-11$ $(7.7\,\mathrm{e}-14)$	$1.0752\,\mathrm{e}-11$ $(9.4\,\mathrm{e}-13)$
bicgstab	$3.8618\,\mathrm{e}-11$ $(2.7\,\mathrm{e}-12)$	$3.4163\,\mathrm{e}-11$ $(1.1\,\mathrm{e}-13)$	$1.0373\,\mathrm{e}-11$ $(2.2\,\mathrm{e}-13)$

Table B.9: Mean relative residuals and standard errors for `gilbert`.

B.2.4 Gilbert Unsymmetric

Matrices generated by `gilbert(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$1.0683\,e-1$ $(9.1\,e-5)$	$1.4764\,e-1$ $(1.1\,e-4)$	$6.1187\,e-1$ $(9.4\,e-4)$
qmr	$1.1504\,e-1$ $(1.7\,e-4)$	$3.5680\,e-1$ $(2.4\,e-4)$	$3.3692\,e-1$ $(6.6\,e-4)$
symmlq	$7.5867\,e-2$ $(6.4\,e-5)$	$1.5890\,e-1$ $(1.2\,e-4)$	N/A[12]
lsqr	$3.2993\,e-1$ $(2.2\,e-4)$	$5.3909\,e-1$ $(4.3\,e-3)$	$5.1542\,e-1$ $(2.0\,e-3)$
minres	$7.3323\,e-2$ $(5.7\,e-5)$	$1.7122\,e-1$ $(1.2\,e-4)$	N/A[12]
cgs	$1.9742\,e-1$ $(3.8\,e-2)$[11]	$1.3185\,e-1$ $(1.3\,e-4)$	$1.4264\,e-1$ $(1.0\,e-4)$
gmres	$1.2624\,e-1$ $(9.6\,e-5)$	$2.2336\,e-1$ $(1.6\,e-4)$	$2.0545\,e-1$ $(1.6\,e-4)$
bicg	$1.0849\,e-1$ $(4.7\,e-4)$	$3.2006\,e-1$ $(2.3\,e-4)$	$3.2172\,e-1$ $(6.4\,e-4)$
bicgstab	$9.8232\,e-2$ $(5.6\,e-4)$	$1.4776\,e-1$ $(1.1\,e-4)$	$1.5091\,e-1$ $(1.1\,e-4)$

Table B.10: Mean running times and standard errors for unsymmetric `gilbert`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$9\,e+0$ (0)	$4\,e+0$ (0)	$3.2120\,e+1$ $(4.1\,e-2)$
qmr	$7.0100\,e+0$ $(1.0\,e-2)$	$4\,e+0$ (0)	$7.0200\,e+0$ $(1.4\,e-2)$
symmlq	$8\,e+0$ (0)	$3\,e+0$ (0)	N/A[12]
lsqr	$9\,e+0$ (0)	$4.6300\,e+0$ $(4.9\,e-2)$	$7.1300\,e+0$ $(3.4\,e-2)$
minres	$8\,e+0$ (0)	$3\,e+0$ (0)	N/A[12]
cgs	$1.2333\,e+1$ $(2.5\,e+0)$[11]	$2\,e+0$ (0)	$4\,e+0$ (0)
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$7.0800\,e+0$ $(3.4\,e-2)$	$4\,e+0$ (0)	$7.0200\,e+0$ $(1.4\,e-2)$
bicgstab	$4.1000\,e+0$ $(2.7\,e-2)$	$2\,e+0$ (0)	$3.5000\,e+0$ (0)

Table B.11: Mean iteration counts and standard errors for unsymmetric `gilbert`.

[11]27% of cases finished in state 0, 67% in state 1 and 6% in state 4
[12]All cases finished in state 1

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8.0662\,\mathrm{e}-11$ $(5.1\,\mathrm{e}-13)$	$6.0111\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-13)$	$9.2411\,\mathrm{e}-11$ $(4.7\,\mathrm{e}-13)$
qmr	$6.7310\,\mathrm{e}-11$ $(5.3\,\mathrm{e}-13)$	$2.8746\,\mathrm{e}-12$ $(6.9\,\mathrm{e}-15)$	$5.7592\,\mathrm{e}-12$ $(1.2\,\mathrm{e}-12)$
symmlq	$7.0394\,\mathrm{e}-11$ $(4.4\,\mathrm{e}-13)$	$5.9819\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-13)$	N/A[12]
lsqr	$2.0339\,\mathrm{e}-11$ $(6.0\,\mathrm{e}-14)$	$3.6598\,\mathrm{e}-11$ $(4.8\,\mathrm{e}-12)$	$5.8596\,\mathrm{e}-11$ $(2.2\,\mathrm{e}-12)$
minres	$6.9776\,\mathrm{e}-11$ $(4.4\,\mathrm{e}-13)$	$5.9819\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-13)$	N/A[12]
cgs	$4.4497\,\mathrm{e}-11$ $(5.2\,\mathrm{e}-12)$[11]	$4.3260\,\mathrm{e}-11$ $(8.0\,\mathrm{e}-14)$	$1.4064\,\mathrm{e}-12$ $(2.9\,\mathrm{e}-14)$
gmres	$6.4798\,\mathrm{e}-11$ $(1.5\,\mathrm{e}-13)$	$2.7882\,\mathrm{e}-12$ $(6.7\,\mathrm{e}-15)$	$1.4220\,\mathrm{e}-12$ $(4.8\,\mathrm{e}-15)$
bicg	$6.6694\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-12)$	$2.8752\,\mathrm{e}-12$ $(6.9\,\mathrm{e}-15)$	$6.0888\,\mathrm{e}-12$ $(1.2\,\mathrm{e}-12)$
bicgstab	$1.6545\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-12)$	$9.9392\,\mathrm{e}-12$ $(2.6\,\mathrm{e}-14)$	$2.0799\,\mathrm{e}-12$ $(2.2\,\mathrm{e}-14)$

Table B.12: Mean relative residuals and standard errors for unsymmetric `gilbert`.

B.2.5 Geometric Symmetric

Matrices generated by `geo(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$4.3740\,\mathrm{e}-2$ $(4.4\,\mathrm{e}-3)$	$8.6341\,\mathrm{e}-2$ $(3.0\,\mathrm{e}-4)$	N/A[14]
qmr	$6.0915\,\mathrm{e}-2$ $(9.5\,\mathrm{e}-4)$	$2.0542\,\mathrm{e}-1$ $(5.4\,\mathrm{e}-4)$	$1.6041\,\mathrm{e}-1$ $(1.0\,\mathrm{e}-3)$
symmlq	$3.1671\,\mathrm{e}-2$ $(5.5\,\mathrm{e}-4)$	$9.8814\,\mathrm{e}-2$ $(4.2\,\mathrm{e}-4)$	N/A[14]
lsqr	$1.0478\,\mathrm{e}-1$ $(5.7\,\mathrm{e}-4)$	$3.0237\,\mathrm{e}-1$ $(5.7\,\mathrm{e}-4)$	$2.6450\,\mathrm{e}-1$ $(5.5\,\mathrm{e}-4)$
minres	$3.2485\,\mathrm{e}-2$ $(4.0\,\mathrm{e}-4)$	$1.0391\,\mathrm{e}-1$ $(4.2\,\mathrm{e}-4)$	N/A[14]
cgs	$6.2964\,\mathrm{e}-2$ $(1.0\,\mathrm{e}-2)$[13]	$9.6193\,\mathrm{e}-2$ $(2.6\,\mathrm{e}-4)$	$6.3851\,\mathrm{e}-2$ $(4.1\,\mathrm{e}-4)$
gmres	$1.0328\,\mathrm{e}-1$ $(2.6\,\mathrm{e}-3)$	$1.6319\,\mathrm{e}-1$ $(4.0\,\mathrm{e}-4)$	$1.3193\,\mathrm{e}-1$ $(3.4\,\mathrm{e}-4)$
bicg	$4.8899\,\mathrm{e}-2$ $(3.4\,\mathrm{e}-4)$	$1.8049\,\mathrm{e}-1$ $(4.5\,\mathrm{e}-4)$	$1.4516\,\mathrm{e}-1$ $(9.7\,\mathrm{e}-4)$
bicgstab	$3.7670\,\mathrm{e}-2$ $(4.5\,\mathrm{e}-4)$	$8.7979\,\mathrm{e}-2$ $(2.8\,\mathrm{e}-4)$	$6.6107\,\mathrm{e}-2$ $(5.1\,\mathrm{e}-4)$

Table B.13: Mean running times and standard errors for `geo`.

[13] 33% of cases finished in state 0, 65% in state 1 and 2% in state 4

[14] All cases finished in state 1

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$11\,e+0$ (0)	$5\,e+0$ (0)	N/A[14]
qmr	$10\,e+0$ $(1.4\,e-2)$	$5.0100\,e+0$ $(1.0\,e-2)$	$7.2800\,e+0$ $(4.5\,e-2)$
symmlq	$10\,e+0$ (0)	$4\,e+0$ (0)	N/A[14]
lsqr	$13\,e+0$ (0)	$7\,e+0$ (0)	$11\,e+0$ (0)
minres	$10\,e+0$ (0)	$4\,e+0$ (0)	N/A[14]
cgs	$1.2818\,e+1$ $(2.2\,e+0)$[13]	$3\,e+0$ (0)	$4.0500\,e+0$ $(2.2\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$10\,e+0$ $(1.4\,e-2)$	$5.0100\,e+0$ $(1.0\,e-2)$	$7.2700\,e+0$ $(4.5\,e-2)$
bicgstab	$5.1500\,e+0$ $(3.9\,e-2)$	$2.500\,e+0$ (0)	$3.6250\,e+0$ $(2.4\,e-2)$

Table B.14: Mean iteration counts and standard errors for `geo`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$3.5124\,e-11$ $(5.2\,e-13)$	$3.2081\,e-12$ $(6.8\,e-14)$	N/A[14]
qmr	$1.5894\,e-11$ $(1.4\,e-12)$	$8.2311\,e-12$ $(9.7\,e-13)$	$3.8360\,e-11$ $(2.8\,e-12)$
symmlq	$3.5617\,e-11$ $(5.5\,e-13)$	$3.2080\,e-12$ $(6.8\,e-14)$	N/A[14]
lsqr	$2.9559\,e-11$ $(4.4\,e-13)$	$2.5582\,e-12$ $(5.3\,e-15)$	$3.056\,e-11$ $(1.4\,e-12)$
minres	$3.4893\,e-11$ $(5.4\,e-13)$	$3.2080\,e-12$ $(6.8\,e-14)$	N/A[14]
cgs	$3.1643\,e-11$ $(4.8\,e-12)$[13]	$2.0820\,e-13$ $(3.3\,e-15)$	$6.2571\,e-12$ $(1.1\,e-12)$
gmres	$5.0387\,e-11$ $(7.0\,e-14)$	$3.2201\,e-12$ $(6.7\,e-14)$	$2.2494\,e-11$ $(5.5\,e-13)$
bicg	$9.8877\,e-12$ $(9.7\,e-13)$	$8.2022\,e-12$ $(9.5\,e-13)$	$3.9295\,e-11$ $(2.8\,e-12)$
bicgstab	$3.2706\,e-11$ $(2.1\,e-12)$	$7.3500\,e-12$ $(1.4\,e-13)$	$4.8303\,e-11$ $(2.8\,e-12)$

Table B.15: Mean relative residuals and standard errors for `geo`.

B.2.6 Geometric Unsymmetric

Matrices generated by `geo(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[15]	$1.1326\,e-1$ $(2.3\,e-4)$	N/A[15]
qmr	$6.4551\,e-2$ $(3.4\,e-4)$	$1.9922\,e-1$ $(1.6\,e-3)$	$1.9357\,e-1$ $(1.2\,e-3)$
symmlq	N/A[15]	$1.2353\,e-1$ $(2.4\,e-4)$	N/A[15]
lsqr	$1.4364\,e-1$ $(6.3\,e-4)$	$2.8366\,e-1$ $(7.8\,e-4)$	$3.9641\,e-1$ $(1.6\,e-3)$
minres	N/A[15]	$1.2973\,e-1$ $(2.6\,e-4)$	N/A[15]
cgs	$3.0932\,e-2$ $(2.7\,e-4)$	$9.1098\,e-2$ $(5.7\,e-4)$	$7.6955\,e-2$ $(6.1\,e-4)$
gmres	$1.1676\,e-1$ $(3.3\,e-4)$	$1.5685\,e-1$ $(3.3\,e-4)$	$1.5448\,e-1$ $(4.3\,e-4)$
bicg	$5.4134\,e-2$ $(3.0\,e-4)$	$1.7646\,e-1$ $(1.6\,e-3)$	$1.7912\,e-1$ $(1.2\,e-3)$
bicgstab	$4.0522\,e-2$ $(2.4\,e-4)$	$8.4300\,e-2$ $(5.9\,e-4)$	$8.1445\,e-2$ $(4.8\,e-4)$

Table B.16: Mean running times and standard errors for unsymmetric `geo`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[15]	$7\,e+0$ (0)	N/A[15]
qmr	$1.1130\,e+1$ $(3.4\,e-2)$	$5.3100\,e+0$ $(4.6\,e-2)$	$9.5100\,e+0$ $(5.4\,e-2)$
symmlq	N/A[15]	$6\,e+0$ (0)	N/A[15]
lsqr	$1.9210\,e+1$ $(4.6\,e-2)$	$7.0200\,e+0$ $(1.4\,e-2)$	$1.7890\,e+1$ $(6.8\,e-2)$
minres	N/A[15]	$6\,e+0$ (0)	N/A[15]
cgs	$6.3200\,e+0$ $(4.7\,e-2)$	$3.0400\,e+0$ $(2.0\,e-2)$	$5.1600\,e+0$ $(3.9\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$1.1160\,e+1$ $(3.7\,e-2)$	$5.3100\,e+0$ $(4.6\,e-2)$	$9.5200\,e+0$ $(5.6\,e-2)$
bicgstab	$5.6400\,e+0$ $(2.4\,e-2)$	$2.5800\,e+0$ $(1.8\,e-2)$	$4.7100\,e+0$ $(2.5\,e-2)$

Table B.17: Mean iteration counts and standard errors for unsymmetric `geo`.

B.2.7 Preferential Attachment Symmetric

Matrices generated by `pref(10000)`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[15]	$1.6565\,\mathrm{e}-11$ $(1.8\,\mathrm{e}-13)$	N/A[15]
qmr	$4.3560\,\mathrm{e}-11$ $(2.3\,\mathrm{e}-12)$	$3.4265\,\mathrm{e}-11$ $(2.6557\,\mathrm{e}-12)$	$3.7655\,\mathrm{e}-11$ $(3.0\,\mathrm{e}-12)$
symmlq	N/A[15]	$1.6593\,\mathrm{e}-11$ $(1.8\,\mathrm{e}-13)$	N/A[15]
lsqr	$5.4887\,\mathrm{e}-11$ $(2.0\,\mathrm{e}-12)$	$3.6318\,\mathrm{e}-11$ $(1.6\,\mathrm{e}-12)$	$5.4997\,\mathrm{e}-11$ $(2.1\,\mathrm{e}-12)$
minres	N/A[15]	$1.6502\,\mathrm{e}-11$ $(1.8\,\mathrm{e}-13)$	N/A[15]
cgs	$1.4298\,\mathrm{e}-11$ $(1.9\,\mathrm{e}-12)$	$4.2325\,\mathrm{e}-12$ $(1.2\,\mathrm{e}-12)$	$1.5813\,\mathrm{e}-11$ $(2.2\,\mathrm{e}-12)$
gmres	$1.6764\,\mathrm{e}-11$ $(1.5\,\mathrm{e}-13)$	$3.0156\,\mathrm{e}-11$ $(4.0\,\mathrm{e}-13)$	$3.0733\,\mathrm{e}-11$ $(1.8\,\mathrm{e}-12)$
bicg	$4.3494\,\mathrm{e}-11$ $(2.5\,\mathrm{e}-12)$	$3.4313\,\mathrm{e}-11$ $(2.7\,\mathrm{e}-12)$	$3.6610\,\mathrm{e}-11$ $(3.0\,\mathrm{e}-12)$
bicgstab	$3.8476\,\mathrm{e}-11$ $(2.1\,\mathrm{e}-12)$	$3.6928\,\mathrm{e}-11$ $(2.1\,\mathrm{e}-12)$	$3.5322\,\mathrm{e}-11$ $(2.8\,\mathrm{e}-12)$

Table B.18: Mean relative residuals and standard errors for unsymmetric `geo`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[16]	$8.6264\,\mathrm{e}-2$ $(1.5\,\mathrm{e}-3)$	$9.5785\,\mathrm{e}-2$ $(1.9\,\mathrm{e}-4)$
qmr	$5.5683\,\mathrm{e}-2$ $(3.1\,\mathrm{e}-4)$	$2.0013\,\mathrm{e}-1$ $(5.0\,\mathrm{e}-4)$	$1.5002\,\mathrm{e}-1$ $(4.8\,\mathrm{e}-4)$
symmlq	N/A[16]	$9.5891\,\mathrm{e}-2$ $(1.2\,\mathrm{e}-4)$	$9.5970\,\mathrm{e}-2$ $(2.0\,\mathrm{e}-4)$
lsqr	$1.8629\,\mathrm{e}-1$ $(8.6\,\mathrm{e}-4)$	$2.6616\,\mathrm{e}-1$ $(5.5\,\mathrm{e}-4)$	$2.0207\,\mathrm{e}-1$ $(3.8\,\mathrm{e}-4)$
minres	N/A[16]	$1.0082\,\mathrm{e}-1$ $(1.4\,\mathrm{e}-4)$	$1.0725\,\mathrm{e}-1$ $(1.8\,\mathrm{e}-4)$
cgs	$2.3920\,\mathrm{e}-1$ $(1.1\,\mathrm{e}-1)$[17]	$9.2846\,\mathrm{e}-2$ $(1.1\,\mathrm{e}-4)$	$6.1941\,\mathrm{e}-2$ $(4.1\,\mathrm{e}-4)$
gmres	$1.0191\,\mathrm{e}-1$ $(3.3\,\mathrm{e}-4)$	$1.6041\,\mathrm{e}-1$ $(2.1\,\mathrm{e}-4)$	$1.2940\,\mathrm{e}-1$ $(1.9\,\mathrm{e}-4)$
bicg	$4.6707\,\mathrm{e}-2$ $(2.0\,\mathrm{e}-4)$	$1.7616\,\mathrm{e}-1$ $(3.2\,\mathrm{e}-4)$	$1.3847\,\mathrm{e}-1$ $(5.2\,\mathrm{e}-4)$
bicgstab	$4.1833\,\mathrm{e}-2$ $(7.2\,\mathrm{e}-4)$	$8.4533\,\mathrm{e}-2$ $(1.1\,\mathrm{e}-4)$	$6.2706\,\mathrm{e}-2$ $(2.1\,\mathrm{e}-4)$

Table B.19: Mean running times and standard errors for `pref`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[16]	$5\,e+0$ (0)	$11\,e+0$ (0)
qmr	$9.1400\,e+0$ $(3.5\,e-2)$	$5\,e+0$ (0)	$7.0400\,e+0$ $(2.0\,e-2)$
symmlq	N/A[16]	$4\,e+0$ (0)	$10\,e+0$ (0)
lsqr	$2.2770\,e+1$ $(9.2\,e-2)$	$6\,e+0$ (0)	$8\,e+0$ (0)
minres	N/A[16]	$4\,e+0$ (0)	$10\,e+0$ (0)
cgs	$48\,e+0$ $(8.6\,e-4)$[17]	$3\,e+0$ (0)	$4.0900\,e+0$ $(2.9\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$9.1100\,e+0$ $(3.1\,e-2)$	$5\,e+0$ (0)	$7.0600\,e+0$ $(2.4\,e-2)$
bicgstab	$5.5100\,e+0$ $(10.0\,e-2)$	$2.5000\,e+0$ (0)	$3.5300\,e+0$ $(1.2\,e-2)$

Table B.20: Mean iteration counts and standard errors for `pref`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[16]	$9.0607\,e-13$ $(2.1\,e-15)$	$5.6163\,e-11$ $(4.0\,e-13)$
qmr	$3.7483\,e-11$ $(2.1\,e-12)$	$4.4731\,e-13$ $(2.0\,e-15)$	$1.4312\,e-11$ $(1.4\,e-12)$
symmlq	N/A[16]	$9.0664\,e-13$ $(2.1\,e-15)$	$7.4791\,e-11$ $(5.7\,e-13)$
lsqr	$5.4649\,e-11$ $(2.0\,e-12)$	$1.1988\,e-11$ $(6.5\,e-13)$	$3.5166\,e-11$ $(2.8\,e-13)$
minres	N/A[16]	$9.0654\,e-13$ $(2.1\,e-15)$	$6.7365\,e-11$ $(5.0\,e-13)$
cgs	$9.6777\,e-11$ $(2.2\,e-12)$[17]	$8.2037\,e-15$ $(3.8\,e-17)$	$1.9262\,e-12$ $(5.3\,e-13)$
gmres	$1.8557\,e-11$ $(3.1\,e-14)$	$4.4280\,e-13$ $(2.0\,e-15)$	$5.4697\,e-12$ $(1.3\,e-13)$
bicg	$4.0678\,e-11$ $(2.1\,e-12)$	$4.4757\,e-13$ $(2.0\,e-15)$	$1.3659\,e-11$ $(1.4\,e-12)$
bicgstab	$3.6298\,e-11$ $(2.6\,e-12)$	$9.6333\,e-13$ $(3.8\,e-15)$	$1.2567\,e-11$ $(1.5\,e-12)$

Table B.21: Mean relative residuals and standard errors for `pref`.

B.2.8 Preferential Attachment Unsymmetric

Matrices generated by `pref(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[18]	$9.1802\,e-2$ $(2.7\,e-4)$	$1.7464\,e-1$ $(7.2\,e-4)$
qmr	$6.7300\,e-2$ $(4.1\,e-4)$	$2.1340\,e-1$ $(4.0\,e-4)$	$1.8468\,e-1$ $(5.5\,e-4)$
symmlq	N/A[18]	$1.0309\,e-1$ $(3.8\,e-4)$	N/A[18]
lsqr	$2.0732\,e-1$ $(1.2\,e-3)$	$3.2337\,e-1$ $(7.7\,e-4)$	$3.7287\,e-1$ $(2.6\,e-3)$
minres	N/A[18]	$1.0823\,e-1$ $(4.0\,e-4)$	N/A[18]
cgs	$3.2758\,e-2$ $(4.2\,e-4)$	$1.0065\,e-1$ $(4.0\,e-4)$	$7.1948\,e-2$ $(7.7\,e-4)$
gmres	$1.1257\,e-1$ $(5.3\,e-4)$	$1.6700\,e-1$ $(4.2\,e-4)$	$1.4721\,e-1$ $(4.1\,e-4)$
bicg	$5.6565\,e-2$ $(3.6\,e-4)$	$1.8892\,e-1$ $(3.8056\,e-4)$	$1.7117\,e-1$ $(5.1\,e-4)$
bicgstab	$4.3494\,e-2$ $(3.7\,e-4)$	$9.2052\,e-2$ $(2.8\,e-4)$	$7.7681\,e-2$ $(3.1\,e-4)$

Table B.22: Mean running times and standard errors for unsymmetric `geo`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[18]	$5\,e+0$ (0)	$1.8860\,e+1$ $(4.9\,e-2)$
qmr	$1.0080\,e+1$ $(2.7\,e-2)$	$5\,e+0$ (0)	$8.0400\,e+0$ $(2.0\,e-2)$
symmlq	N/A[18]	$4\,e+0$ (0)	N/A[18]
lsqr	$2.2820\,e+1$ $(9.3\,e-2)$	$7.0200\,e+0$ $(1.4\,e-2)$	$1.4530\,e+1$ $(1.0\,e-1)$
minres	N/A[18]	$4\,e+0$ (0)	N/A[18]
cgs	$5.4400\,e+0$ $(5.4\,e-2)$	$3.0100\,e+0$ $(1.0\,e-2)$	$4.3000\,e+0$ $(4.6\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$1.0100\,e+1$ $(3.0\,e-2)$	$5\,e+0$ (0)	$8.0300\,e+0$ $(1.7\,e-2)$
bicgstab	$5.0350\,e+0$ $(1.3\,e-2)$	$2.5000\,e+0$ (0)	$4.0100\,e+0$ $(7.0\,e-3)$

Table B.23: Mean iteration counts and standard errors for unsymmetric `geo`.

[15] All cases finished in state 1
[16] All cases finished in state 1
[17] 2% of cases finished in state 0 and 98% in state 1
[18] All cases finished in state 1

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[18]	3.6012 e − 11 (2.7 e − 13)	8.1532 e − 11 (1.1 e − 12)
qmr	3.1354 e − 11 (2.1 e − 12)	3.5730 e − 12 (8.7 e − 13)	1.5556 e − 11 (1.5 e − 12)
symmlq	N/A[18]	3.6031 e − 11 (2.7 e − 13)	N/A[18]
lsqr	5.9152 e − 11 (2.1 e − 12)	2.5528 e − 11 (1.6 e − 12)	4.5799 e − 11 (2.3 e − 12)
minres	N/A[18]	3.5993 e − 11 (2.7 e − 13)	N/A[18]
cgs	3.2534 e − 11 (3.1 e − 12)	3.8497 e − 13 (1.7 e − 13)	2.3528 e − 11 (2.3 e − 12)
gmres	8.6661 e − 12 (2.7 e − 14)	5.8626 e − 13 (2.6 e − 15)	6.0549 e − 12 (9.2 e − 13)
bicg	3.0334 e − 11 (2.0 e − 12)	3.4408 e − 12 (7.8823 e − 13)	1.6454 e − 11 (1.6 e − 12)
bicgstab	1.7061 e − 11 (1.5 e − 12)	2.8621 e − 12 (9.4 e − 13)	1.2058 e − 11 (1.3 e − 12)

Table B.24: Mean relative residuals and standard errors for unsymmetric `geo`.

B.2.9 RENGA Symmetric

Matrices generated by `renga(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	3.5206 e − 2 (1.2 e − 4)	6.3828 e − 2 (2.3 e − 4)	1.5354 e − 1 (5.9 e − 4)
qmr	5.5414 e − 2 (2.2 e − 4)	1.4965 e − 1 (1.6 e − 4)	1.7255 e − 1 (9.4 e − 4)
symmlq	2.8239 e − 2 (1.1 e − 4)	7.4301 e − 2 (1.7 e − 4)	N/A[21]
lsqr	9.0222 e − 2 (2.2 e − 4)	2.0396 e − 1 (4.2 e − 4)	1.9746 e − 1 (5.0 e − 4)
minres	2.9621 e − 2 (1.4 e − 4)	7.7905 e − 2 (1.6 e − 4)	N/A[21]
cgs	5.2388 e − 2 (4.1 e − 3)[19]	5.9017 e − 2 (1.4 e − 4)	7.5688 e − 2 (7.2 e − 4)
gmres	1.0171 e − 1 (1.2 e − 4)	1.3495 e − 1 (1.7 e − 4)	1.4641 e − 1 (2.0 e − 4)
bicg	4.2842 e − 2 (1.8 e − 4)	1.3065 e − 1 (1.8 e − 4)	1.5663 e − 1 (8.5 e − 4)
bicgstab	3.8749 e − 2 (6.1 e − 4)[20]	6.4782 e − 2 (1.5 e − 4)	7.5767 e − 2 (4.5 e − 4)

Table B.25: Mean running times and standard errors for `renga`.

[19]31% of cases finished in state 0, 35% in state 1 and 34% in state 4
[20]98% of cases finished in state 0 and 2% in state 1
[21]All cases finished in state 1

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$10\,e+0$ (0)	$4\,e+0$ (0)	$18\,e+0$ (0)
qmr	$9.0700\,e+0$ $(2.6\,e-2)$	$4\,e+0$ (0)	$8.6600\,e+0$ $(4.8\,e-2)$
symmlq	$9\,e+0$ (0)	$3\,e+0$ (0)	N/A[21]
lsqr	$12\,e+0$ (0)	$5\,e+0$ (0)	$9\,e-+0$ (0)
minres	$9\,e+0$ (0)	$3\,e+0$ (0)	N/A[21]
cgs	$1.0581\,e+1$ $(7.8\,e-1)$[29]	$2\,e+0$ (0)	$4.7900\,e+0$ $(4.6\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$9.0700\,e+0$ $(2.6\,e-2)$	$4\,e+0$ (0)	$8.6800\,e+0$ $(4.7\,e-2)$
bicgstab	$5.3418\,e+0$ $(8.3\,e-2)$[20]	$2\,e+0$ (0)	$4.1200\,e+0$ $(2.1\,e-2)$

Table B.26: Mean iteration counts and standard errors for `renga`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8.3666\,e-11$ $(6.2\,e-13)$	$8.6564\,e-12$ $(1.2\,e-13)$	$6.9782\,e-11$ $(4.2\,e-13)$
qmr	$3.0748\,e-11$ $(1.9\,e-12)$	$8.6606\,e-12$ $(1.2\,e-13)$	$3.3599\,e-11$ $(3.4\,e-12)$
symmlq	$7.8617\,e-11$ $(5.9\,e-13)$	$8.6562\,e-12$ $(1.2\,e-13)$	N/A[21]
lsqr	$2.2933\,e-11$ $(1.7\,e-13)$	$3.2218\,e-11$ $(8.9\,e-13)$	$3.8355\,e-11$ $(1.3\,e-12)$
minres	$7.7324\,e-11$ $(5.8\,e-13)$	$8.6561\,e-12$ $(1.2\,e-13)$	N/A[21]
cgs	$3.5382\,e-11$ $(4.2\,e-12)$[19]	$2.0660\,e-11$ $(1.1\,e-13)$	$2.4845\,e-11$ $(3.7\,e-12)$
gmres	$1.7666\,e-11$ $(8.3\,e-14)$	$8.7993\,e-12$ $(1.2\,e-13)$	$4.6801\,e-11$ $(1.1\,e-13)$
bicg	$3.0687\,e-11$ $(1.6\,e-12)$	$8.6624\,e-12$ $(1.2\,e-13)$	$3.2067\,e-11$ $(3.3\,e-12)$
bicgstab	$3.1871\,e-11$ $(2.8\,e-12)$[20]	$1.5507\,e-11$ $(1.9\,e-13)$	$4.8527\,e-11$ $(2.6\,e-12)$

Table B.27: Mean relative residuals and standard errors for `renga`.

B.2.10 RENGA Unsymmetric

Matrices generated by `renga(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$4.5053\,e-2$ $(6.3\,e-3)$	$7.0716\,e-2$ $(2.9\,e-4)$	$1.4525\,e-1$ $(8.1\,e-4)$
qmr	$6.2019\,e-2$ $(1.7\,e-3)$	$1.6209\,e-1$ $(4.2\,e-4)$	$1.7778\,e-1$ $(1.2\,e-3)$
symmlq	$3.3113\,e-2$ $(2.8\,e-3)$	$8.0637\,e-2$ $(4.3\,e-4)$	N/A[23]
lsqr	$1.0981\,e-1$ $(8.3\,e-3)$	$2.2576\,e-1$ $(5.4\,e-4)$	$2.2338\,e-1$ $(8.2\,e-4)$
minres	$3.0833\,e-2$ $(4.4\,e-4)$	$8.5137\,e-2$ $(4.4\,e-4)$	$1.7774\,e-1$ $(1.0\,e-3)$
cgs	$5.2967\,e-2$ $(2.7\,e-3)$[22]	$6.4268\,e-2$ $(2.7\,e-4)$	$7.7839\,e-2$ $(8.6\,e-4)$
gmres	$1.0858\,e-1$ $(5.2\,e-3)$	$1.4035\,e-1$ $(3.8\,e-4)$	$1.5149\,e-1$ $(4.4\,e-4)$
bicg	$5.0834\,e-2$ $(4.0\,e-4)$	$1.4203\,e-1$ $(3.7\,e-4)$	$1.6352\,e-1$ $(1.1\,e-3)$
bicgstab	$4.1765\,e-2$ $(6.7\,e-4)$	$7.0935\,e-2$ $(2.9\,e-4)$	$8.2060\,e-2$ $(5.2\,e-4)$

Table B.28: Mean running times and standard errors for unsymmetric `renga`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$9\,e+0$ (0)	$4\,e+0$ (0)	$1.4940\,e+1$ $(2.4\,e-2)$
qmr	$9.0700\,e+0$ $(2.6\,e-2)$	$4\,e+0$ (0)	$8.2900\,e+0$ $(4.8\,e-2)$
symmlq	$8\,e+0$ (0)	$3\,e+0$ (0)	N/A[23]
lsqr	$11\,e+0$ (0)	$5\,e+0$ (0)	$9\,e+0$ (0)
minres	$8\,e+0$ (0)	$3\,e+0$ (0)	$1.5780\,e+1$ $(4.2\,e-2)$
cgs	$9.2059\,e+0$ $(4.7\,e-1)$[22]	$2\,e+0$ (0)	$4.5100\,e+0$ $(5.2\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$9.1000\,e+0$ $(3.0\,e-2)$	$4\,e+0$ (0)	$8.3000\,e+0$ $(5.0\,e-2)$
bicgstab	$4.8850\,e+0$ $(6.0\,e-2)$	$2\,e+0$ (0)	$4.0700\,e+0$ $(1.7\,e-2)$

Table B.29: Mean iteration counts and standard errors for unsymmetric `renga`.

[22] 34% of cases finished in state 0, 46% in state 1 and 20% in state 4
[23] All cases finished in state 1

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	5.7732 e − 11 (3.4 e − 13)	1.8171 e − 12 (2.6 e − 14)	5.7950 e − 11 (1.1 e − 12)
qmr	1.7848 e − 11 (1.5 e − 12)	1.8203 e − 12 (2.6 e − 14)	4.0832 e − 11 (2.7 e − 12)
symmlq	5.9544 e − 11 (3.9 e − 13)	1.8170 e − 12 (2.6 e − 14)	N/A[23]
lsqr	5.0959 e − 11 (6.1 e − 13)	8.7265 e − 12 (3.4 e − 13)	1.3503 e − 11 (6.4 e − 13)
minres	5.9055 e − 11 (3.8 e − 13)	1.8170 e − 12 (2.6 e − 14)	6.2905 e − 11 (1.8 e − 12)
cgs	3.5944 e − 11 (5.3 e − 12)[22]	3.7667 e − 12 (2.5 e − 14)	3.0046 e − 11 (3.0 e − 12)
gmres	2.1076 e − 11 (3.2 e − 12)	1.8709 e − 12 (2.7 e − 14)	2.2693 e − 11 (4.3 e − 14)
bicg	2.4720 e − 11 (1.9 e − 12)	1.8204 e − 12 (2.6 e − 14)	4.0882 e − 11 (2.8 e − 12)
bicgstab	3.7027 e − 11 (2.5 e − 12)	3.2768 e − 12 (3.7 e − 14)	3.1679 e − 11 (2.0 e − 12)

Table B.30: Mean relative residuals and standard errors for unsymmetric `renga`.

B.2.11 Kleinberg Symmetric

Matrices generated by `klein(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	3.9441 e − 2 (1.7 e − 4)	7.8759 e − 2 (2.6 e − 4)	1.3515 e − 1 (7.7 e − 4)
qmr	6.1598 e − 2 (3.1 e − 4)	1.8285 e − 1 (2.8 e − 4)	2.0338 e − 1 (1.2 e − 3)
symmlq	3.0901 e − 2 (1.2 e − 4)	8.9166 e − 2 (2.1 e − 4)	1.4343 e − 1 (6.2 e − 4)
lsqr	1.1220 e − 1 (3.1 e − 4)	2.5660 e − 1 (5.3 e − 4)	2.2953 e − 1 (6.7 e − 4)
minres	3.0734 e − 2 (1.4 e − 4)	9.4872 e − 2 (2.1 e − 4)	1.5814 e − 1 (6.4 e − 4)
cgs	7.4085 e − 2 (9.3 e − 3)[24]	7.2069 e − 2 (1.9 e − 4)	9.0449 e − 2 (1.1 e − 3)
gmres	1.0156 e − 1 (1.9 e − 4)	1.5084 e − 1 (2.4 e − 4)	1.6140 e − 1 (3.8 e − 4)
bicg	5.2336 e − 2 (1.9 e − 4)	1.6183 e − 1 (3.0 e − 4)	1.8379 e − 1 (1.2 e − 3)
bicgstab	4.6270 e − 2 (9.0 e − 4)	7.9901 e − 2 (1.8 e − 4)	9.3411 e − 2 (6.3 e − 4)

Table B.31: Mean running times and standard errors for `klein`.

B.2.12 Kleinberg Unsymmetric

Matrices generated by `klein(10000)`.

[24] 41% of cases finished in state 0, 37% in state 1 and 22% in state 4

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8\,e+0$ (0)	$4\,e+0$ (0)	$1.3515\,e-1$ ($7.7\,e-4$)
qmr	$8\,e+0$ (0)	$4\,e+0$ (0)	$8.3300\,e+0$ ($4.7\,e-2$)
symmlq	$7\,e+0$ (0)	$3\,e+0$ (0)	$1.2980\,e+1$ ($1.4\,e-2$)
lsqr	$10\,e+0$ (0)	$5\,e+0$ (0)	$8\,e+0$ (0)
minres	$7\,e+0$ (0)	$3\,e+0$ (0)	$12\,e+0$ (0)
cgs	$1.1415\,e+1$ ($1.5\,e+0$)[24]	$2\,e+0$ (0)	$4.6700\,e+0$ ($5.5\,e-2$)
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$8\,e+0$ (0)	$4\,e+0$ (0)	$8.3100\,e+0$ ($4.6\,e-2$)
bicgstab	$4.7850\,e+0$ ($9.3\,e-2$)	$2\,e+0$ (0)	$4.1100\,e+0$ ($2.1\,e-2$)

Table B.32: Mean iteration counts and standard errors for `klein`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$7.5934\,e-11$ ($2.9\,e-13$)	$3.4481\,e-12$ ($1.9\,e-14$)	$5.8430\,e-11$ ($2.3\,e-12$)
qmr	$3.2371\,e-11$ ($7.6\,e-13$)	$3.4722\,e-12$ ($1.9\,e-14$)	$3.5048\,e-11$ ($2.7\,e-12$)
symmlq	$7.7355\,e-11$ ($3.0\,e-13$)	$3.4481\,e-12$ ($1.9\,e-14$)	$7.6157\,e-11$ ($7.0\,e-13$)
lsqr	$5.3618\,e-11$ ($4.1\,e-13$)	$2.8592\,e-12$ ($3.9\,e-14$)	$2.8131\,e-11$ ($1.7\,e-13$)
minres	$7.7078\,e-11$ ($3.0\,e-13$)	$3.4480\,e-12$ ($1.9\,e-14$)	$7.8038\,e-11$ ($5.5\,e-13$)
cgs	$3.8435\,e-11$ ($4.6\,e-12$)[24]	$7.8089\,e-12$ ($3.1\,e-14$)	$2.2142\,e-11$ ($2.9\,e-12$)
gmres	$2.4697\,e-11$ ($1.2\,e-13$)	$3.5381\,e-12$ ($2.0\,e-14$)	$1.7728\,e-11$ ($2.4\,e-14$)
bicg	$3.6201\,e-11$ ($3.8\,e-13$)	$3.4730\,e-12$ ($1.9\,e-14$)	$3.7416\,e-11$ ($2.9\,e-12$)
bicgstab	$4.3083\,e-11$ ($2.7\,e-12$)	$7.4837\,e-12$ ($4.4\,e-14$)	$3.0859\,e-11$ ($2.3\,e-12$)

Table B.33: Mean relative residuals and standard errors for `klein`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$5.1116\,\mathrm{e}-2$ $(3.2\,\mathrm{e}-3)$	$8.3815\,\mathrm{e}-2$ $(2.4\,\mathrm{e}-4)$	$1.4347\,\mathrm{e}-1$ $(4.8\,\mathrm{e}-4)$
qmr	$7.0876\,\mathrm{e}-2$ $(6.7\,\mathrm{e}-4)$	$1.9205\,\mathrm{e}-1$ $(4.0\,\mathrm{e}-4)$	$2.1039\,\mathrm{e}-1$ $(8.6\,\mathrm{e}-4)$
symmlq	$3.6659\,\mathrm{e}-2$ $(4.4\,\mathrm{e}-4)$	$9.3765\,\mathrm{e}-2$ $(3.8\,\mathrm{e}-4)$	$1.4650\,\mathrm{e}-1$ $(5.2\,\mathrm{e}-4)$
lsqr	$1.3227\,\mathrm{e}-1$ $(6.0\,\mathrm{e}-4)$	$2.7425\,\mathrm{e}-1$ $(4.9\,\mathrm{e}-4)$	$2.5428\,\mathrm{e}-1$ $(5.7\,\mathrm{e}-4)$
minres	$3.6184\,\mathrm{e}-2$ $(4.5\,\mathrm{e}-4)$	$9.9940\,\mathrm{e}-2$ $(4.0\,\mathrm{e}-4)$	$1.6645\,\mathrm{e}-1$ $(8.4\,\mathrm{e}-4)$
cgs	$7.1681\,\mathrm{e}-2$ $(6.9\,\mathrm{e}-3)$[25]	$7.6144\,\mathrm{e}-2$ $(2.4\,\mathrm{e}-4)$	$9.9876\,\mathrm{e}-2$ $(1.1\,\mathrm{e}-3)$
gmres	$1.0313\,\mathrm{e}-1$ $(7.3\,\mathrm{e}-4)$	$1.5212\,\mathrm{e}-1$ $(4.3\,\mathrm{e}-4)$	$1.6531\,\mathrm{e}-1$ $(4.3\,\mathrm{e}-4)$
bicg	$6.1074\,\mathrm{e}-2$ $(4.9\,\mathrm{e}-4)$	$1.7082\,\mathrm{e}-1$ $(3.6\,\mathrm{e}-4)$	$1.9659\,\mathrm{e}-1$ $(9.7\,\mathrm{e}-4)$
bicgstab	$5.4624\,\mathrm{e}-2$ $(1.0\,\mathrm{e}-3)$	$8.4428\,\mathrm{e}-2$ $(2.7\,\mathrm{e}-4)$	$1.0192\,\mathrm{e}-1$ $(5.5\,\mathrm{e}-4)$

Table B.34: Mean running times and standard errors for unsymmetric `klein`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$12\,\mathrm{e}+0$ (0)
qmr	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.1000\,\mathrm{e}+0$ $(3.0\,\mathrm{e}-2)$
symmlq	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	$12\,\mathrm{e}+0$ (0)
lsqr	$10\,\mathrm{e}+0$ (0)	$5\,\mathrm{e}+0$ (0)	$8\,\mathrm{e}+0$ (0)
minres	$7\,\mathrm{e}+0$ (0)	$3\,\mathrm{e}+0$ (0)	$1.1600\,\mathrm{e}+1$ $(4.9\,\mathrm{e}-2)$
cgs	$8.9375\,\mathrm{e}+0$ $(8.8\,\mathrm{e}-1)$[25]	$2\,\mathrm{e}+0$ (0)	$4.7300\,\mathrm{e}+0$ $(5.1\,\mathrm{e}-2)$
gmres	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)	$1\,\mathrm{e}+0$ (0)
bicg	$8\,\mathrm{e}+0$ (0)	$4\,\mathrm{e}+0$ (0)	$8.1300\,\mathrm{e}+0$ $(3.4\,\mathrm{e}-2)$
bicgstab	$4.6650\,\mathrm{e}+0$ $(8.2\,\mathrm{e}-2)$	$2\,\mathrm{e}+0$ (0)	$4.0950\,\mathrm{e}+0$ $(2.0\,\mathrm{e}-2)$

Table B.35: Mean iteration counts and standard errors for unsymmetric `klein`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$2.9904\,e-11$ $(1.1\,e-13)$	$1.8986\,e-12$ $(8.8\,e-15)$	$6.6673\,e-11$ $(3.5\,e-13)$
qmr	$2.0123\,e-11$ $(8.3\,e-13)$	$1.9045\,e-12$ $(8.8\,e-15)$	$3.6976\,e-11$ $(2.0\,e-12)$
symmlq	$3.0227\,e-11$ $(1.1\,e-13)$	$1.8986\,e-12$ $(8.8\,e-15)$	$7.5671\,e-11$ $(5.1\,e-13)$
lsqr	$2.8162\,e-11$ $(2.1\,e-13)$	$1.4876\,e-12$ $(3.0\,e-14)$	$1.6050\,e-11$ $(8.7\,e-14)$
minres	$3.0151\,e-11$ $(1.1\,e-13)$	$1.8986\,e-12$ $(8.8\,e-15)$	$6.9545\,e-11$ $(2.2\,e-12)$
cgs	$3.3615\,e-11$ $(4.6\,e-12)$[25]	$4.6135\,e-12$ $(1.9\,e-14)$	$2.3129\,e-11$ $(3.6\,e-12)$
gmres	$1.5069\,e-11$ $(6.9\,e-14)$	$1.9547\,e-12$ $(9.4\,e-15)$	$1.3894\,e-11$ $(1.4\,e-14)$
bicg	$2.2096\,e-11$ $(2.4\,e-13)$	$1.9047\,e-12$ $(8.9\,e-15)$	$3.4938\,e-11$ $(1.9\,e-12)$
bicgstab	$4.4629\,e-11$ $(2.7\,e-12)$	$4.5373\,e-12$ $(2.2\,e-14)$	$3.2330\,e-11$ $(2.3\,e-12)$

Table B.36: Mean relative residuals and standard errors for unsymmetric `klein`.

B.2.13 Lock and Key Symmetric

Matrices generated by `lockandkey(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$7.1575\,e-3$ $(1.4\,e-3)$[26]	$9.6912\,e-3$ $(8.1\,e-4)$	$1.6911\,e-2$ $(2.3\,e-3)$
qmr	$8.2473\,e-3$ $(8.9\,e-4)$[27]	$2.2802\,e-2$ $(2.0\,e-3)$	$1.7201\,e-2$ $(1.6\,e-3)$
symmlq	$2.0523\,e-2$ $(3.6\,e-3)$[26]	$1.8572\,e-2$ $(1.3\,e-3)$	$1.9948\,e-2$ $(2.2\,e-3)$[31]
lsqr	$8.6489\,e-3$ $(3.6\,e-4)$	N/A[30]	$5.3995\,e-2$ $(2.2\,e-3)$[32]
minres	$9.9927\,e-3$ $(1.1\,e-3)$[26]	$1.9658\,e-2$ $(1.3\,e-3)$	$2.1526\,e-2$ $(2.3\,e-3)$[31]
cgs	$7.3400\,e-3$ $(2.1\,e-3)$[28]	$1.5684\,e-2$ $(1.4\,e-3)$	$1.2531\,e-2$ $(1.3\,e-3)$
gmres	$4.9444\,e-2$ $(1.5\,e-3)$	$6.0070\,e-2$ $(1.6\,e-3)$	$5.4937\,e-2$ $(1.3\,e-3)$
bicg	$5.1370\,e-3$ $(5.1\,e-4)$[29]	$1.7215\,e-2$ $(1.5\,e-3)$	$1.3601\,e-2$ $(1.3\,e-3)$
bicgstab	$4.6786\,e-3$ $(4.0\,e-4)$	$1.1585\,e-2$ $(1.2\,e-3)$	$1.0971\,e-2$ $(1.2\,e-3)$

Table B.37: Mean running times and standard errors for `lockandkey`.

[25] 32% of cases finished in state 0, 37% in state 1 and 31% in state 4

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$2.8723\,e+0\ (3.9\,e-1)^{26}$	$1.1700\,e+0\ (4.0\,e-2)$	$2.8800\,e+0\ (3.4\,e-1)$
qmr	$1.8586\,e+0\ (2.0\,e-1)^{27}$	$1.1700\,e+0\ (4.0\,e-2)$	$1.4800\,e+0\ (8.0\,e-2)$
symmlq	$1.0053\,e+1\ (2.1\,e+0)^{26}$	$1.0100\,e+0\ (1.0\,e-2)$	$2.3367\,e+0\ (2.6\,e-1)^{31}$
lsqr	$1.5600\,e+0\ (8.4\,e-2)$	N/A^{30}	$3.1515\,e+0\ (1.5\,e-1)^{32}$
minres	$3.1064\,e+0\ (4.7\,e-1)^{26}$	$1.0100\,e+0\ (1.0\,e-2)$	$2.3061\,e+0\ (2.5\,e-1)^{31}$
cgs	$2.6322\,e+0\ (7.6\,e-1)^{28}$	$1.1600\,e+0\ (3.7\,e-2)$	$1.4700\,e+0\ (7.6\,e-2)$
gmres	$1\,e+0\ (0)$	$1\,e+0\ (0)$	$1\,e+0\ (0)$
bicg	$1.7021\,e+0\ (1.8\,e-1)^{29}$	$1.1700\,e+0\ (4.0\,e-2)$	$1.4800\,e+0\ (8.0\,e-2)$
bicgstab	$9.4500\,e-1\ (6.5\,e-2)$	$6.6500\,e-1\ (3.8\,e-2)$	$9.6000\,e-1\ (7.3\,e-2)$

Table B.38: Mean iteration counts and standard errors for `lockandkey`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$6.8889\,e-12\ (1.5\,e-12)^{26}$	$4.8725\,e-13\ (4.4\,e-13)$	$1.1774\,e-11\ (2.4\,e-12)$
qmr	$1.7728\,e-12\ (7.4\,e-13)^{27}$	$5.1727\,e-17\ (1.0\,e-17)$	$1.9500\,e-16\ (4.3\,e-17)$
symmlq	$1.4739\,e-11\ (3.3\,e-12)^{26}$	$5.7671\,e-13\ (4.4\,e-13)$	$1.1230\,e-11\ (2.5\,e-12)^{31}$
lsqr	$2.4556\,e-16\ (4.3\,e-17)$	N/A^{30}	$1.8553\,e-16\ (3.1\,e-17)^{32}$
minres	$6.6745\,e-12\ (1.5\,e-12)^{26}$	$5.7684\,e-13\ (4.4\,e-13)$	$1.1720\,e-11\ (2.6\,e-12)^{31}$
cgs	$4.0576\,e-12\ (1.7\,e-12)^{28}$	$7.3252\,e-15\ (7.3\,e-15)$	$1.2098\,e-16\ (1.2\,e-16)$
gmres	$2.1409\,e-13\ (3.8\,e-15)$	$3.0917\,e-13\ (1.2\,e-14)$	$2.6325\,e-13\ (8.6\,e-15)$
bicg	$1.5577\,e-12\ (8.4\,e-13)^{29}$	$4.9720\,e-18\ (1.2\,e-18)$	$1.7983\,e-16\ (4.4\,e-17)$
bicgstab	$2.8427\,e-18\ (7.9\,e-19)$	$1.9780\,e-16\ (2.0\,e-16)$	$1.1301\,e-12\ (8.4\,e-13)$

Table B.39: Mean relative residuals and standard errors for `lockandkey`.

B.2.14 Lock and Key Unsymmetric

Matrices generated by `lockandkey(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$6.4131\,e-3$ $(6.2\,e-4)$[33]	$7.5186\,e-3$ $(3.0\,e-4)$	$1.1582\,e-2$ $(1.0\,e-3)$[37]
qmr	$7.8478\,e-3$ $(7.3\,e-4)$	$1.6445\,e-2$ $(6.8\,e-4)$	$1.9801\,e-2$ $(1.7\,e-3)$
symmlq	$6.6345\,e-3$ $(3.6\,e-4)$[34]	$1.7777\,e-2$ $(8.5\,e-4)$	$1.5498\,e-2$ $(1.1\,e-3)$[37]
lsqr	$1.1477\,e-2$ $(6.6\,e-4)$	$2.9073\,e-2$ $(1.1\,e-3)$[36]	$3.9667\,e-2$ $(2.4\,e-3)$[38]
minres	$7.6789\,e-3$ $(4.5\,e-4)$[35]	$1.8869\,e-2$ $(8.5\,e-4)$	$1.7390\,e-2$ $(1.1\,e-3)$[37]
cgs	$5.2284\,e-3$ $(3.5\,e-4)$	$1.1571\,e-2$ $(5.4\,e-4)$	$1.2455\,e-2$ $(9.7\,e-4)$
gmres	$4.9074\,e-2$ $(2.2\,e-3)$	$5.5348\,e-2$ $(8.2\,e-4)$	$5.5245\,e-2$ $(1.3\,e-3)$
bicg	$5.3762\,e-3$ $(4.3\,e-4)$	$1.2847\,e-2$ $(5.5\,e-4)$	$1.5715\,e-2$ $(1.4\,e-3)$
bicgstab	$5.9524\,e-3$ $(5.0\,e-4)$	$7.4673\,e-3$ $(2.9\,e-4)$	$1.2092\,e-2$ $(9.6\,e-4)$

Table B.40: Mean running times and standard errors for unsymmetric `lockandkey`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$3.0864\,e+0$ $(3.3\,e-1)$[33]	$1\,e+0$ (0)	$2.5222\,e+0$ $(2.0\,e-1)$[37]
qmr	$1.8400\,e+0$ $(9.2\,e-2)$	$1\,e+0$ (0)	$1.9600\,e+0$ $(1.0\,e-1)$
symmlq	$2.2564\,e+0$ $(2.1\,e-1)$[34]	$1\,e+0$ (0)	$2.0333\,e+0$ $(1.8\,e-1)$[37]
lsqr	$2.5700\,e+0$ $(1.6\,e-1)$	$1\,e+0$ (0)[36]	$3.3571\,e+0$ $(1.4\,e-1)$[38]
minres	$2.3797\,e+0$ $(2.4\,e-1)$[35]	$1\,e+0$ (0)	$2.0222\,e+0$ $(1.7\,e-1)$[37]
cgs	$1.8100\,e+0$ $(8.4\,e-2)$	$1\,e+0$ (0)	$1.7500\,e+0$ $(7.8\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$1.8400\,e+0$ $(9.2\,e-2)$	$1\,e+0$ (0)	$1.9600\,e+0$ $(1.0\,e-1)$
bicgstab	$1.3200\,e+0$ $(8.7\,e-2)$	$5.0000\,e-1$ (0)	$1.2950\,e+0$ $(8.2\,e-2)$

Table B.41: Mean iteration counts and standard errors for unsymmetric `lockandkey`.

[26] 94% of cases finished in state 0 and 6% in state 1
[27] 99% of cases finished in state 0 and 1% in state 1
[28] 87% of cases finished in state 0, 2% in state 1, 5% in state 3 and 6% in state 4
[29] 94% of cases finished in state 0, 4% in state 1 and 5% in state 3
[30] 67% of cases finished in state 1, 21% in state 3 and 12% in state 4
[31] 98% of cases finished in state 0 and 2% in state 1
[32] 33% of cases finished in state 0 and 67% in state 2

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	$1.4613\,e-11$ $(3.1\,e-12)$[33]	$1.0158\,e-18$ $(1.0\,e-19)$	$1.1091\,e-11$ $(2.4\,e-12)$[37]
qmr	$2.8011\,e-16$ $(5.0\,e-17)$	$3.7417\,e-17$ $(8.8\,e-18)$	$9.5981\,e-13$ $(9.6\,e-13)$
symmlq	$1.2422\,e-11$ $(2.8\,e-12)$[34]	$9.4417\,e-14$ $(3.4\,e-15)$	$1.1252\,e-11$ $(2.5\,e-12)$[37]
lsqr	$8.8768\,e-13$ $(7.0\,e-13)$	$2.0205\,e-18$ $(7.8\,e-20)$[36]	$3.6997\,e-13$ $(3.7\,e-13)$[38]
minres	$1.2823\,e-11$ $(2.8\,e-12)$[35]	$9.4512\,e-14$ $(3.4\,e-15)$	$1.1083\,e-11$ $(2.4\,e-12)$[37]
cgs	$2.3158\,e-13$ $(2.3\,e-13)$	$8.0436\,e-19$ $(8.9\,e-20)$	$1.5717\,e-12$ $(5.0\,e-13)$
gmres	$1.9012\,e-13$ $(4.5\,e-15)$	$3.3794\,e-13$ $(1.2\,e-14)$	$4.6716\,e-13$ $(2.2\,e-13)$
bicg	$9.9566\,e-17$ $(1.7\,e-17)$	$3.1843\,e-17$ $(8.3\,e-18)$	$9.5978\,e-13$ $(9.6\,e-13)$
bicgstab	$9.4871\,e-13$ $(8.8\,e-13)$	$1.0158\,e-18$ $(1.0\,e-19)$	$2.9658\,e-12$ $(1.2\,e-12)$

Table B.42: Mean relative residuals and standard errors for unsymmetric `lockandkey`.

B.2.15 Stickiness Symmetric

Matrices generated by `sticky(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[39]	$1.6812\,e-1$ $(1.2\,e-2)$[41]	N/A[39]
qmr	$7.7313\,e-2$ $(2.8\,e-3)$	$2.6850\,e-1$ $(8.8\,e-3)$	$1.3832\,e-1$ $(9.9\,e-4)$
symmlq	N/A[39]	$1.3090\,e-1$ $(2.7\,e-3)$[42]	N/A[39]
lsqr	$1.7698\,e-1$ $(2.4\,e-3)$	$2.6442\,e-1$ $(1.9\,e-3)$	$4.3903\,e-1$ $(5.5\,e-3)$
minres	N/A[39]	$1.3306\,e-1$ $(2.5\,e-3)$[41]	N/A[39]
cgs	$5.7569\,e-2$ $(1.1\,e-2)$[40]	$8.8294\,e-2$ $(8.7\,e-4)$	$5.9194\,e-2$ $(6.1\,e-4)$
gmres	$1.8550\,e-1$ $(9.0\,e-3)$	$1.6779\,e-1$ $(2.0\,e-3)$	$1.3474\,e-1$ $(9.9\,e-4)$
bicg	$5.2281\,e-2$ $(1.3\,e-3)$	$1.6642\,e-1$ $(2.0\,e-3)$	$1.3502\,e-1$ $(2.0\,e-3)$
bicgstab	$4.5792\,e-2$ $(1.5\,e-3)$	$9.2890\,e-2$ $(2.3\,e-3)$	$6.1175\,e-2$ $(5.8\,e-4)$

Table B.43: Mean running times and standard errors for `sticky`.

[33] 81% of cases finished in state 0 and 19% in state 1
[34] 78% of cases finished in state 0 and 22% in state 1
[35] 79% of cases finished in state 0 and 21% in state 1
[36] 29% of cases finished in state 0 and 71% in state 1
[37] 90% of cases finished in state 0 and 10% in state 1
[38] 56% of cases finished in state 0 and 44% in state 2

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[39]	$6.7551\,e+0$ $(1.1\,e-1)$[41]	N/A[39]
qmr	$9.9700\,e+0$ $(3.3\,e-2)$	$5\,e+0$ (0)	$7.1000\,e+0$ $(3.0\,e-2)$
symmlq	N/A[39]	$5.6907\,e+0$ $(9.4\,e-2)$[42]	N/A[39]
lsqr	$2.9510\,e+1$ $(3.8\,e-1)$	$7.0600\,e+0$ $(2.4\,e-2)$	$2.3180\,e+1$ $(3.0\,e-1)$
minres	N/A[39]	$5.7653\,e+0$ $(1.2\,e-1)$[41]	N/A[39]
cgs	$1.2409\,e+1$ $(2.5\,e+0)$[40]	$3.0100\,e+0$ $(1.0\,e-4)$	$4.0800\,e+0$ $(2.7\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$1.0090\,e+1$ $(3.8\,e-2)$	$5\,e+0$ (0)	$7.1000\,e+0$ $(3.0\,e-2)$
bicgstab	$5.3700\,e+0$ $(4.5\,e-2)$	$2.5050\,e+0$ $(5.0\,e-3)$	$3.6700\,e+0$ $(2.4\,e-2)$

Table B.44: Mean iteration counts and standard errors for `sticky`.

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[39]	$2.7876\,e-11$ $(2.5\,e-12)$[41]	N/A[39]
qmr	$2.5271\,e-11$ $(2.4\,e-12)$	$1.2495\,e-11$ $(3.5\,e-13)$	$2.6456\,e-11$ $(1.4\,e-12)$
symmlq	N/A[39]	$2.9181\,e-11$ $(2.7\,e-12)$[42]	N/A[39]
lsqr	$6.3856\,e-11$ $(2.0\,e-12)$	$1.4328\,e-11$ $(1.5\,e-12)$	$5.5635\,e-11$ $(2.0\,e-12)$
minres	N/A[39]	$2.9271\,e-11$ $(2.7\,e-12)$[41]	N/A[39]
cgs	$3.4865\,e-11$ $(5.6\,e-12)$[40]	$2.1172\,e-12$ $(4.4\,e-13)$	$1.0306\,e-11$ $(1.8\,e-12)$
gmres	$7.1471\,e-11$ $(2.0\,e-13)$	$4.6139\,e-12$ $(8.2\,e-14)$	$1.8572\,e-11$ $(5.6\,e-13)$
bicg	$2.2073\,e-11$ $(2.1\,e-12)$	$1.2579\,e-11$ $(3.6\,e-13)$	$2.6503\,e-11$ $(1.4\,e-12)$
bicgstab	$4.2499\,e-11$ $(2.7\,e-12)$	$1.3102\,e-11$ $(7.0\,e-13)$	$3.6414\,e-11$ $(2.6\,e-12)$

Table B.45: Mean relative residuals and standard errors for `sticky`.

B.2.16 Stickiness Unsymmetric

Matrices generated by `sticky(10000)`.

	Running time		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[43]	$1.6490\,e-1$ $(4.7\,e-3)$[45]	N/A[43]
qmr	$9.6474\,e-2$ $(3.2\,e-3)$	$2.4333\,e-1$ $(3.9\,e-3)$	$1.7531\,e-1$ $(1.9\,e-3)$
symmlq	N/A[43]	$1.5478\,e-1$ $(2.8\,e-3)$[46]	N/A[43]
lsqr	$2.3375\,e-1$ $(3.0\,e-3)$	$3.3121\,e-1$ $(2.4\,e-3)$	$5.1011\,e-1$ $(5.5\,e-3)$
minres	N/A[43]	$1.6193\,e-1$ $(4.1\,e-3)$[47]	N/A[43]
cgs	$1.0168\,e-1$ $(1.5\,e-2)$[44]	$1.0680\,e-1$ $(1.3\,e-3)$	$7.2610\,e-2$ $(8.6566\,e-4)$
gmres	$4.3444\,e-1$ $(2.2\,e-1)$	$1.9061\,e-1$ $(2.6\,e-3)$	$1.4373\,e-1$ $(9.3\,e-4)$
bicg	$6.8821\,e-2$ $(1.9\,e-3)$	$1.9737\,e-1$ $(1.3\,e-3)$	$1.5328\,e-1$ $(9.1\,e-4)$
bicgstab	$4.9196\,e-2$ $(9.9\,e-4)$	$9.4918\,e-2$ $(6.3\,e-4)$	$7.3379\,e-2$ $(6.6\,e-4)$

Table B.46: Mean running times and standard errors for unsymmetric `sticky`.

	Iteration Count		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[43]	$7.3636\,e+0$ $(1.9\,e-1)$[45]	N/A[43]
qmr	$9.7400\,e+0$ $(5.0\,e-2)$	$5\,e+0$ (0)	$7.1400\,e+0$ $(3.5\,e-2)$
symmlq	N/A[43]	$6.1042\,e+0$ $(9.4\,e-2)$[46]	N/A[43]
lsqr	$2.5640\,e+1$ $(2.7\,e-1)$	$6.9800\,e+0$ $(2.0\,e-2)$	$2.0410\,e+1$ $(2.1\,e-1)$
minres	N/A[43]	$6.1753\,e+0$ $(1.2\,e-1)$[47]	N/A[43]
cgs	$1.4545\,e+1$ $(2.6\,e+0)$[44]	$3.0100\,e+0$ $(1.0\,e-2)$	$4.0300\,e+0$ $(1.7\,e-2)$
gmres	$1\,e+0$ (0)	$1\,e+0$ (0)	$1\,e+0$ (0)
bicg	$9.8500\,e+0$ $(4.8\,e-2)$	$5\,e+0$ (0)	$7.1600\,e+0$ $(3.7\,e-2)$
bicgstab	$5.1400\,e+0$ $(2.8\,e-2)$	$2.5000\,e+0$ (0)	$3.5550\,e+0$ $(1.6\,e-2)$

Table B.47: Mean iteration counts and standard errors for unsymmetric `sticky`.

[39] All cases finished in state 1

[40] 22% of cases finished in state 0 and 78% in state 1

[41] 98% of cases finished in state 0 and 2% in state 1

[42] 97% of cases finished in state 0 and 3% in state 1

	Residual		
Function	No preconditioning	LU preconditioner	Cholesky preconditioner
pcg	N/A[43]	$3.6480\,e-11$ ($2.6\,e-12$)[45]	N/A[43]
qmr	$3.9131\,e-11$ ($3.3\,e-12$)	$5.3275\,e-12$ ($2.0\,e-13$)	$3.5138\,e-11$ ($2.3\,e-12$)
symmlq	N/A[43]	$3.6728\,e-11$ ($2.7\,e-12$)[46]	N/A[43]
lsqr	$6.0827\,e-11$ ($2.0\,e-12$)	$1.2183\,e-11$ ($2.1\,e-12$)	$5.3349\,e-11$ ($2.2\,e-12$)
minres	N/A[43]	$3.7040\,e-11$ ($2.7\,e-12$)[47]	N/A[43]
cgs	$5.4190\,e-11$ ($5.8\,e-12$)[44]	$1.7899\,e-12$ ($8.3\,e-13$)	$7.1965\,e-12$ ($9.6\,e-13$)
gmres	$6.3033\,e-11$ ($3.4\,e-13$)	$1.7253\,e-12$ ($4.8\,e-14$)	$2.3459\,e-11$ ($6.1\,e-13$)
bicg	$3.8281\,e-11$ ($3.1\,e-12$)	$5.3558\,e-12$ ($2.0\,e-13$)	$3.3177\,e-11$ ($2.1\,e-12$)
bicgstab	$4.5445\,e-11$ ($2.6\,e-12$)	$6.0998\,e-12$ ($1.0\,e-12$)	$5.6031\,e-11$ ($2.2\,e-12$)

Table B.48: Mean relative residuals and standard errors for unsymmetric `sticky`.

[43]All cases finished in state 1
[44]33% of cases finished in state 0 and 67% in state 1
[45]99% of cases finished in state 0 and 1% in state 1
[46]96% of cases finished in state 0 and 4% in state 1
[47]97% of cases finished in state 0 and 3% in state 1

Appendix C

SVD Algorithm Code

We present here a MATLAB code that demonstrates the algorithm described in Chapter 6. In practice the code may be extended to produce numerous subgraphs corresponding to different singular vector pairs for a single run.

```
 function [Akeep, Anew] = svdsort(A,Anames)

%%%%%%%%%%%%%%%%%%%%%%% Verify correct input %%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin == 0
    disp('Input an n by n matrix and a list of n names');
end

if nargin > 2
    disp('Too many input arguments');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SVD on A %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[U,S,V] = svd(A);

sortvec = input('Singular vector to sort on: ');
[usort,uind] = sort(U(:,sortvec));
[vsort,vind] = sort(V(:,sortvec));

%   Plot original adjacency matrix, reordered matrix and sorted singular
%   vectors.

spy(A(uind,vind));
title('A reordered');

figure;
plot(usort,'r*');
title('u sorted');
set(gca,'XTick',1:1:length(uind))
set(gca,'XTickLabel',uind)
```

```
figure;
plot(vsort,'r*');
title('v sorted');
set(gca,'XTick',1:1:length(vind))
set(gca,'XTickLabel',vind)

%%%%%%%%%%%%%%%%%%%%% Get cutoff values for vectors %%%%%%%%%%%%%%%%%%%%%

utop = input('Upper bound for u');
ubottom = input('Lower bound for u');
vtop = input('Upper bound for v');
vbottom = input('Lower bound for v');

ukeep1 = uind(find(usort>=ubottom));
vkeep1 = vind(find(vsort>=vbottom));
ukeep2 = uind(find(usort<=utop));
vkeep2 = vind(find(vsort<=vtop));

ukeep = intersect(ukeep1,ukeep2);
vkeep = intersect(vkeep1,vkeep2);

%%%%%%%%%% Get list of nodes to retain and eliminate duplicates %%%%%%%%%

Akeep = sort(cat(1,ukeep,vkeep));
for i = 2:length(Akeep)
    if Akeep(i) == Akeep(i-1)
        Akeep(i) = -1;
    end
end

%%%%%%%% Form submatrix by retaining only nodes listed in Akeep %%%%%%%%%

remove = find(Akeep==-1);
Akeep(remove) = [];
remove = [1:length(A)]';
remove(Akeep) = [];

Anew = A;
Anew(remove,:) = [];
Anew(:,remove) = [];

figure;
spy(Anew);

%%%%%%%%%%%%%%%%% Perform SVD on submatrix and sort %%%%%%%%%%%%%%%%%%%%

[Usub,Ssub,Vsub] = svd(Anew);
[usubsort,usubind] = sort(Usub(:,1));
Akeep = Akeep(usubind);

if nargin == 2
    names = Anames(Akeep);
    figure;
    spy(Anew(usubind,usubind));
    set(gca,'XTick',1:1:length(Akeep))
    set(gca,'XTickLabel',names)
    set(gca,'YTick',1:1:length(Akeep))
    set(gca,'YTickLabel',names)
else
    figure;
    spy(Anew(usubind,usubind));
    set(gca,'XTick',1:1:length(Akeep))
    set(gca,'XTickLabel',Akeep)
    set(gca,'YTick',1:1:length(Akeep))
    set(gca,'YTickLabel',Akeep)
end
title('Submatrix');
```

Appendix D

Matrix Mapping Code

We present here MATLAB code that performs the matrix mapping described in Chapter 8 for a given input adjacency matrix A. Embedded in the code are a number of smaller programs that are called to compute $f(A) + f(A^T)$, reorder the mapped matrix, produce an improved heat map and compute the bipartivity measure from Section 8.5.

```
 function bptest(A)

F = biweight(A);
k = input('Eigenvector to sort on? ');
[F,a] = specord(F,k);

% pcolor plot of F(A)
figure(1)
spy(A);
title('Adjacency matrix');
figure(2)
mypcolor(F);
title('Mapped matrix');

n = length(A);
group1 = input('How many nodes in first group? ');
%i.e. take nodes 1-group1 as first set
group2 = input('How many nodes in second group? ');
%i.e. take the last 'group2' nodes as second set
aind = [a(1:group1);a(end-group2+1:end)];

figure(3)
spy(A(aind,aind));
title('Subgraph');
Abip = bicount(A(aind,aind),group1,group2)

%------------------------------------------------------------------------
```

```
% Calculates f(A) + f(A')
function F = biweight(A);

[U,S,V] = svd(A);
B = U*diag(diag(cosh(S)))*U' - U*diag(diag(sinh(S)))*V';

[U,S,V] = svd(A');
BT = U*diag(diag(cosh(S)))*U' - U*diag(diag(sinh(S)))*V';

F = B + BT;
%------------------------------------------------------------------------
% Reorders A based on the k'th eigenvector
function [S,b] = specord(A,k);

[V,D] = eig(A);
[a,b] = sort(V(:,k));
S = A(b,b);
%------------------------------------------------------------------------
% pcolor that doesn't discard a row and column
function B = mypcolor(A);

n = length(A);
B = zeros(n+1,n+1);
for i = 1:n
    B(i,1:n) = A(n-i+1,:);
end
pcolor(B);
%------------------------------------------------------------------------
function count = bicount(A,front,back);

% Weigh number of nonzeros in top right against number in bottom left

n = length(A);
topright = (nnz(A(1:front,front+1:n)))/(front*back);
bottomleft = (nnz(A(front+1:n,1:front)))/(front*back);
if bottomleft > topright
    A = A';
end
topright = (nnz(A(1:front,front+1:n)))/(front*back);
rest = (nnz(A(1:front,1:front)) + nnz(A(front+1:n,front+1:n))
+ nnz(A(front+1:n,1:front)))/(n^2 - front*back)+1;
if rest == 0
    rest = 1/2*(n^2 - front*back); %avoids division by zero
end
count = topright/rest;
%------------------------------------------------------------------------
```

Bibliography

[1] J. Abello, A. Buchsbaum, and J. Westbrook, *A functional approach to external graph algorithms*, Lecture Notes in Computer Science, 1461 (1998), pp. 332–343.

[2] R. Albert and A. L. Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics, 74 (2002), pp. 47–97.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongorra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, PA, third ed., 1999.

[4] A. L. Barabási and R. Albert, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512.

[5] A. L. Barabási and E. Bonabeau, *Scale-free networks*, Scientific American, 288 (2003), pp. 60–69.

[6] D. Barash, *Second eigenvalue of the laplacian matrix for predicting RNA conformational switch by mutation*, Bioinformatics, 20 (2004), pp. 1861–1869.

[7] V. Batagelj and U. Brandes, *Efficient generation of large random networks*, Physical Review E, 71 (2005).

[8] P. S. Bearman, J. Moody, and K. Stovel, *Chains of affection: The structure of adolescent romantic and sexual networks*, American Journal of Sociology, 110 (2004), pp. 44–91.

[9] V. Boginski, S. Butenko, and P. M. Pardalos, *On structural properties of the market graph*, in Innovations in Financial and Economic Networks, A. Nagurney, ed., Edward Elgar Publishers, 2003, pp. 29–45.

[10] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra, *Matrix market: a web resource for test matrix collections*, in The Quality of Numerical Software: Assessment and Enhancement, R. Boisvert, ed., Chapman and Hall, London, 1997, pp. 125–137.

[11] B. Bollobás, *Random Graphs*, Academic Press, London, 1985.

[12] B. Bollobás, R. O., S. J., and G. Tusnády, *The degree sequence of a scale-free random graph process*, Random Structures and Algorithms, 18 (2001), pp. 279–290.

[13] J.-P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov, *Metagenes and molecular pattern discovery using matrix factorization*, PNAS, 101 (2004), pp. 4164–4169.

[14] N. Chaterjee and S. Sinha, *Understanding the mind of a worm: hierarchical network structure underlying nervous system function in C. elegans*, Progress in Brain Research, 168 (2008), pp. 145–153.

[15] F. CHUNG, *Spectral graph theory*, American Mathematical Society, Providence, RI, 1997.

[16] M. J. CONYON AND M. R. MULDOON, *The small world of corporate boards*, Journal of Business Finance and Accounting, 33 (2006), pp. 1321–1343.

[17] T. COUR, F. BENEZIT, AND J. SHI, *Spectral segmentation with multiscale graph decomposition*, in IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2 (2005), pp. 1124–1131.

[18] D. P. CROFT, J. KRAUSE, AND R. JAMES, *Social networks in the guppy (poecilia reticulata)*, Proceedings of the Royal Society of London Series B - Biological Sciences, 271 (2004), pp. S516–S519.

[19] T. DAVIS, *The university of florida sparse matrix collection*, Tech. Rep. CISE Department, REP-2007-298 (2007).

[20] E. DE SILVA AND M. P. H. STUMPF, *Complex networks and simple models in biology*, J. Royal Society Interface, 2 (2005), pp. 419–430.

[21] E. DE SILVA, T. THORNE, P. INGRAM, A. AGRAFIOT, J. SWIRE, C. WIUF, AND M. P. H. STUMPF, *The effects of incomplete protein interaction data on structural and evolutionary inferences*, BMC Biology, 4 (2006), pp. 39+.

[22] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[23] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Soft, 15 (1989), pp. 1–14.

[24] R. M. Durbin, *Studies on the Development and Organisation of the Nervous System of Caenorhabditis Elegans*, PhD thesis, University of Cambridge, 1987.

[25] P. Erdös and A. Rényi, *On random graphs*, Publ. Math. Debrecen, 6 (1959), pp. 290–297.

[26] E. Estrada, *Protein bipartivity and essentiality in the yeast protein-protein interaction network*, J. Proteome Res., 5 (2006), pp. 2177–2184.

[27] ——, *Topological structural classes of complex networks*, Physical Review E, 75 (2007).

[28] E. Estrada and N. Hatano, *Communicability in complex networks*, Physical Review E, 77 (2008).

[29] E. Estrada, D. J. Higham, and N. Hatano, *Communicability and multipartite structures in complex networks at negative absolute temperatures*, Physical Review E, 78 (2008).

[30] E. Estrada and J. Rodríguez-Velázquez, *Spectral measures of bipartivity in complex networks*, Physical Review E, 72 (2005).

[31] L. Euler, *Solutio problematis ad geometriam situs pertinentis*, Commentarii academiae scientiarum imperialis Petropolitanae, 8 (1740), pp. 128–140.

[32] W. J. Ewens and G. R. Grant, *Statistical Methods in Bioinformatics: An Introduction*, Springer, Berlin, 2001.

[33] M. FALOUTSOS, P. FALOUTSOS, AND C. FALOUTSOS, *On power-law relationships of the internet topology*, Computer Communications Review, 29 (1999), pp. 251–262.

[34] B. FINGLETON AND M. FISCHER, *Neoclassical theory versus new economic geography. competing explanations of cross-regional variation in economic development*, Annals of Regional Science, (2010). to appear.

[35] E. N. GILBERT, *Random graphs*, Ann. Math. Statist., 30 (1959), pp. 1141–1144.

[36] J. R. GILBERT, C. MOLER, AND R. SCHREIBER, *Sparse matrices in matlab: design and implementation*, SIAM J. Matrix Analysis Applications, 13 (1992), pp. 333–356.

[37] P. GLASSERMAN, *Monte Carlo Methods in Financial Engineering*, Springer, Berlin, 2004.

[38] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, third ed., 1996.

[39] G. GRIMMETT, *Percolation*, Springer, second ed., 1999.

[40] P. GRINDROD, *Range-dependent random graphs and their application to modelling large small-world proteome datasets*, Physical Review E, 66 (2002).

[41] P. GRINDROD, D. J. HIGHAM, AND G. KALNA, *Periodic reordering*, IMA J. Numer. Anal. To appear.

[42] P. Grindrod and M. Kibble, *Review of uses of network and graph theory concepts within proteomics*, Expert Review of Proteomics, 1 (2004), pp. 229–238.

[43] J. D. H. Han, D. Dupuy, N. Bertin, M. E. Cusick, and M. Vidal, *Effect of sampling on topology predictions of protein-protein interaction networks*, Nature Biotechnology, 23 (2005), pp. 839–844.

[44] B. Hendrickson and R. Leland, *The chaco user's guide: Version 2.0*, Tech. Rep. SAND94-2692, (1994).

[45] D. J. Higham, *Unravelling small world networks*, J. Comp. Appl. Maths., 158 (2003), pp. 61–74.

[46] ——, *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*, Cambridge University Press, 2004.

[47] ——, *Spectral reordering of a range-dependent weighted random graph*, IMA J. Numer. Anal., 25 (2005), pp. 443–457.

[48] D. J. Higham and N. J. Higham, *MATLAB Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[49] D. J. Higham, G. Kalna, and M. Kibble, *Spectral clustering and its use in bioinformatics*, J. Computational and Applied Math., 204 (2007), pp. 25–37.

[50] D. J. Higham, G. Kalna, and J. K. Vass, *Spectral analysis of two-signed microarray expression data*, IMA Mathematical Medicine and Biology, 24 (2007), pp. 131–148.

[51] D. J. HIGHAM, N. PRZULJ, AND M. RASAJSKI, *Fitting a geometric graph to a protein-protein interaction network*, Bioinformatics, 24 (2008), pp. 1093–1099.

[52] P. HOLME, F. LILJEROS, C. R. EDLING, AND B. J. KIM, *Network bipartivity*, Physical Review E, 68 (2003).

[53] Y. HU AND J. A. SCOTT, *HSL_MC73: A fast multilevel Fiedler and profile reduction code*, RAL-TR-2003-36, (2003).

[54] T. ITO, T. CHIBA, R. OZAWA, M. YOSHIDA, M. HATTORI, AND Y. SAKAKI, *A comprehensive two-hybrid analysis to explore the yeast protein interaction interactome*, Proc. Natl. Acad. Sci., 98(8) (2001), pp. 4569–4574.

[55] S. A. KAUFFMAN, *Metabolic stability and epigenesis in randomly constructed genetic nets*, J. of Theor. Biol., 22 (1969), pp. 437–467.

[56] I. Z. KISS, D. M. GREEN, AND R. R. KAO, *The network of sheep movements within great britain: network properties and their implications for infectious disease spread*, J. Roy. Soc. Interface, 3 (2006), pp. 669–677.

[57] J. M. KLEINBERG, *Navigation in a small world*, Nature, 406 (2000), p. 845.

[58] A. N. LANGVILLE AND C. D. MEYER, *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton, 2006.

[59] J. LINK, *Does food web theory work for marine ecosystems?*, Marine Ecology Press Series, 230 (2002), pp. 1–9.

[60] L. Lovász, *Random walks on graphs*, in Paul Erdös is Eighty, D. Miklós, V. T. Sós, and T. Szönyi, eds., János Bolyai Mathematical Society, Budapest, 1996, pp. 353–398.

[61] S. Maslov and K. Sneppen, *Specificity and stability in topology of protein networks*, Science, 296 (2002), pp. 910–913.

[62] T. Milenkovic, J. Lai, and N. Przulj, *Graphcrunch: A tool for large network analyses*, BMC Bioinformatics, 9:70 (2008).

[63] S. Milgram, *The small world problem*, Psychology Today, 2 (1967), pp. 60–67.

[64] S. Morita, K. Oshio, Y. Osana, Y. Funabashi, K. Oka, and K. Kawamura, *Geometrical structure of the neuronal network of Caenorhabditis elegans*, Physica A, 298 (2001), pp. 553–561.

[65] J. L. Morrison, R. Breitling, D. J. Higham, and D. R. Gilbert, *Generank: Using search engine technology for the analysis of microarray experiments*, BMC Bioinformatics, c (2005), p. 6:233.

[66] ——, *A lock-and-key model for protein-protein interactions*, Bioinformatics, 2 (2006), pp. 2012–2019.

[67] C. Needham, *The Internet Movie Database (IMDb)*, 1990-2009. `http://www.imdb.com/`.

[68] M. E. J. Newman and M. Girvan, *Finding and evaluating community structure in networks*, Physical Review E, 69 (2004).

[69] M. E. J. Newman, C. Moore, and D. J. Watts, *Mean-field solution of the small-world network model*, Phys. Rev. Lett., 84 (2000), pp. 3201–3204.

[70] J. R. Norris, *Markov Chains*, Cambridge University Press, 1997.

[71] R. N. Onody and P. A. De Castro, *Complex network study of brazilian soccer players*, Phys. Rev. E, 70 (2004).

[72] L. Page, S. Brin, R. Motwani, and T. Winograd, *The pagerank citation ranking: Bringing order to the web*, tech. rep., (1998).

[73] M. Penrose, *Geometric Random Graphs*, Oxford University Press, 2003.

[74] M. A. Porter, P. J. Mucha, M. E. J. Newman, and C. M. Warmbrand, *A network analysis of committees in the united states house of representatives*, Proc. Nat. Acad. Sci., 102 (2005), pp. 7057–7062.

[75] N. Przulj, D. G. Corneil, and I. Jurisica, *Modelling interactome: Scale-free or geometric?*, Bioinformatics, 20 (2004), pp. 3508–3515.

[76] N. Przulj and D. J. Higham, *Modelling protein-protein interaction networks via a stickiness index*, J. Royal Society Interface, 3 (2006), pp. 711–716.

[77] F. Roethlisberger and W. Dickson, *Management and the Worker: an Account of a Research Program Conducted by the Western Electric Company*, Harvard University Press, Cambridge, 1939.

[78] S. Rogers, R. E. Sceltema, M. Girolami, and R. Breitling, *Probabilistic assignment of formulas to mass peaks in metabolomics experiments*, Bioinformatics, 25(4) (2009), pp. 512–518.

[79] M. Salathé, R. M. May, and S. Bonhoeffer, *The evolution of network topology by selective removal*, J. R. Soc Interface, 2 (2005), pp. 533–536.

[80] A. SPENCE, Z. STOYANOV, AND J. K. VASS, *The sensitivity of spectral clustering applied to gene expression data*, Proceedings of the 1st International Conference on Bioinformatics and Biomedical Engineering, (2007), pp. 1343–1346.

[81] M. P. H. STUMPF, C. WIUF, AND R. M. MAY, *Subnets of scale-free networks are not scale-free: Sampling properties of networks*, Proc. Nat. Acad. Sci., 102 (2005), pp. 4221–4224.

[82] A. TAYLOR AND D. J. HIGHAM, *Contest: A controllable test matrix toolbox for matlab*, ACM Trans. Math. Software, 35 (2009), pp. 1–17.

[83] A. THOMAS, R. CANNINGS, N. A. M. MONK, AND C. CANNINGS, *On the structure of protein-protein interaction networks*, Biochemical Society Transactions, 31 (2003), pp. 1491–1496.

[84] B. TITZ, M. SCHLESNER, AND P. UETZ, *What do we learn from high-throughput protein interaction data?*, Expert Review of Proteomics, 1 (2004), pp. 111–121.

[85] A. H. Y. T. TONG, G. LESAGE, G. D. BADER, H. DING, H. XU, X. XIN, J. YOUNG, G. F. BERRIZ, R. L. BROST, M. CHANG, Y. Q. CHEN, X. CHENG, G. CHUA, H. FRIESEN, D. S. GOLDBERG, J. HAYNES, C. HUMPHRIES, G. HE, S. HUSSEIN, L. KE, N. KROGAN, Z. LI, H. LEVINSON, J. N. LU, P. MÉNARD, C. MUNYANA, A. B. PARSONS, O. RYAN, R. TONIKIAN, T. ROBERTS, A.-M. SDICU, J. SHAPIRO, B. SHEIKH, B. SUTER, S. L. WONG, L. V. ZHANG, H. ZHU, C. G. BURD, S. MUNRO, C. SANDER, J. RINE, J. GREENBLATT, M. PETER, A. BRETSCHER, G. BELL, F. P. ROTH, G. W. BROWN, B. ANDREWS,

H. Bussey, and C. Boone, *Global mapping of the yeast genetic interaction network*, Science, 303 (2004), pp. 808–813.

[86] P. Uetz, L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J. M. Rothberg, *A comprehensive analysis of protein-protein interactions in saccharomyces cerevisiae*, Nature, 403 (2000), pp. 623–627.

[87] P. J. Valk, R. G. Verhaak, M. A. Beijen, C. A. Erpelinck, S. B. van Waalwijk van Doorn-Khosrovani, J. M. Boer, H. B. Beverloo, M. J. Moorhouse, P. J. van der Spek, B. Lüwenberg, and R. Delwel, *Prognostically useful gene-expression profiles in acute myeloid leukemia*, New England J. Medicine, 16 (2004), pp. 1617–1628.

[88] R. Van Driessche and D. Roose, *An improved spectral bisection algorithm and its application to dynamic load balancing*, Parallel Computing, 21 (1995), pp. 29–48.

[89] J. K. Vass, D. J. Higham, X. Mao, and D. Crowther, *New controls of tca-cycle genes revealed in networks built by discretization or correlation*, Tech. Rep. Department of Mathematics, 10 (2009).

[90] D. J. Watts and S. H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442.

[91] M. Wheel. personal communication.

[92] R. J. Williams, E. L. Berlow, J. A. Dunne, A. L. Barabási, and N. D. Martinez, *Two degrees of separation in complex food webs*, Proc. Nat. Acad. Sci., 99 (2002), pp. 12913–12916.

[93] D. Wishart, *Whisky Classified: Choosing Single Malts by Flavour*, Pavilion Books, London, 2nd ed., 2006.

[94] I. Xenarios, L. Salwinski, X. J. Duan, P. Highney, S. Kim, and D. Eisenberg, *Dip, the database of interacting proteins: a research tool for studying cellular networks of protein interactions*, Nucleic Acids Research, 30 (2002), pp. 303–305.

[95] P. Yodzis, *Diffuse effects in food webs*, Ecology, 81 (2000), pp. 261–266.