MODELS FOR SPECTRAL CLUSTERING AND THEIR APPLICATIONS

by

Donald, F. McCuan

B.A., Austin College, 1972

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Applied Mathematics

2012

This thesis for the Doctor of Philosophy degree by

Donald, F. McCuan

has been approved

by

Andrew Knyazev, Advisor and Chair

Steven Billups

Tzu Phang

Julien Langou

Weldon Lodwick

Date _____

McCuan, Donald, F. (Ph.D., Applied Mathematics)

Models for Spectral Clustering and Their Applications

Thesis directed by Professor Andrew Knyazev

ABSTRACT

In this dissertation the concept of spectral clustering will be examined. We will start by discussing biclustering of images via spectral clustering and give a justification for this technique by analogy to vibrational problems that is independent of that given by relaxation of a combinatorial optimization problem. The importance in clustering of the Fiedler vector and some theorems by Fiedler are emphasized. We will extend Fiedlers theorem to the case of the generalized eigenvalue problem.

By examining the application of these theories to the clustering problem we hope to develop a better understanding of the eigenfunctions and their use in clustering. Practical problems with clustering that occur due to construction of edge weights are studied.

Courants Nodal Domains Theorem (CNLT) as an analog of the Fiedler vector for eigenvectors of higher dimension are studied and the literature for discrete CNLTs are reviewed. A new definition for r-weak sign graphs is presented and a modified discrete CNLT theorem for r-weak sign graphs is introduced. The application of these to spectral clustering is discussed.

The discussion of spectral clustering is continued via an examination of clustering on DNA micro arrays. This allows us to develop an algorithm for successive biclustering. In this we develop a new technique and theorem for dealing with disconnected graph components. All of this is incorporated in new MATLAB software. Results of clustering using real micro array data is presented.

The last section deals with the software package Block Locally Optimal Precondi-

tioned Eigenvalue Xolver (BLOPEX) which as part of the Authors graduate work was upgraded to Version 1.1. BLOPEX implements the Locally Optimal BLock Preconditioned Conjugate Gradient (LOBPCG) method for solution of very large, sparse, symmetric (or Hermitian) generalized eigenvalue problems.

Version 1.1 of BLOPEX adds (amongst other things) support for complex matrices and 64bit integers. BLOPEX provides interfaces to independently developed software packages such as PETSc and Hypre which provide high quality preconditioners and parallel MPI-based processing support. There is also support for execution of BLOPEX from MATLAB and stand alone C programs. This was a multi person effort. The Authors contribution was in recoding the abstract routines and PETSc and Hypre interfaces for the new functionality, the development of a new complex driver test program, and aid and assistance in testing, debugging and documentation of all interfaces.

We will describe the BLOPEX software, design decisions, the LOBPCG algorithm as implemented, and all of the various interfaces. Some numerical results will be presented.

The form and content of this abstract are approved. I recommend its publication.

Approved: Andrew Knyazev

DEDICATION

I dedicate this thesis to my wife Carolyn, and the volunteers and staff of the Denver Museum of Natural History who have given me encouragement over the many years of this endeavor.

ACKNOWLEDGMENT

Foremost, I would like to to thank my advisor, Andrew Knyazev, for his support and motivation. Also, I thank my other Professors at UCD who renewed by interest in Mathematics, and raised my knowledge and understanding of the subject to new levels.

TABLE OF CONTENTS

FIGURES

Figure

TABLES

Table

1. Spectral Clustering and Image Segmentation

1.1 Introduction

We are concerned with the problem of partitioning an image into two parts (a bicluster) with the objective that the elements (image pixels) within each part are more similar to each other than elements between parts tend to be. This statement while simple has left undefined such concepts as similarity and measures of whether any particular bicluster is better than another.

There is a large literature which takes the approach of defining this problem as a biclustering of a weighted graph where the biclustering is performed by minimization of some vertex cut function; for example see [53],[56]. These problems can be expressed as a minimization under constraints of the Rayleigh-Ritz ratio of the associated graph Laplacian matrix.

This combinatorial problem is NP complete and to solve it the constraints are relaxed, leading to a problem of solving for an eigenvector of the second largest eigenvalue of the graph Laplacian, commonly referred to as the Fiedler Vector. Then this eigenvector is used to separate the graphs vertices into two groups. This is the technique of spectral clustering. We will discuss the graph Laplacian in Section 1.2 which is background for the rest of the paper.

Bichot [6] attributes the origin of spectral clustering to Donath and Hoffman [20] 1970. The concept is simple. Its complexity lies in understanding why it works.

Spectral clustering does not always give good solutions to the original combinatorial problem. We examine some of these issues in Section 1.3 and will present an alternative justification for spectral clustering in Section 1.5.

But, before this will give a brief overview of the literature in Section 1.4 which examines the field of combinatorial and spectral clustering.

Spectral clustering involves using the Fiedler vector to create a bipartition of the graph. Some theorems by Fiedler are needed to understand the character of the

Fiedler vector and how this relates to clustering. These theorems are reviewed in Section 1.6 and we expand the scope of one of these theorems in Section 1.7.

Having expanded the scope of the Theorems we examine numerical results for the generalized eigenvalue problem using mass matrices.

We then examine the problem of image segmentation in Section 1.9 and discuss generation of edge weights in Section 1.10.

We define our algorithm for image biclustering in Section 1.11. Then in Section 1.12 give examples of how weight parameters can effect clustering, connect this with the Fiedler theorems, and show how problems can occur.

This is followed in Section 1.13 by a discussion of eigenvalue solvers and the need for solvers for eigenpairs of large sparse matrices for the image segmentation problems.

Nodal sets and sign graph theory is reviewed in Section 1.14 where we derive an extension of the discrete nodal domain theorem that includes Fiedlers theorem as a special case.

We summarize in the last section.

1.2 Graph Laplacian

Let $G = (V, E)$ denote a graph where V is its vertex set, E is its edge set, and the number of vertices $|V| = n$. We number the vertices of G and this index is then used to represent the vertices V. The edges of G are undirected and have no loops.

The n x n adjacency matrix A of G has entries representing the edges E of G. If vertices i and j have an edge between them then the element a_{ij} of A has a real positive weight assigned to that edge and zero otherwise. The degree matrix D for G is a diagonal matrix where $d_{ii} = \sum_{j \in V} a_{ij}$.

Definition 1.1 *The unnormalized graph Laplacian is* $L = D - A$.

This is the discrete analog of the continuous Laplacian $\triangle = \sum_i \frac{\partial}{\partial x_i}$. A justification for this can be found in [4].

The discrete Laplacian has the following properties [56]:

- it is real symmetric,
- positive semi definite,
- its smallest eigenvalue is 0,
- $\mathbb{I} = (1, 1, \ldots, 1)^T$ is an eigenvector for eigenvalue 0, and
- the multiplicity of 0 is the number of connected components of the graph.

We note here the existence of normalized graph Laplacians, and will have more to say about them later.

Definition 1.2 *The symmetric normalized graph Laplacian is* $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

Definition 1.3 *The random walk graph Laplacian is* $L_{rw} = D^{-1} L$.

1.3 Combinatorial Model

One approach to clustering is to minimize some function that reflects how closely the clusters are connected. For example in Figure 1.1 vertex cuts are used to partition the graph into two subgraphs A and B where $|A| + |B| = |V|$. A cut is assigned a value $cut(A, B) = \sum_{i \in A, j \in B} a_{ij}$. This is unsatisfactory in many cases where subgraphs that are highly unbalanced are produced. For example Cut 1 in Figure 1.1 has the same cut value as that of Cut 2.

To overcome this the following ratios are often used [53],[56].

$$Ratiocut(A, B) = \frac{cut(A, B)}{|A|} + \frac{cut(A, B)}{|B|}$$

and

$$Ncut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(A, B)}{vol(B)} \quad \text{where} \quad vol(A) = \sum_{i \in A} d_{ii}$$

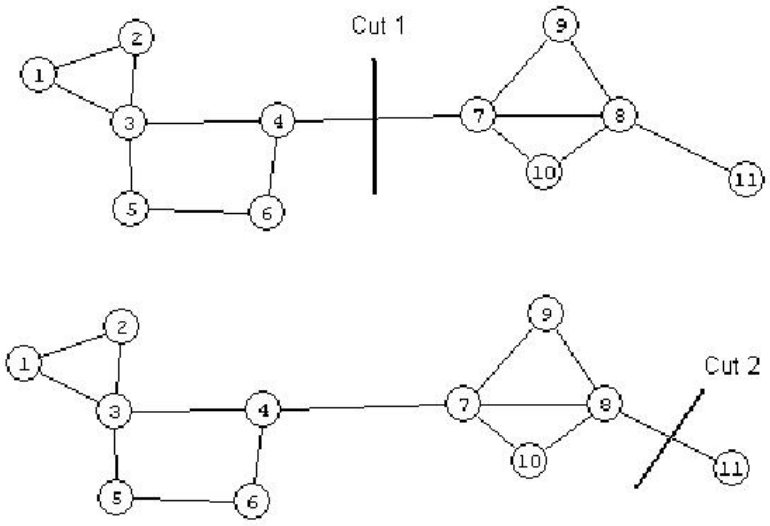


Figure 1.1: Partition of a Graph by Vertex Cuts. All edge weights are equal.

Assuming the graph is connected (more on this in Section 1.6), it can be shown [56] that minimizing the Ratiocut is equivalent to

$$\min_{\substack{A\subset V \\ f\perp\mathbb{1} \\ \|f\|=\sqrt{n}}} f'Lf \quad \text{where} \quad f_i = \begin{cases} |B|/|A| \text{ if } i \in A \\ -|A|/|B| \text{ if } i \in B \end{cases}$$

where we will refer to f as the combinatorial solution.

This problem is NP complete (see Appendix of [53]), but if the constraints are relaxed it becomes

$$\min_{\substack{f\in\mathbb{R}^n \\ f\perp\mathbb{1} \\ \|f\|=\sqrt{n}}} f'Lf$$

The term $f'Lf$ is the numerator of the Rayleigh-Ritz Ratio and the Rayleigh Ritz characterization of eigenvalues [37] gives the solution as the 2nd smallest eigenvalue of L with f as its eigenvector. An eigenvector of the 2nd smallest eigenvalue (which could have multiplicity greater than 1) is called the Fiedler vector.

A similar analysis can be performed for $Ncut$ [53].

A spectral clustering analysis then consists of finding the Fiedler vector and using it to produce the clusters A and B. Commonly, these are chosen as $A = \{i | f_i \geq 0\}$

and $B = \{i | f_i < 0\}$. This method is derived by analogy to the combinatorial solution of f.

There are some questions that can be raised at this point.

- How closely does the relaxed solution come to the actual solution?

Graphs can be constructed where the difference in cut values is arbitrarily large [56]. This might be expected since the original problem is NP complete, the relaxed problem is not, and we have placed no restrictions on the type of graphs involved.

- Even if we have a solution that is close to optimal, how balanced is it?

We make the following observation. The balancing term in *Ratiocut* is $\frac{1}{|A|} + \frac{1}{|B|}$. If we plot this for various values of $|A|$ and n (Figure 1.2) we see a large penalty near the extremes which is desirable but a very small penalty for values of $|A|$ between. So if we want a balance closer to $\frac{|A|}{n} = .5$ this can be achieved even when $|A| >> |B|$. This is particularly true when there are a large number of vertices which is usually the case for images.

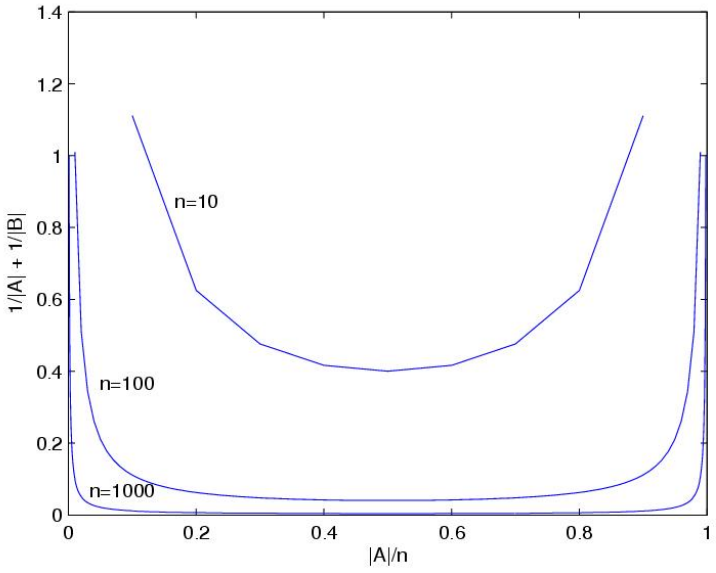


Figure 1.2: Ratiocut Penalty

- What should we do with vertices where $f_i = 0$? This situation is precluded in the original combinatorial solution for f.

The choice of putting vertex i where $f_i = 0$ into cluster A along with $f_i > 0$ is arbitrary. We could just as well have chosen to place it in cluster B.

- Are clusters connected?

We will see in Section 1.6 that one of the clusters will always be connected but the other may not be. This upsets our intuition about what a cluster should be. We expect the nodes within a cluster to have some similarity, but if the cluster itself is not connected how can this be. Also, this is a practical issue if we should do successive biclusters to break an image into multiple clusters. We want to start our algorithm with a connected graph and this might no longer be the case. Why this is the case will be explained in Section 1.6.

In an attempt to overcome these issues and force the spectral clustering results to give clusters with better Ratiocut values some algorithms resort to adhoc methods such as looking for a better Ratiocut around zero; for example $A = \{i | f_i \geq r\}$ and $B = \{i | f_i < r\}$ where $r \geq 0$. The result may still not be optimal and one of the clusters could still not be connected.

These kind of problems have led us to adopt an alternative justification for spectral clustering and an algorithm which reflects this.

1.4 Overview of Literature

The clustering literature, with connections to spectral clustering, is primarily concerned with the solution of a combinatorial problem. Since this problem turns out to be NP hard, its solution is attempted via relaxation to a spectral clustering problem.

The literature is generally concerned with one or more of the following issues:

- alternatives to spectral clustering,

- justification of spectral clustering,
- edge weight selection,
- how to use of eigenvectors to obtain clusters,
- how many clusters to construct, and
- applications.

An excellent starting point for the study of spectral clustering is the tutorial by Ulrike von Luxburg [56]. After that, possibly the most referenced paper on the subject is by Shi and Malik [53].

There are alternatives to spectral clustering. An algorithm for finding bipartitions with minimum cut weight is presented by Stoer and Wagner [55]. However, this algorithm does not address the problem of finding balanced partitions.

Some authors make use of several eigenvectors, collect these into a matrix, and partition based on the row vectors.

Alpert and Yao [2] construct a min-cut problem where cluster volumes must fall within predefined ranges. They then define a max-min k-way vector partitioning problem based on the eigenvector rows, and show that these problems are equivalent when all eigenvectors are used. Of course in this case, since min-cut is NP hard, their vector partitioning must also be NP hard. No justification, except for numerical experiments, is presented when not using all eigenvectors.

Alpert, Kahng, and Yao [1] define their MELO (multiple eigenvector linear orderings) algorithm, which uses multiple eigenvectors to produce a bicluster.

White and Smyth [58] define a new cost function, referred to as the Q function, which is a measure of the deviation between the probability that both ends of an edge lie in the same cluster, and the square of the probability that either end of an edge lies in the cluster. Minimization of the Q function is a problem whose relaxation

results in (approximately) L_{rw}, the random walk Laplacian. K-means clustering is then applied to the rows of some of the eigenvectors of L_{rw}. The best clustering is computed by varying K and computing Q for each partition produced.

Another probabilistic approach is done by Meila and Shi [46] [47]. They define a cost function via a random walk and connect this to the NCUT balanced cost function. They then present a machine learning algorithm to determine edge weights, trained by predefined image clusters.

While Meila and Shi's random walk development is fairly intuitive, a more obscure attempt at justifying spectral clustering via the random walk Laplacian, is done by Nadler, Lafon, Coifman, and Kevrekidis [48]. This is based on an assumption that the graph vertices are distributed according to the probability distribution, $p(x) = e^{-U(x)}$ where $U(x)$ is a potential.

Another discrete cost function is presented in the paper by Kanna, Vempala, and Vetta [40]. They define an (α,ϵ) bi-criteria, relax the minimization problem, and then show that spectral clustering has a worst case approximation guarantee with respect to the (α,ϵ) measure. While interesting, it does not seem to be that practical.

Other attempts to place bounds on the cluster results with respect to a cost function date back to Donath and Hoffman [21]. They set a lower bound on the sum of the edge cuts, where the number of clusters, and the number of vertices in the clusters was preselected. So, again, not that practical.

Ng, Jordan, and Weiss [49] use a typical approach to spectral clustering; i.e. the symmetric Laplacian and k-means clustering applied to multiple eigenvectors. They then use the Cheeger constant [11] of the clusters to construct criteria assumptions. If these assumptions are met, they show that there exists orthogonal vectors in k space such that rows of the eigenvectors are "close" to them.

Sharon, Galun, Sharon, Basri, and Brandt [52] apply spectral clustering to image segmentation. They present a multigrid approach (see also Bichot [6]) where spectral

clustering (Ncut cost function) plays the part of bi-clustering at each coarseness level.

Dhillon, Guan, and Kulis [17] [16] give another multigrid approach using the weighted Kernal k-means algorithm. This projects the data vectors onto a higher dimension space where k-means does the clustering for the refinement steps. Spectral clustering is applied at the coarsest level. They apply this technique to image segmentation and gene networks.

A technique that is not spectral based, is presented by Felzenszwalb and Uttenlocher [24]. This is applied to image segmentation, and while not spectral in nature, has an interesting approach for coarsening of the image. They define a concept of a segmentation being too fine or too coarse. A lumping procedure is then applied to lump vertices into segments that are neither too fine or too coarse.

Orponen and Schaeffer [50], while accepting the spectral clustering method in principle, seek to avoid the solution of the eigenvalue problem, by construction of an approximate Fiedler vector. This is done by minimization of the Rayleigh Quotient of a reduced Dirchlet matrix via gradient descent.

The issue of round off error in computation of the Fiedler vector via Krylov subspaces is taken up by Matsekh, Skurikhin, and Rosten [45]. They observe that round off error can make the Fiedler vector unsuitable for image segmentation. We address this issue later this thesis.

Clustering the Fiedler vector is analyzed by Chan, Ciarlet, and Szeto [7]. Their issue is, what is the best way to split the Fiedler vector? They show that a median cut on the second eigenvector is optimal, in the sense that the partition vector induced by this is the closest partition vector to the second eigenvector. What they actually prove, is that the median cut on any vector is optimal, to that vector. So, it's not really a justification for using the median cut for segmentation.

Ding, He, and Simon [18] show the equivalence of symmetric Nonnegative Matrix Factorization (NMF) and Kernal k-means, and then the equivalence of Ncut and

NMF. The definition of Ncut they use is not the same as that defined by Shi and Malik. The NMF of a $p \times n$ matrix X is $X \approx FG^T$ when F is $p \times k$ and G is $k \times n$.

Another clustering approach, that is entirely different, is presented by Grady and Schwartz [32]. They start with the RatioCut problem and convert to an isoperimetric problem which can be solved as a solution to a linear system.

A very interesting paper comes from Zelnik-Manor and Perona [59]. They use the inverse Gausian to compute weights, but associate a different scaling factor with each vertex. The scaling factor reflects the density of vertices at each vertex. This allows for handling of multi-scale data and background clutter. They also propose a cost function for automatic determination of the number of clusters.

Similarly, Fischer and Poland [28], also adjust the scaling factor for each vertex. But, their technique for doing so is different. They also propose use of k-lines clustering of the eigenvector rows instead of k-means.

We deal with image segmentation and micro-array analysis in this paper. Another application that is only starting to receive some attention is the application of spectral clustering to Phylogenetics [10] [8] [60]. Here spectral clustering has been proposed as an alternative to maximum parsimony, maximum likelihood, and neighbor-joining. The data, so far, has been gene sequences. The techniques are mostly fairly standard: symmetric laplacian, multiple eigenvectors, and k-means.

Other papers are mentioned elsewhere in the thesis and are not repeated here.

1.5 Vibrational Model

We first consider a problem with an infinite number of points. Suppose we have a membrane which is continuous. It is stretched over some geometric shape Ω which is connected but possibly not convex; such as a circle, square, L shape, etc. Vibrations on the membrane are described by the wave equation $-\rho\ddot{u} + \triangle u = 0$ in Ω [15],[23] with natural (free) boundary conditions. Here u is the displacement of the membrane, and $\rho = \frac{1}{v^2}$ where v is the speed of propagation of the wave.

Suppose we only have transversal vibration without friction. We assume standing wave solutions of the form $u(x, y, t) = U(x, y)e^{i\omega t}$. The problem then becomes a generalized eigenvalue problem of the form $-\triangle U = \omega^2 \rho U$.

The smallest vibrational frequency is zero, which corresponds to the first vibration mode, which is a constant function. The second vibration mode corresponds to the Fiedler vector. This solution divides the membrane into two parts. Each part is connected and moves synchronously. One part will have positive displacement while the other has negative displacement. These are the nodal domains induced by the eigenfunction (more on this is Section 1.14).

Now, suppose instead of a membrane we have a mass/spring system with a finite number of points (nodes) in a plane. The masses are free to move perpendicular to the plane. The equation describing this is $M\ddot{z} + Kz = 0$ where M is a mass matrix, K a stiffness matrix and z a vector whose z_i entry describes the displacement of the ith point mass.

Assuming a standing wave solution $z = Ze^{i\omega t}$, the equation becomes the generalized eigenvalue problem $-\omega^2 MZ + KZ = 0$. Here the values of any eigenvector Z represent the magnitude and sign of vibration of each node for the corresponding fundamental frequency ω. Again consider the Fiedler vector which defines the 2nd vibrational mode. Nodes with the same sign are connected and move synchronously. This consideration of vibrational nodes imposes a natural bicluster on the mass/spring system.

Suppose we have masses of all unity so that $M = I$. Take the point masses to be vertices in a graph and the spring constants w_{ij} to be a weight assigned to the edge connecting vertices i and j where $i \neq j$. The stiffness matrix K is now the graph Laplacian. To see this note that an element of the vector KZ is the net force on node

i for displacement vector Z or

$$(KZ)_i = \sum_j w_{ij}(z_i - z_j) = [\sum_j w_{ij}]z_i - \sum_j w_{ij}z_j$$

where

$$K_{ij} = \begin{cases} \sum_j w_{ij} \text{ if } i = j \\ -w_{ij} \text{ if } i \neq j \end{cases}$$

Thus we arrive at the problem expressed in terms of a graph Laplacian. Spectral clustering produces a biclustering corresponding to the second mode of vibration. This clustering satisfies our intuitive notion of a clustering as each cluster is connected, vibrates in phase (similarity) and vibrates π degrees out of phase with the other cluster (dissimilarity).

The statement that the clusters are connected is intuitively true for the membrane with an infinity of points and can be proven (see Section 1.14) but is not as apparent for the finite mass/spring system. We have also assumed that there are no boundary ($z_i = 0$) nodes. We will use some theorems by Fiedler to resolve these issues.

1.6 Fiedler Theorems

The 1975 paper by Miroslav Fiedler [26] proved two theorems on the Fiedler vector which he referred to as the characteristic valuation of a graph.

While the work of Fiedler is mentioned by some papers [53],[57],[36] it is not central to the combinatorial justification for spectral clustering. We will see however that it clarifies certain aspects and is important for understanding practical numerical results of spectral clustering.

We repeat Fiedlers theorems here with some modification of terminology.

Theorem 1.4 *(Fiedler [26]) Let $G = (V, E)$ be a finite connected, weighted graph. Let L be the unnormalized Laplacian of G and u the Fiedler vector of L. Then $\forall \;\; r \geq 0$ the subgraphs induced by $A = \{i \in V : u_i \geq -r\}$ and $B = \{i \in V : u_i \leq r\}$ are connected.*

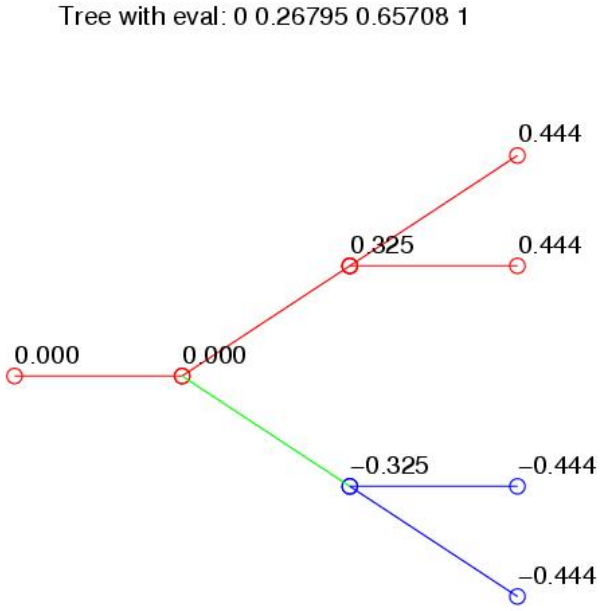


Figure 1.3: Fiedler Vector for a Tree

Corollary 1.5 *(Fiedler [26]) If $r = 0$ then both A and B are connected.*

Theorem 1.6 *(Fiedler [26]) Given any edge cut of a finite, connected, weighted graph G that creates two connected components there exists a weighting (positive valuation) of the edges of G such that the Fiedler vector defines the same two components. This partition is derived via positive and negative values of the Fiedler vector and there are no zero values in the Fiedler vector.*

In Figure 1.3 (a tree), Figure 1.4 (a lattice), and Figure 1.5 (a wheel) we display Fiedler vector values for three graphs with constant edge weights. The values for the Fiedler vector are listed against the corresponding vertex. These examples demonstrate how Theorem 1.4 would partition the graph. Lattices will be used for image biclustering. The wheel in particular shows how using ≥ 0 and < 0 as partitioning criteria can lead to disconnected clusters.

The first thing to note is that there is no good reason to preferentially place nodes with $u_i = 0$ in cluster A or B. But there is a very good reason to place these boundary

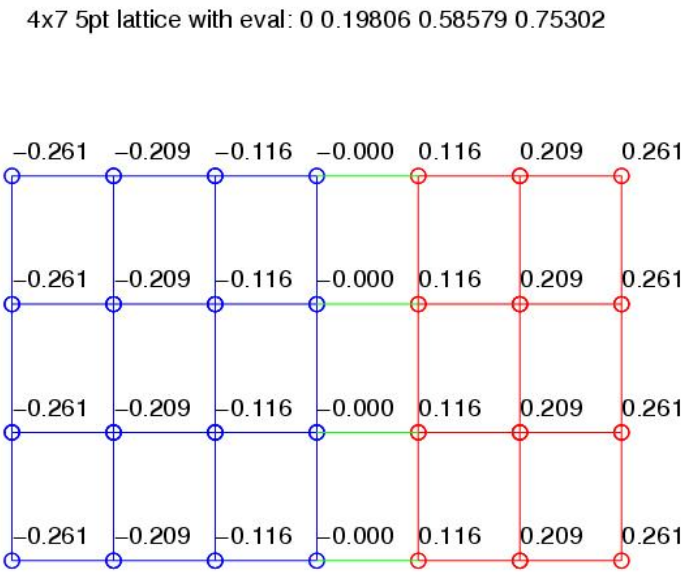


Figure 1.4: Fiedler Vector for a Lattice

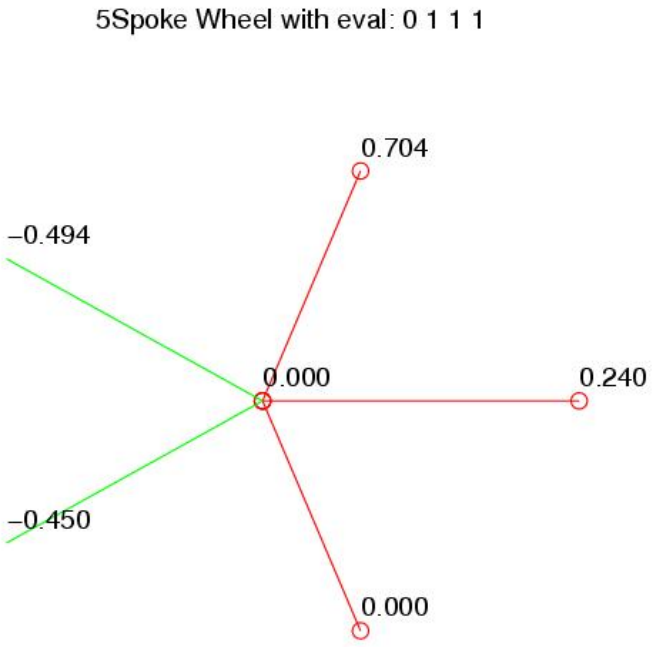


Figure 1.5: Fiedler Vector for a Wheel

nodes in both clusters A and B. We see from Corollary 1.5 that both clusters A and B will now be connected.

Theorem 1.6 demonstrates the importance of the weights we place on edges. Choosing any bicluster giving connected clusters there is a weighting of the edges which will produce it. So we must be very careful to choose weights which impart some meaning to the similarity of vertices.

We will present a model for edge weights on images in Section 1.10.

1.7 An Extension of Fiedlers Theorem

The theorems of Fiedler as applied to graphs are stated for unnormalized graph Laplacians. We propose and prove a more general version of them such that they will apply to the normalized graph Laplacians and the more general eigenvalue problem $Lx = \lambda Mx$ where M is any positive diagonal matrix. This problem corresponds to a mass/spring system where the mass matrix is not necessarily the identity matrix. In particular, we want to show that an eigenvector corresponding to the second smallest eigenvalue of this generalized eigenvalue problem can be used to partition the associated graph for the problem into connected components.

To this end we state two lemmas and a theorem from Fiedlers 1975 paper [26] that we will need in our proof. Recall that a matrix is irreducible if it is not reducible and (see "Special Matrices and Their Applications in Numerical Mathematics" [27] page 79) a square matrix is reducible if it is in the form

$$A = \begin{vmatrix} B & C \\ 0 & D \end{vmatrix}$$

or can be placed in this form via permutation $P^T AP$.

Lemma 1.7 *(Fiedler [26]) Let C be a diagonal matrix with $c_{ii} > 0$ and K a symmetric irreducible matrix, then CKC is symmetric irreducible.*

Lemma 1.8 *(Fiedler [26]) Let K be a symmetric irreducible matrix and $\alpha > 0$ such that the diagonals of $K + \alpha I$ are not zero, then $K + \alpha I$ is symmetric irreducible.*

These Lemmas follow from the observation that under the operations as defined the positions of non-zero elements in K are unchanged and no new zero elements are created.

Theorem 1.9 *(Fiedler [26]) Let A be an nxn nonnegative irreducible symmetric matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Let $u = (u_i) > 0$ be an eigenvector corresponding to λ_1 and $v = (v_i)$ an eigenvector corresponding to λ_2. Then for any $\alpha \geq 0$, the submatrix $A(M_\alpha)$ is irreducible where $M_\alpha = \{i : v_i + \alpha u_i \geq 0\}$.*

Note: $u = (u_i) > 0$ exists by the Perron-Frobenius Theorem [37] [25]. That is. since A is nonnegative and irreducible, there exists a positive eigenvector u for λ_1.

We now generalize Fiedlers Theorem as follows.

Theorem 1.10 *Let $G = (V, E)$ be a finite connected graph with vertices V numbered $1, \ldots, n$ and edges E assigned a positive number k_{ij}. Let $K \in \mathbb{R}^{nxn}$ be the unnormalized Laplacian of G. Let $M \in \mathbb{R}^{nxn}$ be a diagonal matrix where $m_{ii} > 0$. Let y be the Fiedler vector of the generalized eigenvalue problem $Kx = \lambda Mx$. For $r \geq 0$, let $M_r = \{i : y_i + r(M^{\frac{1}{2}})_i \mathbb{I}_i \geq 0\}$ where $\mathbb{I} = (1, 1, \ldots, 1)^T$. Then G_r the subgraph induced on G by M_r is connected.*

Proof: We note that this proof follows closely that of Fiedler for the unnormalized Laplacian.

Since G is connected K is irreducible. Let $B = -M^{-\frac{1}{2}} K M^{-\frac{1}{2}}$. B is nonnegative on the off diagonal elements, since $B_{ij} = -\frac{1}{\sqrt{m_{ii}}} M_{ij} \frac{1}{\sqrt{m_{jj}}}$ and is the product of terms with sign of $-+-+$. some $\alpha > 0 : B + \alpha I$ is nonnegative, symmetric, has positive diagonals and by Lemmas 1.7 and 1.8 is irreducible. Furthermore, (λ, x) is an eigenpair

of $Kx = \lambda Mx \iff (\hat{\lambda} = -\lambda + \alpha, y = M^{\frac{1}{2}}x)$ is an eigenpair of $(B + \alpha I)y = \hat{\lambda}y$.

Now, 0 is an eigenvalue of $Kx = \lambda Mx$ with eigenvector $\mathbb{I}$ (all ones). Since K is positive semidefinite 0 is the smallest eigenvalue. Then $u = M^{\frac{1}{2}}\mathbb{I}$ is the eigenvector of the largest eigenvalue of $B + \alpha I$. Let v be the eigenvector of the second largest eigenvalue of $B + \alpha I$.

Let $r > 0$ and $M_r = \{i : v_i + ru_i \geq 0\}$. By Theorem 1.9, the submatrix $(B + \alpha I)(M_r)$ is irreducible $\implies K(M_r)$ is irreducible $\implies$ the subgraph induced on G by M_r is connected.

■

In particular take $r = 0$. Then $\{i : y_i \geq 0\}$ defines a connected subgraph of G and since $-y$ is also a Fiedler vector for $Kx = \lambda Mx$, we see that $\{i : y_i \leq 0\}$ also defines a connected subgraph of G.

The generalized eigenvalue problem $Kx = \lambda Mx$ is equivalent to $-By = \lambda y$ where $y = M^{\frac{1}{2}}x$. Note that y preserves the sign of eigenvector x and so x and y produce the same clustering for $r = 0$.

The generalized eigenvalue problem is also equivalent to $M^{-1}Kx = \lambda x$ with the same Fiedler vector as in the original problem.

For the case $M = D$ (the diagonal of K) we get the normalized equivalents of the graph Laplacian, $D^{-\frac{1}{2}}KD^{-\frac{1}{2}} = L_{sym}$ and $D^{-1}K = L_{rw}$.

So, without recourse to combinatorial models we have shown that the normalized graph Laplacians and the more general forms of these equations using a mass matrix produce connected subgraphs via spectral clustering.

1.8 Effect of mass matrix on segmentation

Having extended one of Fiedlers Theorems (see Section 1.7) with respect to the mass matrix Laplacian problem $Lx = \lambda Mx$, we wished to examine the effect of the mass matrix on clustering.

Numerical experiments were performed. We present the following observations which apply to biclusters produced via the Fiedler vector.

1. Clusters defined by division of the Fiedler vector via ≥ 0 and ≤ 0 will be connected. This is insured by our extension of Fiedlers Theorem.

2. The two vertices of highest mass tend to be in separate clusters; see Figure 1.6 for two examples of this.

3. For the vertex of highest mass, if the mass is large enough, it will be in a cluster by itself provided observation 1 is not violated; see Figure 1.7 and Figure 1.8.

4. The vertex of smallest mass is never isolated in a cluster by itself; see Figure 1.9.

Observation 3 can be understood (but not proven) by the following. Reformulate the problem $Lx = \lambda Mx$ as $M^{-\frac{1}{2}}LM^{-\frac{1}{2}}y = By = \lambda y$. Let m_i be the mass of vertex i. The matrix B has elements $B_{ij} = \frac{1}{\sqrt{m_i}}A_{ij}\frac{1}{\sqrt{m_j}}$. If we take B row by row we see that we are adjusting the weight of the edges of row (vertex) i by $\frac{1}{\sqrt{m_i}\sqrt{m_j}}$. Suppose m_i is the largest weight and $m_i >> m_j$ for every $i \neq j$. If the original edge weights are large relative to $\frac{1}{\sqrt{m_i}}$, then we now have the smallest weights around vertex i and would expect it to be in a cluster by itself.

Observation 4 follows for similar reasons with $m_k << m_i$ and the largest weights are around vertex k. So it will be strongly associated with the surrounding vertices.

Numerical experiments for these results are reproduced in the following figures. At each vertex its mass and Fiedler vector value is listed. The Laplacian was setup

with constant edge weights and the Matlab eig function was used to solve for the Fiedler vector.

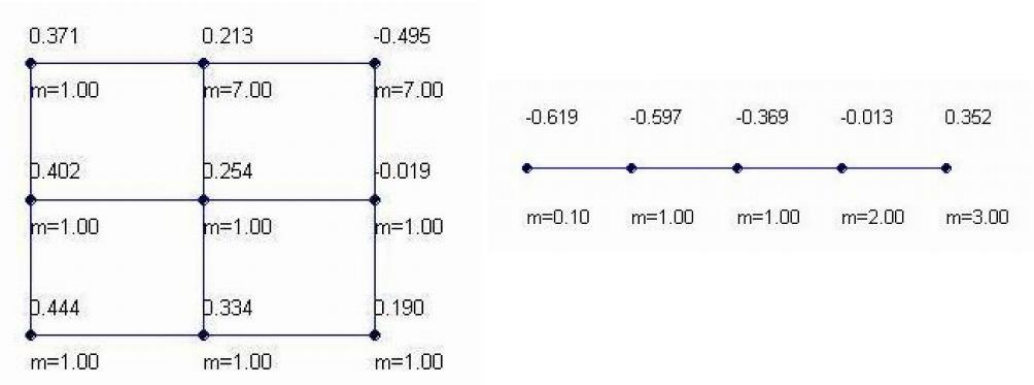


Figure 1.6: Observation 2: Separation of Two Highest Masses

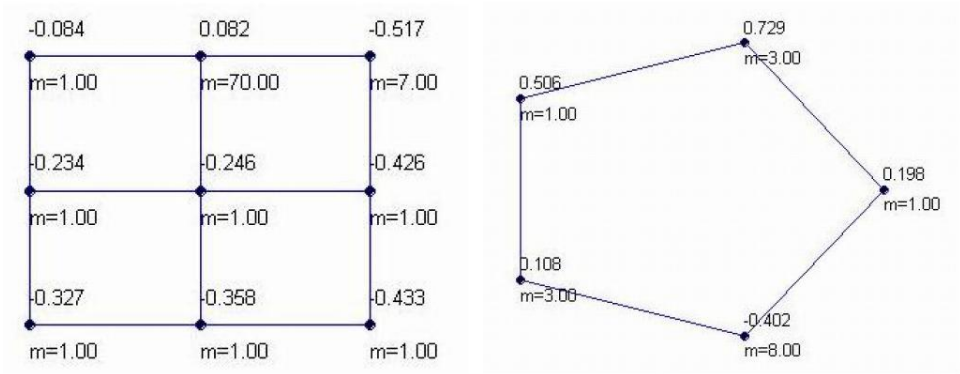


Figure 1.7: Observation 3: Largest Mass in a Cluster by Itself

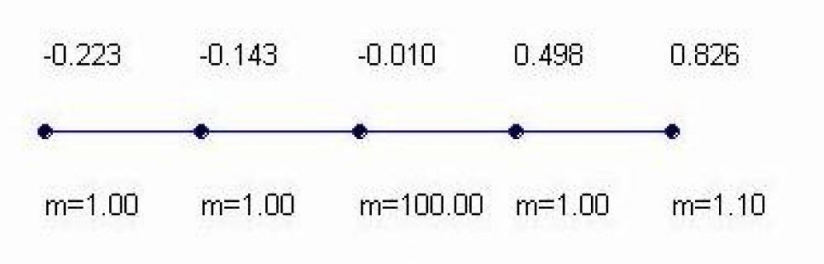


Figure 1.8: Observation 3: When Largest Mass can't be in a Cluster By Itself

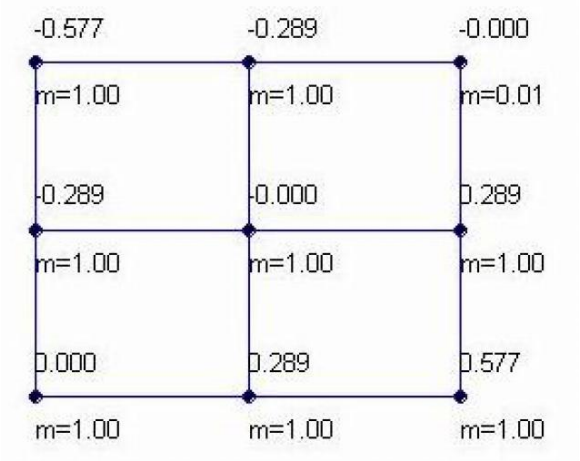


Figure 1.9: Observation 4: Smallest Mass never in a cluster by itself

1.9 Image Segmentation

Our problem is to bipartition an image according to the ideas discussed in Section 1.5. A real world object is like the membrane, a continuous, infinite (for all practical purposes) number of points. When we take a picture we reduce the real world image to a finite number of pixels. In a simple case, each pixel has red,green,blue (RGB) values from 0 to 255. We consider the discrete image as a graph with vertices corresponding to the pixels which are labeled as consecutive positive integers from the lower left edge of the image to the upper left edge column by column as shown in Figure 1.10 for an image with m rows and l columns.

Edges are defined based on a 5 point stencil. The result is a $m \times l$ lattice. If we had a 3 dimensional image we would define similarly a pixel numbering with edges defined on a 7 point stencil.

Note that the graph Laplacian of a lattice has a structure imposed by the 5 point stencil and the numbering choice for the vertices. The only non zero elements of the Laplacian lie on the diagonal, adjacent to the diagonal, and $\pm m$ columns from the diagonal.

$$
\begin{array}{llll}
m \circ & 2m & \circ \ldots \circ & ml \\
\vdots & & \vdots & \vdots \\
2 \circ & m+2 \circ \ldots & \vdots & \\
1 \circ & m+1 \circ \ldots \circ & (l-1)m+1 &
\end{array}
$$

Figure 1.10: Labeling of Pixels in an Image

The final component we need to construct the Laplacian are the edge weights.

1.10 Edge Weights

There are many ways to assign weights. The method we choose is implemented in Leo Grady's software [31]. It is based on a combination of RGB values and geometric distance.

Say pixel 1 and pixel 2 are connected by an edge and they have RGB values $\{R1, G1, B1\}$ and $\{R2, G2, B2\}$. Let $\delta R = R1 - R2$, $\delta G = G1 - G2$, and $\delta B = B1 - B2$.

Let $valDist = \sqrt{\delta R^2 + \delta G^2 + \delta B^2}$. Then normalize the values with respect to the range $[0, 1]$. Set $geomDist = 1$, which is the same for all edges.

The distances are a measure of dissimilarity between pixels. For the Laplacian we need measures of similarity. This is achieved by using the function $e^{-dist^k * a}$ where a is a constant and k a positive integer. These control the rate that similarity declines with distance. For our model we choose $k = 1$.

Definition 1.11 $edgeWeight = e^{-dist} + epsilon \; where \; dist = geomScale * geomDist + valScale * valDist$

We have introduced 3 parameters: geomScale and valScale adjust the relative importance of geometric vs RGB values. Geometric distances are always unity, since the pixels are nodes of a uniformly spaced lattice, but value distance ($valDist$) can vary from 0 to 1 and values are typically less than 0.05. geomScale is typically chosen

to be 1 and valScale 25 or larger. *epsilon* is the minimum allowable weight. A typical value for *epsilon* is 10^{-5}. We will discuss the impact of these parameters in Section 1.12.

1.11 Spectral Clustering Algorithm

The remarks of the previous Sections are embodied in the following algorithm.

1. Formulate the problem as a graph $G = (V, E)$ with n vertices.

2. Number the vertices of G from 1 to n.

3. To each edge we assign weights and construct a weighted adjacency matrix A for G and the associated degree matrix D.

4. Construct Laplacian matrix $L = D - A$.

5. Compute the Fiedler vector $Y = \{y_1, y_2, \ldots, y_n\}$ of the Laplacian L.

6. BiCluster the graph into subgraphs A and B using Y as follows:

   $A = \{i \in V : y_i \geq 0\}, \quad B = \{i \in V : y_i \leq 0\}.$

Notes:

- The graph G must be connected.
- There are many ways to assign weights. Section 1.10 only describes one method.
- $A \bigcup B = V$ but it's possible that $A \bigcap B \neq \emptyset$ since there may be $y_i = 0$.
- With respect to the Vibrational justification for spectral clustering, A and B are the optimal solutions. Their RatioCut values are irrelevant.
- Since both A and B are connected, we can use successive applications of the algorithm to generate smaller clusters.

1.12 Tuning the Edge Weight Parameters

Some numerical experiments using Matlab with the image in Figure 1.11 were performed to illustrate the effect of the parameters on weight assignment as described in Section 1.10. The image is of a cats ear with dark fur against a light gray background with some white whiskers and fur. This has an obvious clustering consisting of the ear and the background.

The primary image is 200x320 pixels with RGB values from 0-255. A graph Laplacian is constructed as defined in the previous Sections.

Figure 1.11: Base Image for Analysis

For the initial analysis the image is scaled by sampling of pixels into images with smaller resolution. Weights are assigned with $valScale = 25$, $geomScale = 1$, and $epsilon = 10^{-5}$. The Matlab *eigs* sparse eigensolver was used.

The results are shown in Figure 1.12, where a light line has been added to the image to show the boundary separating the clusters generated. The graph in the

lower right hand corner shows solution time in seconds (y-axis) versus image size in pixels (x-axis).

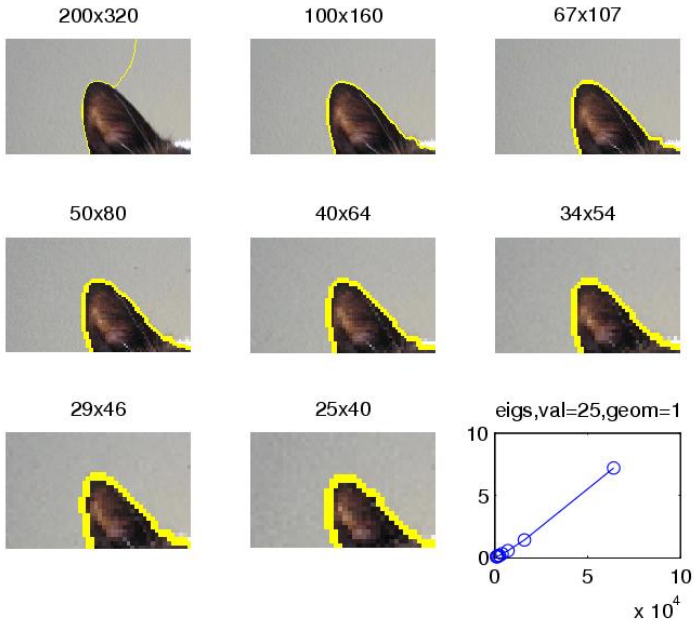


Figure 1.12: Initial Analysis

The results are good with the exception of the 200x320 image where the boundary deviates upwards to the top edge of the image. This can happen when the $valDist$ is too small with respect to the $geomDist$.

Consider the lattice in Figure 1.4. In this graph the edge weights are all one and the lattice could be considered as an image where RGB values for all pixels are the same. Note how the boundary $\{i : y_i = 0\}$ bisects the image from top to bottom (the smaller dimension) at the midpoint of the larger dimension. This illustrates how the geometry of the image can affect the Fiedler vector, and we might suspect that the distance term is overpowering the value term.

Adjusting the $valScale$ to 50 and repeating the experiment solves the problem, see Figure 1.13.

Why don't we see this problem at the reduced image scales? As the scale is reduced by sampling the computed mean $valDist$ is increased from .018 for the 200x320

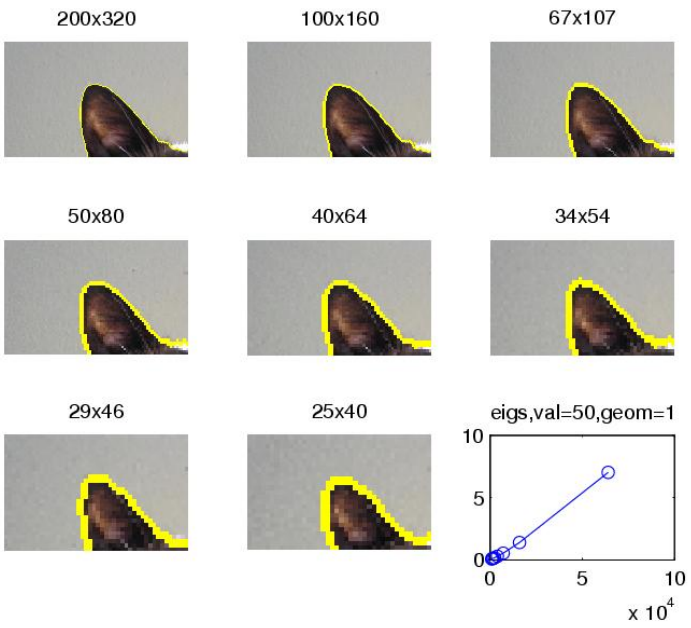


Figure 1.13: Effect of increasing ValScale

image to .05 for the 25x40 image. As a result the $valScale * valDist$ term increases with respect to the $geomScale * geomDist$ term which is unchanged. This has the same effect as adjusting $valScale$.

Similarly varying $geomScale$ with respect to $valScale$ can cause degradation of the bicluster as shown in Figure 1.14 for the 40x64 image.

So what does $epsilon$,in $edgeWeight = e^{-dist} + epsilon$, do? Very small edge weights can occur when we have a small number of connected pixels with RGB values very different from those surrounding them. These "islands" or "specks" within the image can dramatically affect the clustering. To see this consider the lattice in figure 1.4 but with the edges of a single vertex made smaller than one (vertex number 6 using the numbering previously defined). The result is shown in Figure 1.15. Vertex 6 is now in a cluster by itself.

We can see this in our ear image if we set $epsilon = 10^{-8}$. The result shown in Figure 1.16 shows the effect of one such "speck" in the 100x160, 50x80, and 40x64 scaled images. In the 40x64 image we have magnified the image to show more clearly

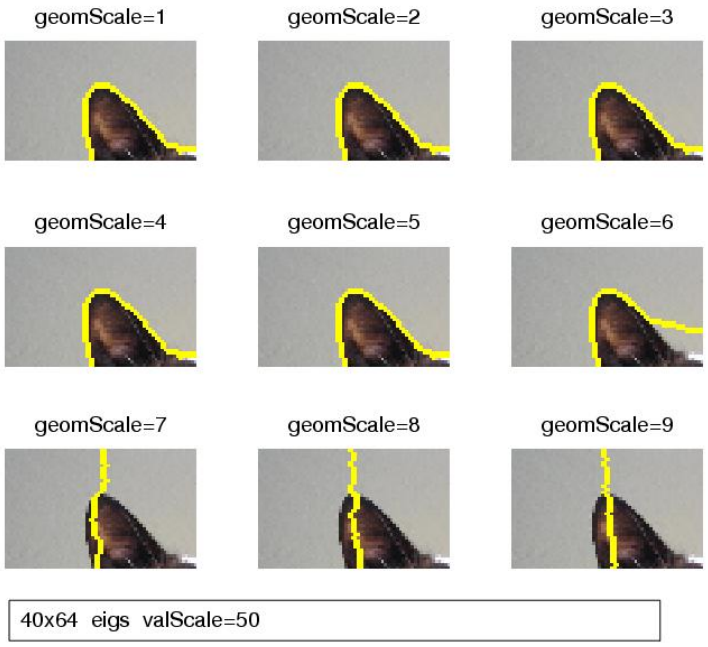


Figure 1.14: Effect of *geomScale*

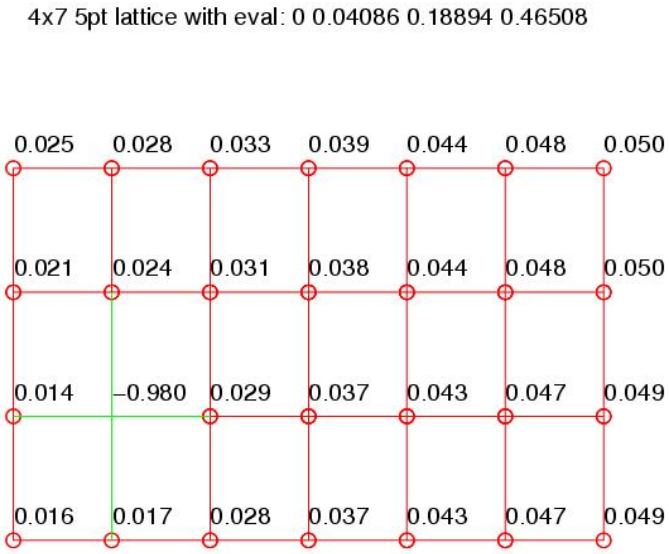


Figure 1.15: Effect of Islands

how a single pixel has been isolated. Epsilon smooths out large dissimilarities in RGB values.

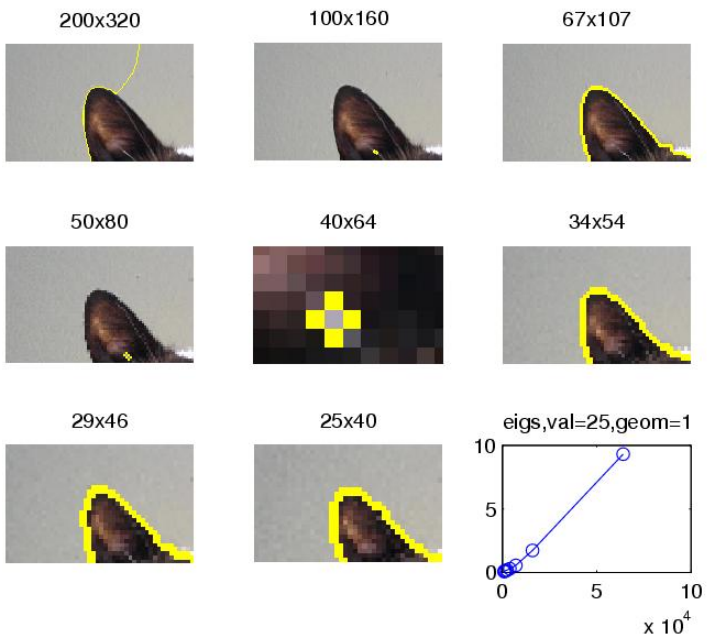


Figure 1.16: Effect of Epsilon

1.13 Eigenvalue Solvers

When using the Matlab *eigs* function we add a shift to the Laplacian, so that it is now positive definite. If the shift is too large, there is not good relative separation with respect to the smaller eigenvalues and *eigs* may not return a good Fiedler vector. Also, if we are solving to a tolerance that is too large the Fiedler vector may be of low quality.

In these cases disconnected clusters can be produced, see Figure 1.17 where a shift of 10^-5 was used. Note that for the 50x80, 40x64, and 29x46 scaled images the cluster boundary gives disconnected clusters. We know from Corollary 1.5 that this is not possible, so something must be wrong with the solution for the Fiedler vector.

The *eigs* function was usable in our experiments since the size of the image was small. Larger images will produce Laplacians with dimensions in excess of 10^7. This size problem requires the use of large sparse eigensolvers such as BLOPEX

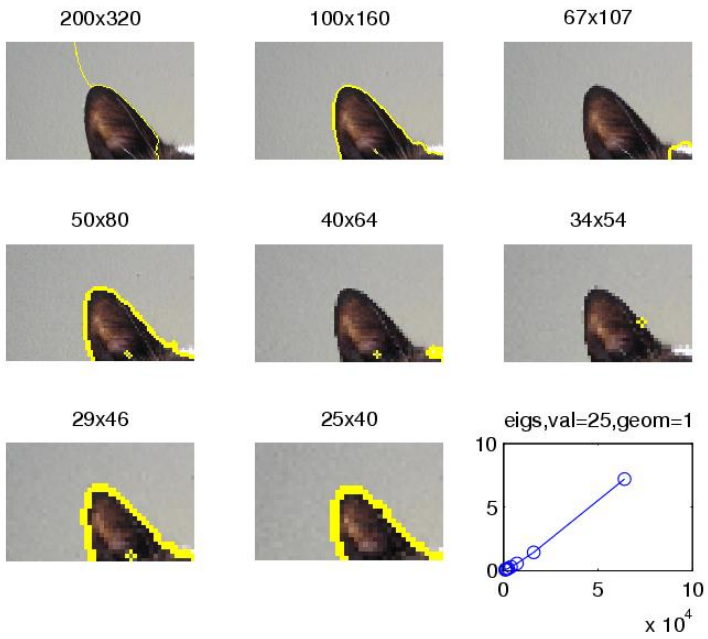


Figure 1.17: Poor Quality Fiedler Vector produced by too small a shift

[44],[41]. These implement iterative solvers such as LOBPCG [43] and make use of preconditioners [42]. But, whatever method is used, if either $\{i \in V : f_i \geq 0\}$ or $\{i \in V : f_i \leq 0\}$ is not connected then the Fiedler vector produced is suspect.

Image segmentation can practically exploit these eigensolvers. The Laplacian is symmetric positive semi-definite with smallest eigenvalue of zero. It can be shifted to produce a positive definite matrix without changing the eigenvectors. We know what the eigenvector is for eigenvalue zero, and this can be used as an orthogonal constraint for solution of the next eigenpair. These eigensolvers are designed to solve for only a few of the smallest eigenvalues, and we only need the second eigenvalue and associated eigenvector, so spectral clustering can be done efficiently.

1.14 Nodal Domain Theorems

Fiedlers theorem deals with the bi-clustering of a graph based on the 2nd eigenvector of a graph Laplacian (Fiedler Vector). We have provided an extension of this to a generalized eigenvalue problem. A Theorem similar to Fiedlers exists for higher eigenvectors. We summarize the following work done on nodal domains and

it's discrete counterparts.

In Courant and Hilberts book "Methods of Mathematical Physics, Volume 1",Ch. 6, Sec. 6 [13] they present the following theorem. This dates from 1924, and is frequently referred to in papers on nodal domains. It is often referred to as CNLT (Courants Nodal Line theorem).

Theorem 1.12 *(CNLT [13] pg 452) Given the self-adjoint second order differential equation*

$$L[u] + \lambda \rho u = 0 \qquad (\rho > 0)$$

for a domain G with arbitrary homogeneous boundary conditions; if its eigenfunctions are ordered according to increasing eigenvalues, then the nodes of the n-th eigenfunction u_n divide the domain into no more than n subdomains.

By nodes is meant the set $\{x \in G : u_n(x) = 0\}$. Depending on the dimension of G this might be a nodal point, curve, surface, etc.

These nodes (also referred to as nodal lines) divide the domain G into connected sub-domains called nodal domains.

If G has dimension m then the nodes will be hyper surfaces of dimension $m - 1$. It can be shown that the nodes are either closed or begin and end on the boundary [30], and are of Lebesque measure zero [9].

The proof of CNLT provided by Courant and Hilbert is rather cryptic; being more of a discussion than a formal proof and is for the case of $G \subset R^2$. A more accessible proof for the more general case of $G \subset R^m$ is provided in a paper by Gladwell and Zhu [30].

While not giving the details of the proof we will examine it's basic features and see how it acts as a template for a discrete CNLT. Ultimately, we will derive a new version of the discrete CNLT, that includes Fiedlers theorem as a special case. Fiedlers proof depends on theorems of of positive, irreducible matrices. It is instructive to see how an entirely different approach than that used by Fiedler yields the same result.

The CNLT proof depends on:

1. the nature of the eigenfunctions and eigenvalues of the problem, i.e. the eigenvalues can be ordered and the eigenfunctions form a complete orthonormal set,
2. the variational form of the problem,
3. the application of the min-max characterization of eigenvalues (Courant-Fischer Theorem), and
4. the unique continuation theorem.

Gladwell and Zhu prove the problem for the Helmholtz equation $\triangle u + \lambda \rho u = 0$ on a domain $\mathbb{D}$.

This has infinitely many positive eigenvalues $0 \leq \lambda_1 \leq \lambda_2 \leq ...$ whose corresponding eigenfunctions form a complete orthonormal set, see Griffel [33] Thm 9.16. $\lambda_1 = 0$ for the free membrane problem and $\lambda_1 > 0$ for the fixed membrane problem.

Those of us primarily versed in matrix analysis are familiar with the Courant-Fischer theorem from Horn and Johnson [37] Thm 4.2.11. The equivalent on a domain $\mathbb{D}$ that is a subset of a Hilbert space is given in Griffel [33] Thm 10.18.(b) and is repeated here.

Theorem 1.13 *(Griffel [33] Maximum Principle pg 287) If A is a positive symmetric differential operator, on a Hilbert space H, with domain* D*, a compact inverse, and with eigenvalues* $\lambda_1, \lambda_2, ...,$ *then*

$$\lambda_{n+1} = \max_{k_1,...,k_n \in \mathbb{D}} \{\min\{R(u) : u \perp span\{k_1, ..., k_n\}\}\}$$

where $R(u) = \frac{<u,Au>}{<u,u>}$ *is the Rayleigh Quotient.*

Gladwell's proof will examine solutions of the variational form of the problem on the space $H_0^1(D)$. $H_0^1(D)$ is the space of $L_2(D)$ functions, whose derivatives are in

$L_2(D)$ and vanish on the boundary of D [39] [33]. The domain D is assumed to be bounded and connected. In this case, we have, $A = -\frac{1}{\rho(x)}\triangle$ and $R(u) = \frac{\int_{\mathbb{D}} \triangle u \cdot \triangle u}{\int_{\mathbb{D}} \rho u u}$.

The unique continuation theorem [38] states that if any solution $u \in H_0^1(\mathbb{D})$ of the Helmholtz equation vanishes on a non-empty open subset of $\mathbb{D}$ then $u \equiv 0$ in $\mathbb{D}$.

The complete proof of the CNLT is in three parts. First a theorem by Courant and Hilbert that there are at most $n + r - 1$ sign domains for the eigenfunction u_n of λ_n. Then a theorem by Hermann and Pleijel that shows for a connected domain this number $(n + r - 1)$ can be improved to n, and finally the extension of this limit to unconnnected domains.

The continuous CNLT serves as motivation for a discrete version. The proof for this discrete version was not completed until 2001 in a paper by Davies, Gladwell, Leydold, and Stadler [14]. The discrete CNLT deals with solutions of $Au_n = \lambda_n u_n$ and requires the following terminology and definitions.

Let $A \in M_n$ be a real symmetric matrix with non-positive off diagonal elements. A graph Laplacian L would be an example. Other examples would be problems of the form $L + \alpha D$ where D is a diagonal matrix and $L = -1$ times the adjacency matrix of a graph.

Solutions of $Au_n = \lambda_n u_n$ are of the form $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. Whatever the exact form of A we can associate it with the off diagonal elements of a graph $G = (V, N)$ with vertices V corresponding to the columns and edges E (possibly weighted) associated with the non-zero elements a_{ij}.

The nodal set of an eigenvector u_n does not lend itself to an exact correspondence with the nodal set and nodal domains in the continuous case. Instead, strong and weak sign graphs are defined as follows.

Definition 1.14 *(Davies [14]) A strong positive (negative) sign graph is a maximal, connected subgraph of G, with vertices $i \in V$ and $u(i) > 0$ $(u(i) < 0)$.*

Definition 1.15 *(Davies [14]) A weak positive(negative) sign graph is a maximal, connected subgraph of G, with vertices $i \in V$ and $u(i) \geq 0$ $(u(i) \leq 0)$.*

We speak of a vertex as having a sign of positive, negative, or zero. We note two key components of these definitions. The subgraphs are connected and they are maximal in the sense that no vertices of the same sign as the subgraph can be added to the subgraph and have it still be connected.

A few examples in Figure 1.18, that we derived, will illustrate these concepts. The eigenvectors of the graphs were computed using the unnormalized Laplacian with constant edge weights. Only the signs of vertices are shown.

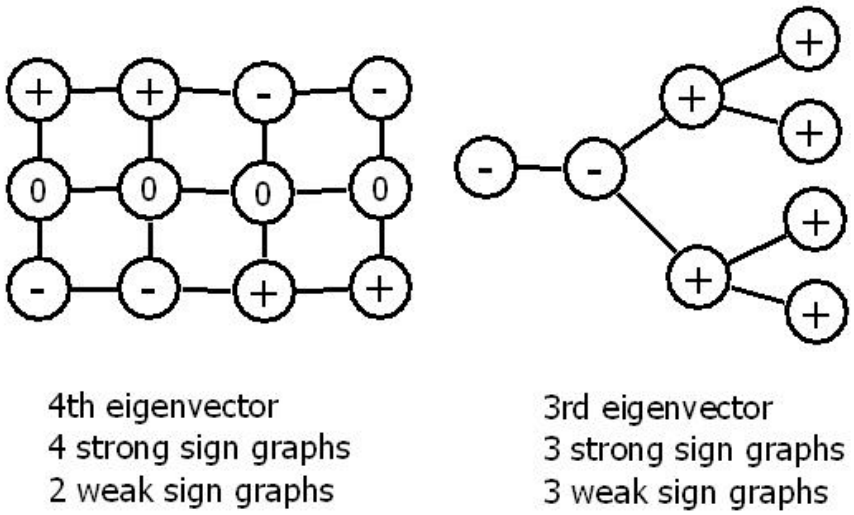


Figure 1.18: Examples of Strong and Weak Sign Graphs

Given these definitions and terminology the discrete CNLT is formulated as follows.

Theorem 1.16 *If $G = (V, E)$ is a connected graph and A a symmetric matrix with non-positive off diagonal elements, the n-th eigenfunction u_n of A divides the graph into no more than n weak sign graphs.*

The proof of the discrete CNLT proceeds as for the continuous CNLT and depends on the ordering of the eigenvalues, the orthonormality of the eigenfunctions, and the min-max characterization of the eigenvalues. Instead of using the variational form used in the continuous CNLT an identity from Duval and Reiner [22] is used. Also, we do not have the unique continuation theorem in the discrete case but there is an analog (see Davies et al. Lemma 3 [14]).

It should be noted that much of this proof was done in the previous paper by Duval and Reiner. However, they made the claim that the CNLT was valid for strong sign graphs. Friedman [29] had previously given examples showing this is not true. Figure 1.19 shows two examples of this. The more complicated one on the left being one we derived.

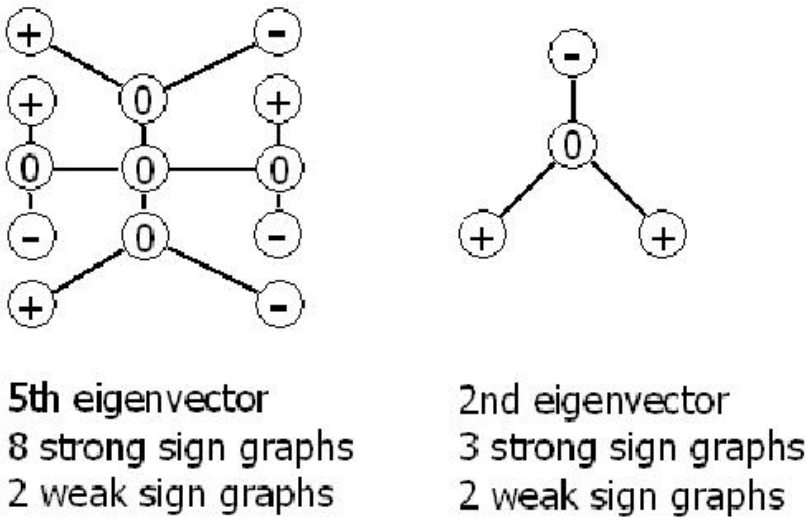


Figure 1.19: Strong Sign Graphs exceeding eigenvector number

Also, note that the graph G must be connected and the theorem applies to weak rather than strong sign graphs.

A few conclusions can be draw from the definitions and an examination of examples.

- The number of weak sign graphs is always $\leq$ the number of strong sign graphs.
- The number of both weak and strong sign graphs can be $< n$.
- If there are no zero vertices the number of strong and weak sign graphs are the same.
- If there are zero vertices the number of strong and weak sign graphs can still be the same, see Figure 1.20.
- The number of weak or strong sign graph can decrease with increasing n.

The last item seems a bit counter intuitive, but is demonstrated by an example we derived in Figure 1.20.

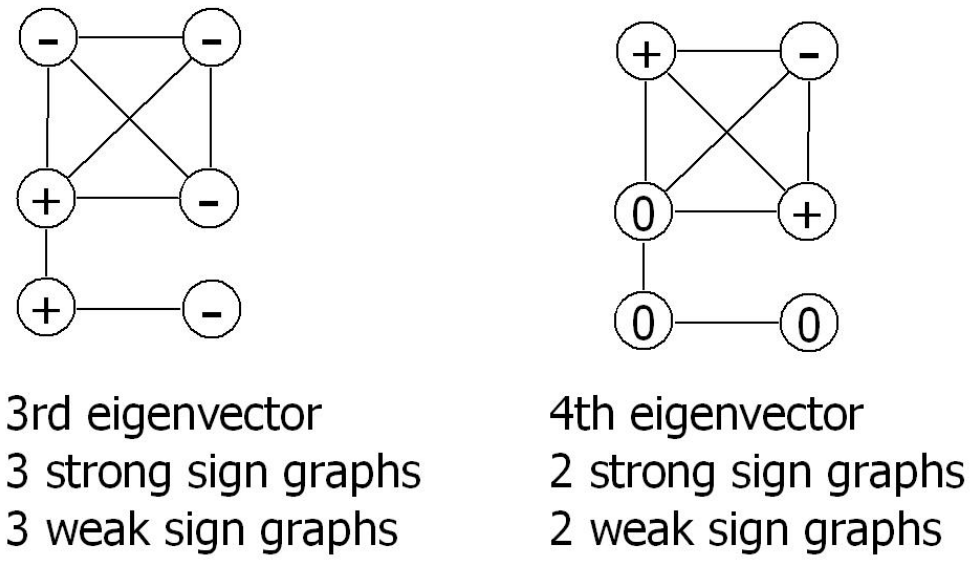


Figure 1.20: Sign Graphs decreasing in number

Finally, in the concluding remarks of Davies paper he states that the theorem can be extended to the generalized eigenvalue problem $(K - \lambda M)u = 0$ where K is positive semi-definite and M is positive definite.

Fiedlers theorem is in part a special case of the discrete CNLT for $n = 2$ where the cut is with respect to $r = 0$. However, note that Fiedlers theorem covers subgraphs induced by $\{i \in V : u(i) \geq -r\}$ and $\{i \in V : u(i) \leq r\}$ where $r > 0$, which the CNLT does not address.

We will modify the definition of weak sign graphs and produce a new version of CNLT that completely covers Fiedlers theorem as a special case.

Definition 1.17 *An r-weak positive (negative) sign graph is a maximal, connected subgraph of G, with $r > 0$ and vertices $i \in V$ and $u_i \geq -r$ ($u_i \leq r$). (For an example see Figure 1.21.)*

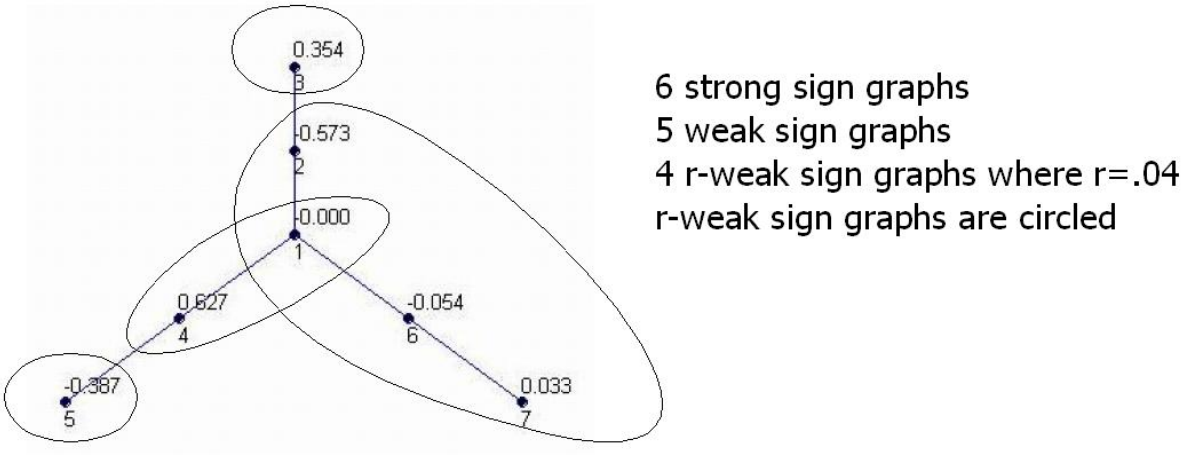


Figure 1.21: An example of R-weak Sign Graphs

Theorem 1.18 *(Modified Discrete CNLT) If G is a connected graph, A a symmetric matrix with non-positive off diagonal elements, and $r > 0$ the n-th eigenfunction u_n of A divides the graph into no more than n r-weak sign graphs.*

Proof: Let $r \geq 0$ and R be an r-weak sign graph (rwsg). Then either there exists a weak sign graph (wsg) W such that $W \subset R$ (we will call this type 1) or if no such wsg exists then there exists a wsg U such that $R \subset U$ (we will call this type 2). To see this, suppose R is a maximal set generated from $f_i \leq r$. Then if R has

entries ≤ 0 it is type 1. If it doesn't then it has entries ≥ 0 and $\leq r$ in which case R is a wsg U with $R \subset U$ and it is type 2. We can make similar arguments when R is generated from $f_i \geq -r$.

Let $|wsg|$ be the number of wsg and $|rwsg|$ be the number of rwsg. Suppose R_1 and R_2 are type 1 rwsg and W is a wsg such that $W \subset R_1$ and $W \subset R_2$. Since R_1 and R_2 are maximal we must have $R_1 = R_2$. So $|wsg| \geq |type\ 1\ \ rwsg|$.

Let R_1 and R_2 be rwsg of type 2 with $R_1 \subset W_1$ and $R_2 \subset W_2$ where W_1 and W_2 are wsg. By maximality if $R_1 \neq R_2$ then $W_1 \neq W_2$. So $|wsg| \geq |type\ 2\ \ rwsg|$.

Suppose there is also a rwsg R_3 of type 1 such that $R_1 \subset W1 \subset R_3$ then by maximality $R_1 = R_3$ a contradiction. So the set of wsg associated with type 2 is disjoint from the wsg associated with type 1. From this we conclude that $|wsg| \geq |rwsg|$ and by the discrete CNLT the n-th eigenfunction divides the graph into no more than n r-weak sign graphs. ■

We make an observation that the definition of strong sign graph excludes zero vertices from the partition of the graph and they are mutually exclusive. Weak sign graphs include the zero vertices, if any, and if they do they will not be mutually exclusive. This means that the set of all strong or weak sign graphs may not constitute a partition of V in the sense as usually stated in the combinatorial graph partitioning problem [6]. This being the case, partitioning objective functions (ratio cut, Ncut) lose their meaning without some redefinition.

From a spectral clustering perspective this is not a restriction and we can ask the question, "Should we be partitioning based on strong or weak sign graphs?". Both have an association with modes of vibration. And if we use strong sign graphs we may be excluding vertices, which does not seem desirable. But, in either case our clusters will be connected which we intuitively think of as desirable.

The use of r-weak sign graphs has a further advantage. We have said previously there is no good reason to include/exclude zeros from multiple clusters. But when

dealing with a numerical solution to our eigenvalue problem we will in general not know any vertex eigenvector value to exact precision. If the error is $\leq r$ where r is small we would more properly define clusters based on r-weak sign graphs. In this case the overlap between clusters may be enlarged but our clusters will still be connected.

1.15 Summary

We have reviewed the concept of spectral clustering and proposed an alternative explanation (the Vibrational model) of why it is an effective technique for clustering. We have provided an extension to one of Fiedlers theorems using similar techniques as those used by Fiedler which complements the Vibrational model. We have examined discrete nodal domain theory and by expanding the definition of weak sign graphs produced a modified CNLT that includes as a special case the CNLT and Fiedler theorem. It's use for image segmentation was explained. The importance of edge weights and a particular method for their determination was examined. Finally, some numerical results were given.

2. MicroArrays

2.1 Introduction

In the previous chapter we introduced the ideas involved in spectral segmentation and applied these to image segmentation. In this chapter we will extend these to the problem of microarray analysis. We will advocate a normalization rule for the analysis of microarray data where the objective is recognition of genes with similar expression levels in time dependent experiments.

Microarray analysis involves some practical problems that were not relevant in image segmentation. The primary one being that if we are going to have sparse Laplacians then we will almost certainly end up with disconnected graphs. We will propose a new technique for dealing with this.

We also alluded to using successive bisection in the previous chapter. We will develop and implement a technique for doing this and as a part of this propose a rule for partition selection.

2.2 What is a Microarray?

A microarray is a chip containing oligonucleotide sequences used to detect the expression level of genes. We will give a brief review of the biology inherent in this statement.

Genes are DNA sequences of nucleotides A,C,T, and G which code for proteins. The process of protein formation in a cell involves: transcription of DNA to mRNA (messenger RNA) in the cell nucleus and then translation of mRNA to a protein via a ribosome, see Figure 2.1. When proteins are being formed for a gene this is called gene expression.

Gene expression is not a constant process. It can depend on the various states a cell is in, such as metabolic cycles, disease response, and stages of embryonic development. The level of gene expression will vary. This can either be an increase or decrease from "normal" levels, referred to as up regulation and down regulation.

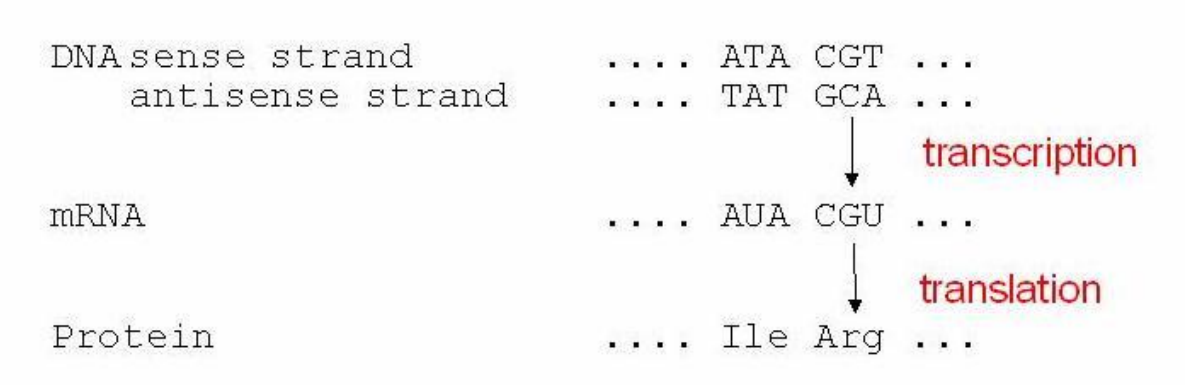


Figure 2.1: Gene Expression

Knowing which genes are expressed and by how much can aid in the understanding of cellular processes and in diagnosis of disease.

However, measurement of the concentration of proteins in a cell is difficult. One solution is to measure mRNA as a proxy. This depends on the assumption that most mRNA created is actually translated to a protein.

Various DNA microchip technologies have been designed to perform this function. They have the advantage of being able to measure the expression levels of thousands of genes at the same time. For purposes of discussion we will use the Affymetrix GeneChips.

A GeneChip microarray is a quartz wafer less than 1/2 inch square, on which millions of oligonucleotide sequences have been assembled using a photolithographic process.

GeneChips are specific to the genome of a particular organism (Ecoli, Aribidopsis, Homo Sapiens, etc. The oligonucleotide sequences consist of nucleotide chains of length 25 (25-mers). These chains are chosen to correspond to selected parts of genes, and these genes (ideally) cover the entire genome of the organism. These 25-mers are complementary to the sense strand of DNA and correspond to mRNA sequences.

Some 11-20 of these 25-mers target a specific gene and are referred to as perfect matches (PM). In addition, a 25-mer corresponding to each of the PMs is constructed

with a mismatch (MM) in the 13th base pair.

The 25-mers are built on the GeneChip in a pattern of dots as small as 11 microns in size. Each dot is referred to as a probe and the set of probes for a single gene are called probe sets. Also, each chip contains calibration probes. Each probe contains hundreds of the 25-mers with the same nucleotide sequence. The information about where these probes are on the chip is contained in an Affymetrix file with an extension of .chp. Meta information about the probe set such as gene name is contained in a (.gin) file.

Now, omitting most of the biochemical details (just know it's not as simple as it sounds), a sample of mRNA is extracted from the cells of an organism and is converted to a biotin labeled strand of complementary RNA (cRNA). The cRNA is complementary to the 25-mers on the GeneChip. When the GeneChip is exposed to this sample a process called hybridization occurs. During hybridization, complementary nucleotides line up and bond together via weak hydrogen bonds. The greater the concentration of a particular mRNA strand, the greater the number of bonds formed within one or more of the probes in the corresponding probe set, see Figure 2.2 for a depiction of a GeneChip.

Now, the number of those hybridized probes have to be counted. A fluorescent stain is applied to the GeneChip that bonds to the biotin and the GeneChip is processed through a machine that paints each pixel of the chip with a laser (a pixel is the minimum resolution of the laser and is a small part of a probe) causing that pixel to fluoress, the magnitude of which is measured. The results are stored in a (.dat) file containing the raw information about the GeneChip experiment.

The pixel locations and intensities are mapped and normalized into probe locations and these results are stored in a (.cel) file.

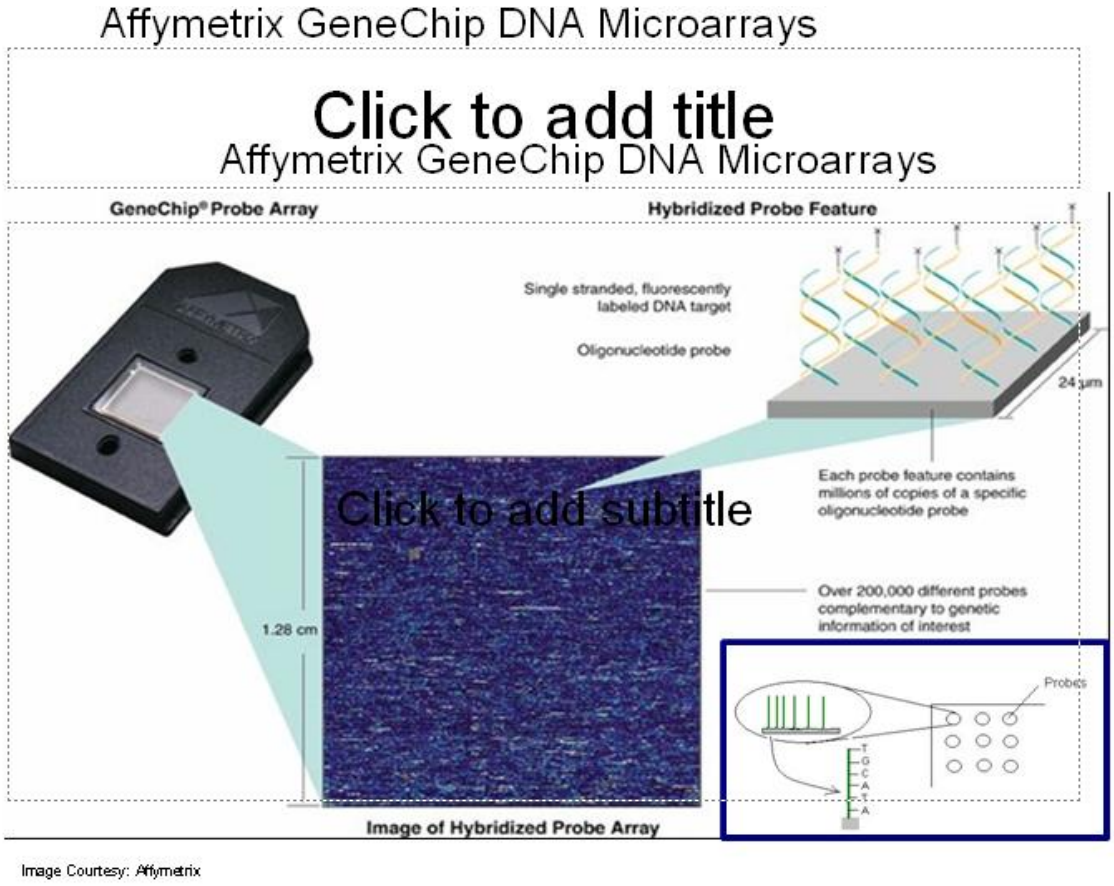


Figure 2.2: An Affymetrix GeneChip

All of this can be referred to as pre-analysis. We will pick up the discussion of analysis in the next section. For now, lets discuss microarray experiments in more detail.

A measurement by a single microarray is just a sample size of one. It's multiple sample sizes that are going to tell us biologically meaningful information. In general microarray experiments are of two kinds. (1) Studying a process over time. For example: We want to measure the gene response to a drug or a metabolic process. (2) Looking for differences between states. For example: Normal cells versus Cancer cells, which would have utility in disease identification.

In this paper our analysis will be concerned with an experiment of the first type. This involves multiple chips measuring responses in one or more organisms. Taken over time or during identifiable metabolic stages.

2.3 How is Microarray data analyzed?

After the pre-analysis is done we have a lot of raw data and still a long way to go. The first part is referred to as low level analysis and involves a three step process, see Bolstad [5] for a more detailed discussion.

1. The adjustment of probe level information in each array for background noise. This is due to variations in the cRNA preparation process.

2. The normalization of data across multiple arrays. Arrays cannot be created perfectly identical. Data distributions and calibration probes are involved in this.

3. Summarization of probe information into an intensity (expression level) for each probeset.

This gives an expression level for each gene in each array. The results of this analysis are stored in a (.chp) file.

At this point we have the following files.

- .cdf probe to probe set (gene) mapping
- .gin information about the probe set
- .dat pixel level data
- .cel probe level data
- .chp gene level data

The next level of analysis (referred to as High level) involves looking for relationships between genes via various forms of cluster analysis (hierarchical, K-means, principle component, spectral, etc.).

2.4 Normalization of data

Various statistical normalizations have been applied to the raw data to produce gene expression levels. Before performing spectral clustering **we propose normalizing the expression vectors to one**. Here's why.

Suppose, our microarray experiment examines a metabolic process at 5 distinct points. The processing of the raw data has produced an array of data where the rows are the genes and the columns (5 in number) are the expression levels. We are interested in clustering genes according to how similarly they are upgraded or downgraded. Now suppose we have two genes with expression levels (vectors) of $(1, 3, 4, 2, 1)$ and $(2, 6, 8, 4, 2)$, see Figure 2.3. The expression levels of these two genes show a strong correlation with respect to how they change through the metabolic process; the second just has a magnitue twice the first. We would like to see these genes clustered together.

But, looking at these within our 5D expression space, they may be far enough apart that when we generate edge weights they are not connected. Geometrically, we are saying that vectors close to the same straight line segment starting at the origin are correlated and should be clustered together. To accomplish this we normalize the data to vectors of length 1. This is equivalent to projecting the vectors onto a unit sphere around the origin. An example of this in the 2D case is illustrated in Figure 2.4. Without normalization (circles in green) there are three obvious groups. With this normalization (circles in red) there are two.

2.5 Disconnected Graphs

When defining edges between graph nodes (i.e. expression level vectors) we are typically dealing with thousands of nodes. To control the sparcity of the resulting

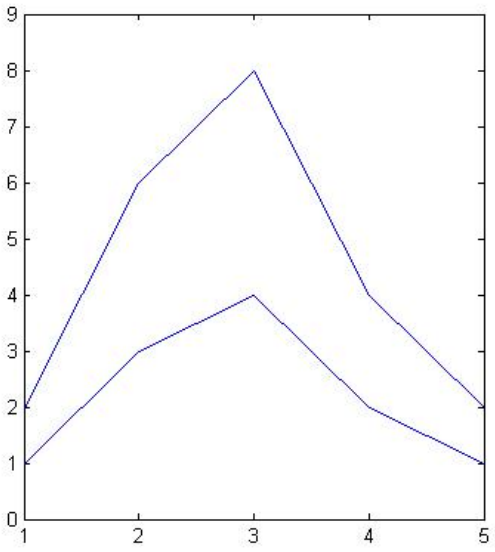


Figure 2.3: Genes with correlated expression levels

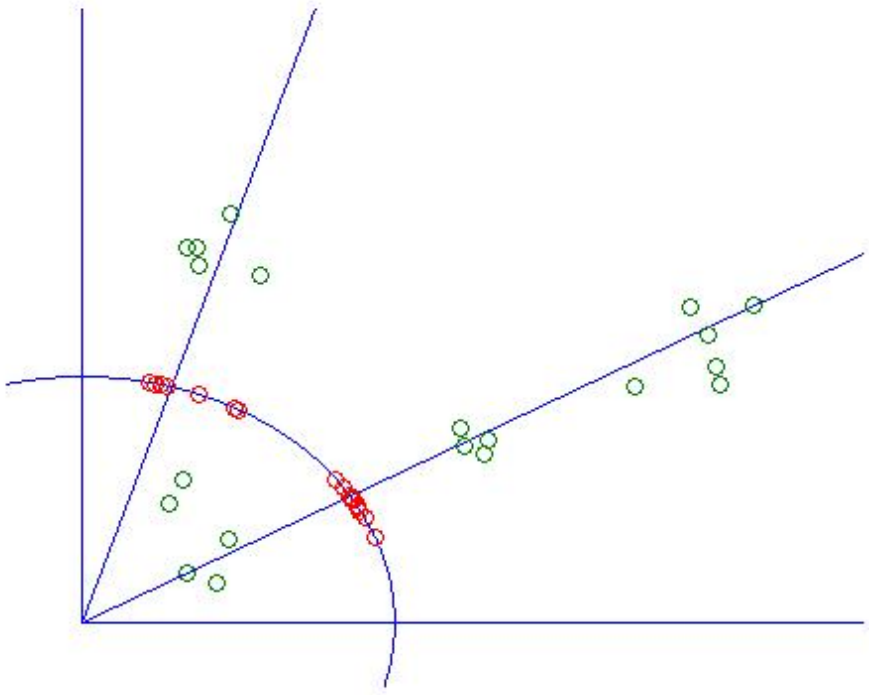


Figure 2.4: Effect of normalization on clusters

Laplacian we usually have to limit the number of edges in some way. For images we controlled the number of edges by defining our graph as a lattice with edges defined

by a 5 point stencil.

The data for microarrays does not fit into any preconceived graph architecture. The edge weight between two nodes is computed via a function such as an inverse gaussian distance which we will describe in more detail later. Then we apply a cutoff weight value. If the computed edge weight is less than this value the edge weight is set at zero. This effectively says there is no edge between those two nodes.

Selection of the limiting value can produce a graph that is almost complete or one that is so sparse few of the nodes are connected. Ideally we want a graph whose Laplacian is sparse (so it's easier to solve), has enough edges to retain the inherent structure of the data, and is connected (so we can apply the nodal theory previously developed). The first two conditions can be met by examination of the sparsity of the resulting adjacency matrix and adjustment of the cutoff weight. The last condition (connectivity) is not as easy.

As the edges are reduced we inevitably create multiple components in the resulting graph. The discrete nodal domain theorem required our graph to be connected. These disconnected components need to be identified. For components of size one (no edges connect to it) this is simple and efficiently accomplished by examination of the adjacency matrix; i.e. all values in a row are zero. For components of a larger size this is not as easy.

Graph algorithms such as a Breath First Search can be utilized to do this (see [12] page 534) but have the disadvantage of having run time of $\Theta(V+E)$. Generally there are a lot more edges than vectors and this can be quite large (potentially $\Theta(V^2)$).

An alternate method to find components using spectral methods was proposed by Ding, He, and Zha [19]. Their method is based on examination of the eigenvectors of the Laplacian with eigenvalue zero. We know the multiplicity of eigenvalue zero will be the number of components of the graph. Careful examination these eigenvectors can identify the individual components.

We propose an alternate method which is also spectral in nature and will easily integrate into the successive biclustering algorithm we will introduce later. This method starts with the addition of a connecting node to the graph and the definition of an edge of a small weight between the connecting node and every other node. Our graph is now connected, and this small perturbation to an unnormalized Laplacian is subject to an exact solution in terms of the original Laplacian for the problem of $Lv = \lambda v$. **We next give the solution to this problem in the following theorem**, and then apply it to an algorithm for extraction of components.

Theorem 2.1 *Let L be the unnormalized Laplacian of a graph G with n vertices, not necessarily connected. Let $\hat{G}$ be a graph formed by the addition of a vertex to G and an edge between this vertex and every other vertex. Let all of these edges have weight $b > 0$. Let $\hat{L}$ be the Laplacian of $\hat{G}$ and let $\mathbb{I}_n$ be the vector of all ones of size n. Then the eigenpairs of $\hat{L}w = \lambda w$ can be characterized as follows:*

1. $(0, \mathbb{I}_{n+1})$,
2. $(b, \left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right])$ *where* $Lv = 0v$ *and* $v \neq 0$ *and* $\sum v_i = 0$,
3. $((n+1)b, \left[\begin{smallmatrix} \mathbb{I}_n \\ -n \end{smallmatrix}\right])$,
4. $(\lambda + b, \left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right])$ *where* $Lv = \lambda v$ *and* $\lambda > 0$.

Proof:

By construction of $\hat{G}$ it is connected and so (1) follows.

We have $\hat{L} = \left[\begin{smallmatrix} L+D_b & B \\ B^T & nb \end{smallmatrix}\right]$ where $B = -b\mathbb{I}_n$ and D_b is the diagonal matrix with b on all diagonals. Let v be a solution of $Lv = 0v$ where $v \neq 0$ and the sum of the entries of v is zero. Then $\hat{L}\left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right] = \left[\begin{smallmatrix} Lv+D_bv \\ B^Tv \end{smallmatrix}\right] = \left[\begin{smallmatrix} bv \\ 0 \end{smallmatrix}\right] = b\left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right]$ which proves (2). Note that this case can only occur when G is connected.

We also have $\hat{L}\left[\begin{smallmatrix} \mathbb{I}_n \\ -n \end{smallmatrix}\right] = \left[\begin{smallmatrix} L\mathbb{I}_n+D_b\mathbb{I}_n+B(-n) \\ B^T\mathbb{I}_n+nb(-n) \end{smallmatrix}\right] = \left[\begin{smallmatrix} b\mathbb{I}_n+nb\mathbb{I}_n \\ -nb+nb(-n) \end{smallmatrix}\right] = (n+1)b\left[\begin{smallmatrix} \mathbb{I}_n \\ -n \end{smallmatrix}\right]$ which proves (3).

Suppose $Lv = \lambda v$ and $\lambda > 0$, then by the orthogonality of v to $\mathbb{I}_n$ we have $\hat{L}\left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right] = \left[\begin{smallmatrix} Lv+D_b v \\ B^T v \end{smallmatrix}\right] = \left[\begin{smallmatrix} \lambda v + bv \\ 0 \end{smallmatrix}\right] = (\lambda + b)\left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right]$ which proves (4).

Now, we have only to show that all possible eigenvalues have been accounted for and their eigenvectors are linearly independent.

If G is connected then by (1),(3) and (4) we have defined $n+1$ eigenpairs. The eigenvectors v in (4) are independent and orthogonal to $\mathbb{I}_n$. So, (1) and (4) define n linearly independent eigenvectors. For the same reason, $\left[\begin{smallmatrix} \mathbb{I}_n \\ -n \end{smallmatrix}\right]^T \left[\begin{smallmatrix} v \\ 0 \end{smallmatrix}\right] = 0$ and we have $n+1$ independent eigenvectors. Since, G is connected (2) does not apply.

If G is not connected then the multiplicity of the eigenvalue zero is the number of components of G. Let m be the number of components of G. For any two components, say G_j and G_k of G, we can define a vector v such that $Lv = 0$ and $\sum v_i = 0$. Just take $v_i = 1$ if $i \in G_j$ and $v_i = \alpha$ if $i \in G_k$ where $\alpha = |G_j|/|G_k|$. We can construct exactly $m-1$ such vectors which are linearly independent. So, by (1),(2), and (4) we have 1, $m-1$, and $n-m$ independent vectors for a total of n and then (3) adds the last.

■

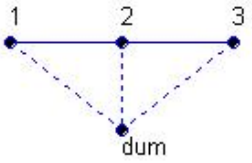
Corollary 2.2 *If $\hat{L}v = bv$ then $\sum v_i = 0$ and all values in v for any component of G will be the same.*

Proof: We note that our choice of eigenvectors for (2) in the proof of 2.1 span the eigenspace of eigenvalue b. Let us denote these as w^i. Any other eigenvector in the eigenspace will be of the form $v = \sum_i \beta_i w^i$. We have $\sum_j v_j = \sum_i \beta_i \sum_j w^i_j = 0$. Finally, since the values of components in any w^i are the same by construction, this must also be true in v. ■

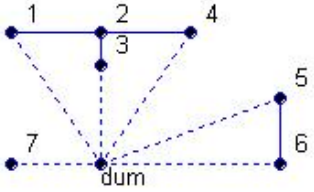
The importance of this corollary is that an eigenvector of eigenvalue b must be a Fiedler vector of $\hat{L}$ since $0 < b < \lambda + b$. Choosing any one of these vectors to use with bisection we see that all values in v for any component of G must be the

same and this implies a component cannot be split into separate clusters by finding an r-weak sign graph. As such, a Fiedler vector of L will provide a way to separate components of the graph G. Successive, biclustering will naturally separate out all of the components.

The examples in Figure 2.5 illustrate these theorems.



node	eigenvectors			
1:	1	1	-1	1
2:	1	1	0	-2
3:	1	1	1	1
dum:	1	-3	0	0
eval:	0	4b	1+b	3+b



node	eigenvectors							
1:	1	0	1	1	-1	-1	0	1
2:	1	0	1	1	0	0	0	-3
3:	1	0	1	1	1	0	0	1
4:	1	0	1	1	0	1	0	1
5:	1	1	0	1	0	0	-1	0
6:	1	1	0	1	0	0	0	0
7:	1	-2	-4	1	0	0	0	0
dum:	1	0	0	-7	0	0	0	0
eval:	0	b	b	8b	1+b	1+b	2+b	4+b

Figure 2.5: A connected and unconnected graph and the effect of a connecting node (dum) on their eigenpairs

For purposes of clustering we want the option of using the normalized Laplacians: $L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ and $L_{rw} = D^{-1}L$. Theorem 2.1 does not apply to them and deriving exact solutions to the perturbation of L by a connecting node has proven to be elusive. Perturbation of the algebraic connectivity (2nd eigenvalue) is addressed in [34], but the issue of perturbation of the Fiedler vector is more difficult. But, from numerical experiments we can make the following observations.

Assume b (the perturbation weight) is small relative to the minimum of the weights in L. For the case of a connected graph, let v be the Fiedler vector of the unperturbed problem $L_{rw}v = \lambda v$. The Fiedler vector of the perturbed problem, $\hat{L}_{rw}w = \lambda_p w$ or equivalently $\hat{L}v = \lambda_p Dv$, is of the form $w = \left[\begin{smallmatrix} v+\delta \\ \alpha \end{smallmatrix}\right]$ where vector δ is

small relative to the values of v and $\lambda_p \approx \lambda$. The perturbed Fiedler vector of G with the connecting node removed should then give us the same partition as the original vector.

For the problem of the symmetric Laplacian, $L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, we have similar results since eigenvalues of L_{rw} and of L_{sym} are the same, and if v is an eigenvector of L_{rw} then $w = D^{\frac{1}{2}} v$ is an eigenvector of L_{sym}. Note that one implication of this is that the eigenvector of eigenvalue zero is no longer a constant vector.

For the case of a graph with multiple components, the Fiedler vector of the perturbed matrix will have values distributed according to the components of the graph. For the L_{rw} Laplacian the values of any component will be nearly a constant. Each component may or may not have the same constant, but the eigenvector will include both positive and negative values and so a partition of the vector will yield non-empty partitions where a component cannot be split into separate partitions (but could be in both).

We can now define the following algorithm for identification of components and successive biclustering. This will be valid for the unnormalized Laplacian and we have some intuition based on numerical experiments that it is valid for the normalized Laplacians.

Step 1 Reduce the adjacency matrix of G by all singleton components.

Step 2 Select a value for b.

Step 3 Add a connecting node to G as described above to create $\hat{G}$.

Step 4 Construct the Laplacian and solve for the first few eigenpairs.

Step 5 Excluding the zero eigenvalue identify the next smallest eigenvalue.

Step 6 Use the eigenvector of this eigenvalue to partition the graph.

Step 7 If the partition results in an empty partition then exclude that eigenvalue and using the next smallest eigenvalue goto Step 6.

Step 8 Using these components, repeat the process starting with step 3 until a sufficient number of clusters has been identified.

Comments:

If we only want to identify all components then we can use the unnormalized Laplacian and stop in step 5 when the eigenvalue is not b.

In Step 1, we choose to eliminate all singletons. This is not necessary, but is done because it is easy and cheap (the diagonal value of a singleton is zero) and the singletons don't carry much information about their relation to other vectors (genes). In the program to follow we just put them all in a partition by themselves. This can significantly reduce the size of the Laplacian and speed up the analysis.

We choose b to be .001 times the weight limit. Since, all edge weights must be greater than this limit, this makes $b << $ all other edge weights.

One can ask, "Why bother with the connecting node?". Just examine the zero eigenvectors of L. There are some practical reasons

If the graph G is connected we don't know this and we still have to examine the zero eigenvector. Ideally this would have an eigenvector with identical values or for the symmetric Laplacian one which has all positive values. Then we know the graph is connected. Numerical solutions will give eigenvectors that are not exact. When the solutions are normalized and the dimensionality of L is very large this can result in values close to zero including both positive and negative values. This will result in a bad partition of the graph. Adding a connecting node allows us to identify and ignore the zero eigenvector.

In step 6 we partition according to the r-weak sign graphs. If r is too large this can result in two identical partitions. This is a major type of error and should terminate

processing less successive iterations endlessly replicate the same cluster. Since we will partition by r-weak sign graphs the resulting partitions may intersect. As discussed earlier, we consider this an advantage of the technique.

2.6 Successive BiClustering

In the previous section we developed an algorithm for dealing with graphs with multiple components. This algorithm not only deals with multiple components but also handles partitioning of a connected graph. We want to incorporate this algorithm into another for successive biclustering applied to our original graph. We have two problems to solve. (1) Which partition do we apply the algorithm to next?, and (2) When do we stop?

To answer the first question we need a measure of the internal connectivity of a graph. We would say a graph is fully connected if the graph were complete and have the same weight on all of the edges. In this case partitioning of it would not be meaningful. All nonzero eigenvalues are the same and every partition creates another set of complete graphs. On the other hand a graph with no edges has no internal connectivity.

We developed the following measure which reflects these two extremes.

Definition 2.3 *The internal connectivity of a graph is* $\frac{|G|}{n(n-1)}$*, where* $|G|$ = *sum of weights of all edges of* G*.*

This is based on the observation that the number of edges in a complete graph with n vertices is $n(n-1)/2$. For convenience we have dropped the constant factor 2 in the definition of connectivity.

We will select the next cluster to partition by choosing the one with the lowest internal connectivity. Note that while this is motivated by mathematical considerations it is in essence an adhoc procedure.

Figure 2.6 illustrates this process. The example is of a graph formed from 4 normal distributions of data. These are centered at positions (2,3), (8,9), (2,9), and (5,6).

Weights are generated via a gaussian weight function with a minimum weight. This results in some isolated vertices (singleton clusters, in dark blue) and one cluster of 2 vertices (in red). Successive biclustering roughly identifies the original distributions. Note that there may be vertices in more than one cluster but these are not identified in the plots generated here.

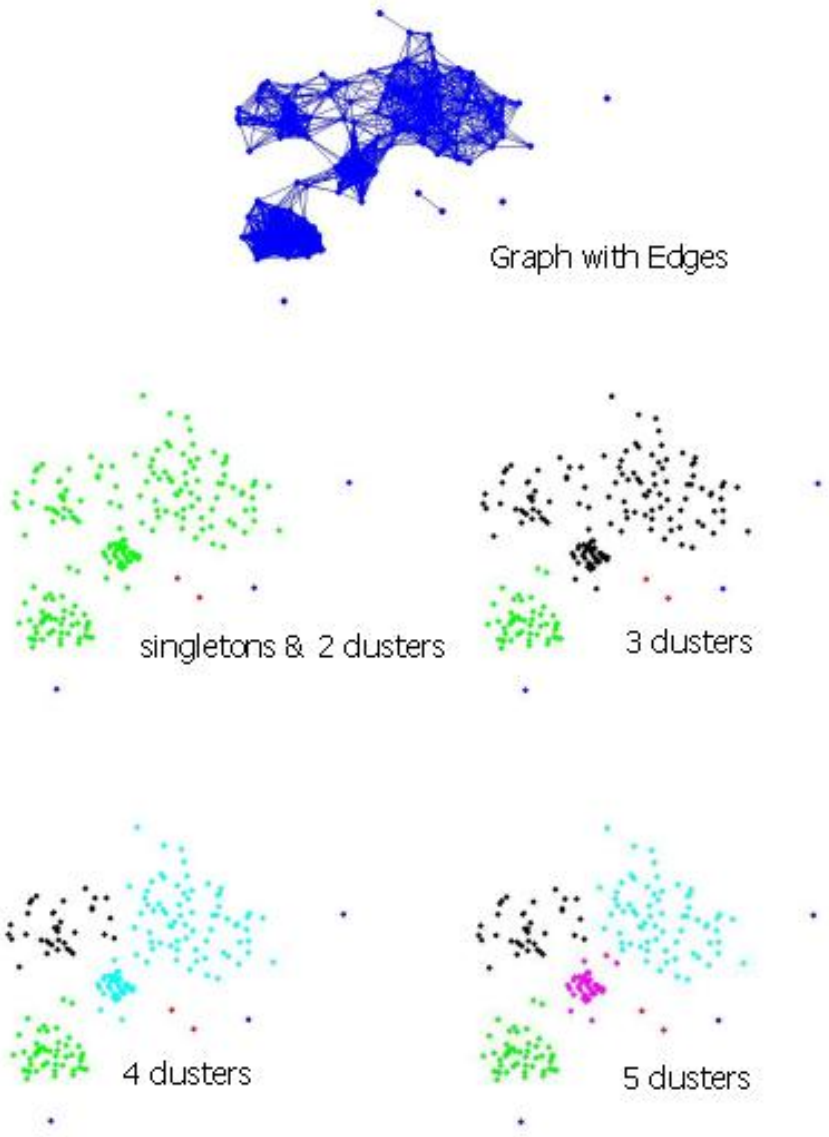


Figure 2.6: An example of successive bicluster

The solution to the second question (about stoping criteria), is also adhoc in nature. We could choose to proceed until all partitions achieved some minimum connectivity. We could also choose to proceed till a fixed number of clusters is produced, or until we achieve a full hierarchical breakdown to one or two vertices per cluster. In either case we have to make an a-priori judgement about these values.

Some authors have suggested using eigenvalue jumps to determine the number of clusters to find [49]. This works well if we are looking for disconnected or nearly disconnected clusters. For microarrays, however, we do not expect the resulting graph to be that simple. We choose here to explore the data by looking for 16 clusters. This choice is arbitrary, and would only be a starting point for analysis. After examination of the results, this choice would be modified for succeeding runs.

2.7 Weight construction

In the software, several techniques are implemented to compute edge weights.

Inverse Gausian distance is computed via the function e^{-sc*d^2} where sc is a scaling factor which defaults to 1 and d is the Euclidian distance between two gene expression vectors; i.e. the norm of the difference between the two vectors. This is then limited to zero by a radius value; i.e. if the weight is less than the radius then the edge weight is zero. This is done to introduce sparsity to the Laplacian.

A fully connected (complete) graph can be produced. Here all of the edge weights are one. Edges can be predefined and in this case the edge weights are one. These techniques are implemented to analyze certain test graphs.

The technique we will use for microarray data is a Euclidean distance limited by a radius value. When the distance is greater than the radius value the weight will be zero, otherwise it is one. This was chosen because we will be projecting the gene expression vectors onto the unit sphere and we do not have to handle large variation in distances. We note that angles could be used here as a measure of distance.

2.8 "Toy" experiments

In this thesis and in many of the papers referenced, so called "Toy" data is used to analyze the algorithms presented there. The use of this data is done for two reasons.

1. validation of techniques

2. validation of software

Toy data provides test examples where the clustering results are well defined, or at least roughly defined intuitively. For "Real world" data the clustering may not be easily recognized. If it could be there would be no need to do the analysis. We can also construct Toy data which should represent specific situations the software is supposed to handle.

The Toy data gives us an objective way to test our techniques and their implementation in software. This is not always as simple as it sounds. Debugging of the software for successive bicluster revealed several problems that required adjustments to the algorithm and enhancements to the theory supporting them. Software testing was thus an integral part of the overall process of doing the mathematical research.

Once the Toy data has served it's purpose, we can now use the software to examine "real world" data. Some of the mathematics reviewed and developed here is fairly advanced and some rather simple. The ultimate purpose of what we have developed is not just an exercise in pure mathematics, but hopefully has applicability to real world data and problems. In the next section we will apply these techniques to real microarray data.

2.9 Analysis of real microarray experiments

We will analyze yeastvalues taken in a time sequence over a metabolic shift from fermentation to respiration. The source of this data is the Matlab demo "Gene

Expression Profile Analysis" from the Bioinformatics Toolbox. The data represents a set of 7 microarrays where all of the low level analysis has occurred prior to ours to produce a set of gene expression levels. Further, genes with very low expression levels have been filtered out. This leaves 822 genes represented in an 822x7 matrix.

One of the problems we have to face for multi dimensional data is how to represent the results. Toy experiments were all 2D so these had a natural way to graph clusters. We could do something similar for 3D but with difficulty. Microarray data will usually have dimensions larger than this.

Listing the genes in groups is necessary because this is what a biological scientist is going to need to evaluate the cluster. This is always possible by cross referencing a genes matrix row number to a meta data list of gene descriptions. The listing of the clusters for this experiment is given in Appendix D.

We also desire a graphical way to represent the results. We want to visually confirm how "good" the cluster is and identify interesting characteristics of the data. The technique we choose is to graph a genes expression level (unnormalized) against its microarray number(1 to 7 for our test data). All genes within the same cluster are placed in the same graph. Multiple graphs are produced; one for each cluster, see Figure 2.7. The set of genes with no edge connection to any other gene are not represented. The number of entries in the cluster and the clusters connectivity (see Definition 2.3) is printed above each graph.

This analysis was performed using the Matlab eigs function solving the $Lv = \lambda v$ problem, vectors normalized, Euclidian distance function with a radius cut off of .2, and a value of $r = .0000001$ used to compute r-weak sign graphs.

2.10 The Software

The software is implemented in Matlab. The program to perform the analysis is coded as a function and listed in Appendix A. A program to invoke the function for the examples in this paper is listed in Appendix 2.

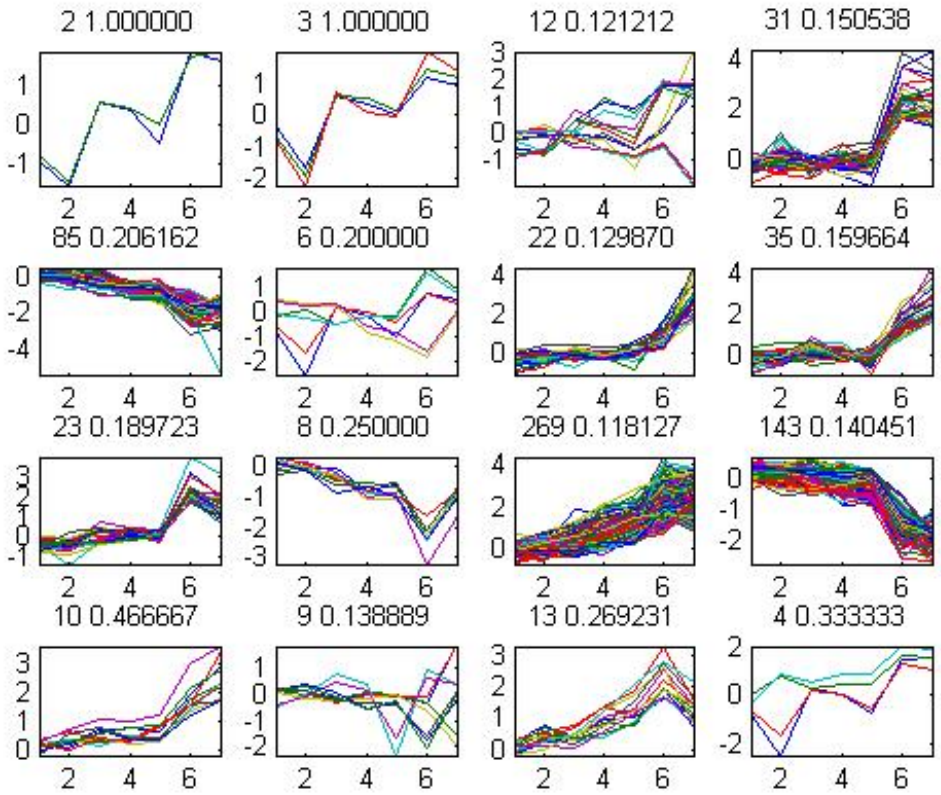


Figure 2.7: Microarray data clustering result

The function arguments consist of 4 fixed parameters and a variable argument list. These are an array of the vertices to cluster, the number of clusters to produce, a string defining the computational options, and a string defining the graphs to produce. The variable arguments define Radius, Sigma, Scale, Mass, Edges, and Rweak the value to use for determination of sign graphs. The first 2 arguments are required. The remainder have defaults if not entered.

The function returns a cell array where the clusters are defined and a vector which records a connectivity value for each cluster. The first cluster in the cell array is all of the isolated vertices. A vertex can appear in more than one cluster.

Two Methods for determining clusters are provided in the MATLAB program; successive bicluster and kmeans. Kmeans clustering is described in Luxburg [56] and

involves applying the kmeans algorithm to the row vectors of a matrix of the first k eigenvectors of the graph Laplacian. It works well provided clusters are nearly disconnected and there are not lots of isolated vertices. It produces a classic partitioning where a vertex can only be in a single cluster.

Parameters and their options are detailed in the program comments in Appendix A.

The computation of eigenvalues is based on the Matlab eigs or eig function. The eigs function should be used for large sparse matrices. The eigenvalue problem can be rcut $Lv = \lambda v$, mass $Lv = \lambda Mv$, or ncut $Lv = \lambda Dv$.

The organization of the program is as follows:

- Analyze parameters, set defaults,
- Define weight matrix
- Define connecting node edge weight
- Do kmeans clustering if requested
  - let k be the number of clusters to solve for
  - solve the eigenvalue problem
  - identify the eigenvectors of the k lowest eigenvalues (not including zero)
  - perform k-means on the row vectors of these eigenvectors
  - define these clusters in the cell array partition
- Do successive biclustering if requested
  - initialize the cell array partition with one entry containing all vertices
  - split out the unconnnected vertices into their own partition entry and define its connnectivity as 999
  - select the other partition entry for processing

- WHILE the number of partition entries is less than the number requested do
- bicluster the selected partition entry based on fiedler vector
  - solve the eigenvalue problem
  - identify lowest nonzero eigenvalue and it's associated eigenvector
  - B: split the partition into two clusters based on the r-weak value and the eigenvector
  - if both partition are identical to the original then error the program
  - if either partition has zero entries then skip to next eigenpair and goto B:
- split the selected partition into two entries based on bicluster results
- compute the connectivity of all partition entries
- select the partition entry with smallest connectivity
- end WHILE

Various plots are produced throughout the program as requested in the Plot parameter.

3. Blopex

3.1 Introduction

The software package Block Locally Optimal Preconditioned Eigenvalue Xolver (BLOPEX) was introduced in 2005. It has recently been upgraded to Version 1.1. BLOPEX implements the Locally Optimal BLock Preconditioned Conjugate Gradient (LOBPCG) method for solution of very large, sparse, symmetric (or Hermitian) generalized eigenvalue problems. Version 1.1 adds support for complex matrices and 64bit integers.

The generalized eigenvalue problem for large, sparse symmetric and Hermitian matrices occurs in a variety of traditional problems in science and engineering; as well as more recent applications such as image segmentation and DNA microarray analysis via spectral clustering. These problems involve matrices that can be very large (dimension $> 10^5$) but fortunately are sparse and frequently we only need to solve for a few smallest or largest eigenpairs. This is the function of the BLOPEX software.

BLOPEX has been previously described in [44] and [41] . This paper seeks to give a more detailed description of the software than has previously been supplied. This software includes not just the implementation of the LOBPCG method but interfaces to independently developed software packages such as PETSc [1], Hypre [2], and MATLAB [3].

The remainder of this chapter organized as follows. We will start in section 3.2 by a general discussion of the problem. Review some of the other software available for the problem in section 3.3. Present the LOBPCG method in section 3.4. Section 3.5 then covers the BLOPEX software. BLOPEX is available via a new Google Source

[1]PETSc (Portable Extensible Toolkit for Scientific Computation) is developed by Argonne National Laboratory

[2]Hypre (High Performance Preconditioners) is developed at the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory

[3]MATLAB is a product of The MathWorks^{TM}

site and this is covered in section 3.6. We discuss the environments BLOPEX has been tested on in section 3.7. Give some numerical results in section 3.8, and wrap up in section 3.9.

3.2 The Problem

We seek solutions to the generalized eigenvalue problem $Ax = \lambda Bx$ where A and B are real symmetric or complex Hermitian. B must be positive definite. A and/or B may be defined as a matrix or be supplied in functional form.

Note that the requirement that B be positive definite implies all eigenvalues are finite and the symmetry of A and B imply all eigenvalues are real.

We emphasize that A and B need not be supplied in matrix form, but can be defined as functions.

Problems of this type arise from discretizations of continuous boundary value problems with self-adjoint differential operators [43]. We often only need the m smallest eigenvalues or eigenpairs; where m is much less then the dimension of the operator A. We don't usually need solutions to high accuracy, since the discretization of the problem is itself an approximation to the continuous problem.

The large dimensionality of the problem precludes solution by direct (factorization) methods. Thus the need for iterative methods. But iterative methods can have slow convergence and so we require a preconditioner [42]. The choice of preconditioner is separate from the choice of iterative method.

We will be using the LOBPCG iterative method (see section 3.4). Preconditioners are supplied to BLOPEX by the calling programs. This and the interfaces to PETSc and Hypre make possible the use of high quality preconditioners.

3.3 Current Software

There are a number of existing software packages for solutions of large, sparse eigenvalue problems. We discuss two of these that have been previously described in ACM transactions.

Anasazi [3] is a package within the Trilinos framework, written in C++ which uses object oriented concepts. It implements 3 block variants of iterative methods: LOBPCG, Davidson, and Krylov-Schur.

Anasazi solves for a partial set of eigenpairs of the generalized eigenvalue problem. It uses a preconditioner which must be supplied by the user. Starting with Trilinos 9.0 there is interoperability with PETSc.

Preconditioned Iterative Multi Method Eigenvalue (PRIMME) [54] was released Oct 2006. It implements the JDQMR and JD+k methods to solve for a partial set of eigenvalues of the problem $Ax = \lambda x$. It does not currently handle the Generalized Eigenvalue problem. Written in C it has an emphasis on being "user friendly", by which is meant a minimal parameter set can be used to obtain solutions without extensive tuning or knowledge on the part of the user. More sophisticated users can utilize an extended set of parameters to tune the performance.

PRIMME can handle real and complex numbers and orthogonality constraints. The preconditioner is supplied by the user. Interfaces to PETSc and Hypre are not mentioned and presumably not available.

Neither PRIMME or Anasazi mention interfaces to Matlab.

By contrast BLOPEX:

- handles both real and complex numbers
- is written in C
- has similar parameters as PRIMME
- has interfaces to PETSc, Hypre, Matlab, and stand alone serial interfaces
- PETSc and Hypre allow for use of high quality preconditioners

3.4 LOBPCG

To solve for a single eigenpair of the problem $Ax = \lambda Bx$ the LOBPCG iterative method can be described as a 3 term recurrence formula as follows:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} x^{(i-1)}, \tag{3.1}$$

where

$w^{(i)} = Tr^{(i)}, r^{(i)} = Ax^{(i)} - \lambda^{(i)} Bx^{(i)},$

$\lambda^{(i)} = (x^{(i)}, Ax^{(i)})/(Bx^{(i)}, x^{(i)})$ the Rayleigh quotient, and,

T is a preconditioner for the matrix A.

The values $\tau^{(i)}$ and $\gamma^{(i)}$ in (3.1) are chosen to minimize $\lambda^{(i+1)}$ within the subspace $span\{w^{(i)}, x^{(i)}, x^{(i-1)}\}$. This minimization is done via the Rayleigh–Ritz method as described in [51]. The preconditioner T should be linear, symmetric, and positive definite.

Use of $x^{(i)}$ and $x^{(i-1)}$ as basis vectors for $span\{w^{(i)}, x^{(i)}, x^{(i-1)}\}$ can lead to ill-conditioned Gram matrices in the Rayleigh-Ritz method, because $x^{(i)}$ can be very close to $x^{(i-1)}$.

The effect of basis vectors is a non-trivial problem discussed in [35]. An improvement on the basis used in (3.1) was proposed by [43]. This replaces $x^{(i-1)}$ with $p^{(i)}$ as follows:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} p^{(i)}, \tag{3.2}$$

and for the next iteration

$p^{(i+1)} = w^{(i)} + \gamma^{(i)} p^{(i)}$ and the other terms are as in (3.1).

In this case, it can be shown that $span\{w^{(i)}, x^{(i)}, p^{(i)}\} = span\{w^{(i)}, x^{(i)}, x^{(i-1)}\}$. So, the two iterative problems (3.1) and (3.2) are mathematically equivalent but (3.2) is more numerically stable.

When more than one eigenpair is to be computed a block version of LOBPCG is used. To compute the m smallest eigenpairs we want to apply the Rayleigh-Ritz method to the subspace spanned by $\{x_1^{(i)}, w_1^{(i)}, p_1^{(i)}, \cdots, x_m^{(i)}, w_m^{(i)}, p_m^{(i)}\}$. This gives m Ritz vectors $x_j^{(i+1)}$ as estimates for the m smallest eigenvectors with estimates for eigenvalues given by their Rayleigh quotients.

We note that the choice of block size m is in part problem dependent and in part a tuning consideration. This is discussed in some detail in [41].

3.5 BLOPEX Software

The BLOPEX software provides functions to the user for solution of eigenvalue problems as described in section 3.2. The software external to BLOPEX which the user writes must do the following:

- setup matrices or functions for A and B
- setup the preconditioner T
- provide functions for matrix-vector operations
- call LOBPCG solver in BLOPEX

BLOPEX software is in written in C. C was chosen since it provides for ease and versatility of interfacing with a wide variety of languages including C, C++, Fortran, and Matlab. This makes BLOPEX highly portable.

BLOPEX can be logically separated into two parts. The first part implements the LOBPCG algorithm. We refer to this as the "abstract" code. It contains the functions called by the user as well as a number of utility functions needed internally. We will discuss this in detail in section 3.5.2.

The second part is code which provides functions for interfacing with software packages such as PETSc, Hypre, and Matlab. One challenge for all eigensolver software is the necessity of supporting multiple diverse formats for sparse matrices and vectors. Functions for accessing matrix (vector) elements and doing matrix vector operations are inherent in the calling software and BLOPEX will have need to access these routines. This access occurs via the specific interface functions. We will describe the interfaces in section 3.5.3.

3.5.1 Structure

Figure 3.1 shows a high level overview of BLOPEX and how it fits with the calling software. The Driver is software written by the user which calls the LOBPCG solver in BLOPEX abstract. The driver can be written in numerous external environments such as PETSc, Hypre, Matlab, etc. BLOPEX provides a number of sample Drivers which are described in section 3.5.3.

The Driver will use macros, commands, or functions from it's environment to define matrices, vectors, and matrix/vector operations. These are communicated to the LOBPCG solver via parameters, (see section 3.5.2.3). To access these external environment matrix/vector routines BLOPEX supplies interfaces.

These interfaces package data as multivectors (see section 3.5.2.2) to pass to the LOBPCG solver and provide functions to convert parameters in formats defined within BLOPEX to parameters specific to the external environment functions.

BLOPEX requires LAPACK functions or equivalents to perform orthonormalization and solve the generalized eigenvalue problem for the Gram matrices in the Rayleigh-Ritz method. These can be the standard LAPACK functions dsygv and dpotrf for real, or zhegv and zpotrf for complex numbers. Equivalents from the ACML, MKL, or ESSL libraries can be used. The addresses of the routines to use are passed by the Driver to BLOPEX. If the parameters are not the same as the standard

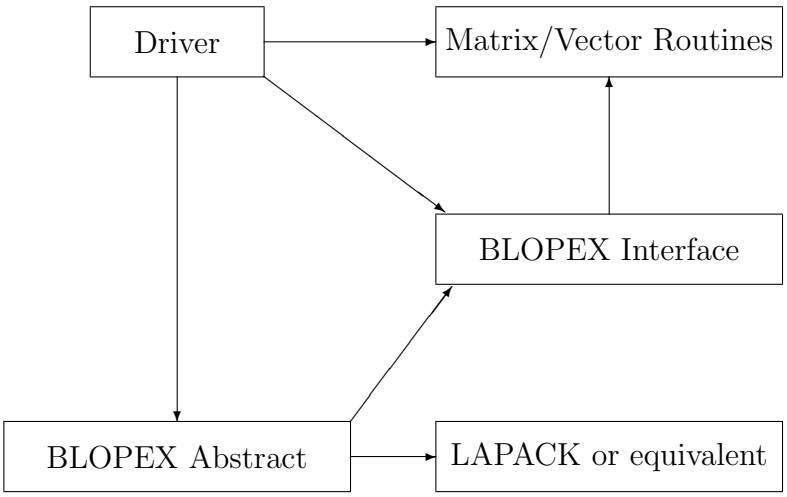


Figure 3.1: Structure of BLOPEX Functions

LAPACK funtions then a function must be coded to do parameter conversions.

3.5.2 Abstract Code

This is the solver. It consists of three modules `lobpcg.c`, `multivector.c`, and `fortran_matrix.c`. Small matrices and vectors that arise as result of Rayleigh-Ritz Method are kept internal to the abstract code in Fortran column major order and processed via routines in `fortran_matrix.c`.

Two routines in `lobpcg.c` are callable by Drivers. `lobpcg_solve_double` and `lobpcg_solve_complex`. These routines setup a function interpreter which is a list of function addresses. The functions in `multivector.c` and `fortran_matrix.c` are specific to double (real) or complex numbers. These two functions then call `lobpcg_solve` where the LOBPCG algorithm is implemented. As a result BLOPEX does not have to be compiled specifically for complex or real numbers.

The functions in `multivector.c` provide for conversions and operations between matrices in Fortran format and multivectors with pointers to matrices and vectors in the external environment format. These functions in turn call interface functions to access these matrix/vector operations.

3.5.2.1 Algorithms

The steps of the Block LOBPCG algorithm as implemented in BLOPEX follow with detailed comments on various steps appearing afterwards. We list only the major parameters here but we give a complete list and more explanation in section 3.5.2.3.

Input:

- X m starting vectors (in block form)
- A matrix or address of function to compute $A * X$
- B matrix or address of function to compute $B * X$
- T preconditioner (operator, and data)
- Y constraint vectors (in block form)

Output:

- X m computed eigenvectors
- Λ m computed eigenvectors

Algorithm:

1. Apply constraints Y to X
2. B-orthonormalize X
3. $[C, \Lambda] = RR(A, B, X)$ Apply Raleigh-Ritz Method
4. $X = X * C$ Compute Ritz vectors
5. $J = [1, \cdots, m]$ Initialize the index set of active residuals

6. $for \;\; k = 1, \cdots, MaxIterations$

7. $R_J = B * X_J * \Lambda - A * X_J$ Compute Residual vectors

8. Compute norms of residual vectors

9. Check residual norms for convergence

10. Exclude converged vectors from index J (soft locking)

11. if all vectors have converged then stop

12. $W_J = operatorT(R_J, dataT)$ Apply preconditioner to residuals

13. Apply constraints Y to W_J

14. B-orthonormalize W_J

15. if $k > 1$

16. B-orthonormalize P_J

17. basis for RR is $S = [X \; W_J \; P_J]$

18. else

19. basis for RR is $S = [X \; W_J]$

20. end

21. $[G, \Theta] = RR(A, B, S)$ Apply Raleigh-Ritz Method

22. $C = G(1 : m, :)$ Get columns of G corresponding to X

23. $\Lambda = \Theta(1 : m)$ Get eigenvalues corresponding to X

24. if $k > 1$

25. Partition $C = \begin{bmatrix} C_X \\ C_W \\ C_P \end{bmatrix}$ according to columns of X, W_J, and P_J

26. $P = W_J * C_W + P_J * C_P$

27. else

28. Partition $C = \begin{bmatrix} C_X \\ C_W \end{bmatrix}$ according to columns of X and W_J

29. $P = W_J * C_W$

30. end

31. $X = X * C_X + P$

32. end

Comments:

(1) Constraints Y are previously computed or known eigenvectors. For example, the vectors of all ones is an eigenvector of the smallest eigenvalue of a graph Laplacian. So we can choose this as a constraint and then force all vectors of X to be B-orthogonal to Y. In this case LOBPCG solves for the next m smallest eigenvalues. We apply constraint Y to X via replacing X with the difference of X and the B-orthogonal projection of X onto the subspace generated by Y; that is $X = X - Y * (Y^T * B * Y)^{-1} * ((B * Y)^T * X)$.

(2) B-orthonormalize X using Cholesky factorization; that is $R = chol(X' * B * X); X = X * R^{-1}$. Block vector X must be composed of linearly independent vectors to being with else orthonormalization will fail.

(3) We adopt the following notation: $[G, \Lambda] = RR(A, B, S)$ to specify the Rayleigh-Ritz method which finds eigenvectors G and eigenvalues Λ of the generalized

eigenvalue problem $(S^TAS)G = \Lambda(S^TBS)X$. S^TAS is referred to as the gramA matrix and S^TBS as the gramB matrix. S is given in block form such as $[X\ W\ P]$ where X, W, and P are block vectors. S forms a basis for the subspace to find estimated eigenvectors in. Note that the eigenvectors G are chosen to be B-orthogonal.

(4) Note that the B-orthonormality of the Ritz vectors X is preserved.

(5),(9),(10) Initially all residuals of X are active. As we iterate this will change as their norms converge towards zero. When one of the vectors in X converges to within a prescribed tolerance it is removed from the index. This we call soft locking. All of X will remain as part of the basis for the next iteration. However, only the residuals (denoted by R_J and CG step vectors (denoted by P_J) will be used to create the new subspace. All of X is retained on the expectation that converged vectors in X will continue to improve.

(12) To apply the preconditioner to R_J, a call to a routine T provided by the Driver is done. This function is usually coded in the Driver and must have parameters of the form `void operatorT(void * dataT, void * R, void * W)` and must be able to handle W and R as block vectors. The code for `operatorT` is highly dependent on the Drivers environment.

(13),(14) We transform the preconditioned residual block vector W_J to be B-orthonormal to the constraint Y and the vectors of W_J to be B-orthonormal to each other.

(15) For $k = 1$ we do not have the first P_J yet. It is first computed in (29) and in (26) there after.

(17),(19) For the 1st iteration the basis for Rayleigh-Ritz is $S = [X\ W_J]$. Note that X and W_J are B-orthonormal with respect to their own vectors, but X is not necessarily B-orthonormal to W_J. Consequently, the symmetric Gram matrices take

the form

$$gramA = S^T AS = \begin{bmatrix} \Lambda & X^T AW_J \\ . & W_J^T AW_J \end{bmatrix}$$

and

$$gramB = S^T BS = \begin{bmatrix} I & X^T BW_J \\ . & I \end{bmatrix}$$

where the dot notation indicates the symmetry of the matrix. Note that $X^T AX = \Lambda$ since X is B-orthonormal.

For subsequent iterations the X, W_J, and P_J blocks are B-orthonormal within their blocks but not between them. So the Gram matrices are

$$gramA = S^T AS = \begin{bmatrix} \Lambda & X^T AW_J & X^T AP_J \\ . & W_J^T AW_J & W_J^T AP_J \\ . & . & P_J^T AP_J \end{bmatrix}$$

and

$$gramB = S^T BS = \begin{bmatrix} I & X^T BW_J & X_T BP_J \\ . & I & W_J^T BP_J \\ . & . & I \end{bmatrix}$$

Finally, we note that computations for the components of the Gram matrices are optimized in the code, so they are not computed directly from the basis vectors. For clarity these details have been omitted.

(26), (29) Computation of the block vector P corresponds to $p^{(i+1)}$ in equation 3.2. Note that P has the same number of rows as X.

(31) Finally, we compute a new X which is just the Ritz vectors $X = S * C$.

3.5.2.2 Data Types

BLOPEX defines several data types which are described here.

To deal with block vectors in various diverse formats BLOPEX defines the structure `mv_MultiVector`. This contains the following fields.

- A pointer to another structure that defines the vectors. How these vectors are formatted depends on the interface. For example in the PETSc interface it points to another structure `mv_TempMultivector` which contains fields that define the number of vectors, active mask and a pointer to an array of pointers, each of which point to a Vec PETSc variable. Interfaces for Hypre, Serial, and Matlab have different but similar structures. Creation of variables of `mv_MultiVector` type is done by routines in `multivector.c`.

- An integer variable that is set to 1 when data for the pointer defined above is allocated. This is to aid in deletion of the multivector when we are finished with it.

- A pointer to a structure `mv_InterfaceInterpreter` which is a list of function addresses. These pointers are set to the appropriate interface functions.

An `mv_Multivector` variable such as parameter X then encapsulates the data and interface functions to manipulate the data.

The second data type is to deal with matrices. This also depends on the interface. Matrices do not have to be manipulated like block vectors. Typically they are involved in some matrix vector operation and just need to be passed to the appropriate interface routine which is specified via the interpreter in the associated block vector.

So it is possible we only need to pass a pointer to the matrix as a parameter in the form it appears in the external environment. This is what is done for the Matlab interface. Other interfaces take a different approach. The PETSc interface includes in a multivector like structure, variables for A, B, and KSP solver and then passes this as the parameter for both A, B, and preconditioner data. The operatorA, operatorB, and operatorT functions then use the appropriate variable.

Internally the BLOPEX abstract code uses data type `utilities_FortranMatrix` to define Fortran style matrices. This includes variables for global column height,

column height, row width, pointer to position $(1, 1)$ of the matrix, and an ownsData variable. The global height can be different from the current height because we overlay in memory blocks of the Gram matrices to optimize their computation.

For BLOPEX version 1.1 we added a type for complex numbers which we call `komplex`. Since there is no standard between C compilers for complex numbers we chose to implement our own type along with routines for the basic math functions of addition, subtraction, multiplication, and division. This maintains portability of BLOPEX.

3.5.2.3 Parameters

A description of the parameters for the LOBPCG solver follows. The functional definition can be found in include `lobpcg.h` available in the online site.

Some parameters are operators. For these a pointer to the operator is passed. The operator must be defined as

```
void (*operatorA)(void*,void*,void*).
```

If a parameter is not needed then a NULL is passed.

Parameter Description

X Required. A block vector. This is the initial guess of eigenvectors. This can be based on prior knowledge or just random guesses. The number of vectors defines the number of eigenvalues to solve for. On output it will contain the computed eigenvectors.

A Optional. Use this if A is a matrix.

operatorA Required. This implements a matrix vector multiplication. If A is NULL then it must define A as an operator and do a matrix vector multiplication.

B Optional. Use if B is a matrix.

operatorB Optional. Only needed if solving a generalized eigenvalue problem. This implements a matrix vector multiplication. If B is NULL then it must define B as an operator and do a matrix vector multiplication.

T Optional. Use this if a preconditioner is supplied. This is data in matrix form to be passed to the preconditioner operatorT. The data is dependent on the preconditioner that operatorT implements. It could be NULL, a preconditioned matrix based on A, or A.

operatorT Optional. But must be supplied if a preconditioner is used. This preforms the actual preconditioning on the residuals block vector.

Y Optional. This is block vector of constraints. Orthogonality of X to Y will be enforced in the LOBPCG solver.

blapfn Required. A structure which contains the addresses to lapack functions (or equivalents) dsygv, dpotrf, zhegv, and zpotrf.

tolerance Required. A structure containing absolute and relative tolerances to apply to the residual norms to test for convergence.

maxIterations Required. The maximum number of iterations to perform.

verbosityLevel Required. The LOBPCG algorithm can print error messages and messages to track progress of the solver. verbosityLevel values control this. Value of 0 means print no messages. Value of 1 means print error messages, max residual norm after each iteration, and eigenvalues after last iteration. Value of 3 means print error messages and eigenvalues after every iteration.

iterations Required. Output. The number of iterations actually performed.

eigs Required. Output. An array containing the eigenvalues computed.

eigsHistory Optional. Output. An array containing eigenvalues produced after each iteration.

eigsHistNum Optional. Input. Max number of eigenvalues. This should be $\geq$ the number of eigenvalues to compute. It is used to reformat the eigsHistory array into a matrix format. Required if eigsHistory is not NULL.

residNorms Optional. Output. An array containing residual norms of eigenvalues computed.

residHistory Optional. Output. An array containing residual norms of eigenvalues produced after each iteration.

residHistNum Optional. Input. Max number of eigenvalues. This should be $\geq$ the number of eigenvalues to compute. It is used to reformat the residHistory array into a matrix format. Required if residHistory is not NULL.

3.5.3 Drivers and Interfaces

Drivers are the programs that setup the eigenvalue problem and BLOPEX abstract is where they are solved. These encompass two separate environments. That of the Driver (PETSc, Hypre, etc.) handle matrix and vector sparse formats and the matrix vector operations on them including application of preconditioners. BLOPEX abstract has all of the logic for the LOBPCG algorithm. The interface is where the functionality of the two environments overlaps.

The interfaces and various multivector structures reviewed in section 3.5.2.2 can be intimidating to a user. To overcome this we supply various Drivers which serve both as examples and in some cases generic problem solvers.

The next sections describe the Drivers and interfaces that are available. For details of execution of tests with these drivers and configuration for PETSc and

Hypre, review the Wiki's available on the Google source html site. See section 3.6 for more information. For execution of BLOPEX under PETSc and Hypre also see the appendices of [41].

3.5.3.1 PETSc

BLOPEX is included as part of the PETSc distribution which must be configured with the option `--download-blopex=1`. Scalar values in PETSc are either real or complex and this must be specified during configuration via the option `--with-scalar-type=complex`. PETSC provides parallel processing support on matrix vector operations.

There are 4 Drivers distributed with PETSc located in the PETSc subdirectory `../src/contrib/blopex`.

- `driver.c` builds and solves a 7pt Laplacian.
- `driver_fiedler.c` accepts as input the matrix A in Petsc format. These can be setup via some Matlab programs in the PETSc socket interface to Matlab; `PetscBinaryRead.m` and `PetscBinaryWrite.m`. These programs read and write Matlab matrices and vectors to files formatted for Petsc. The version from Petsc only supports double. We have modified these programs to also support complex and 64bit integers. Our versions are included in the Google source directory `../blopex_petsc` along with `PetscWriteReadExample.m` to illustrate how to use them.
- `driver_diag.c` solves an eigenvalue problem for a diagonal matrix. This serves as a test program for very large sparse matrices. It has been executed successfully with over 8 million rows.
- `ex2f_blopex.F` is an example of using BLOPEX with PETSc from Fortran.

3.5.3.2 HYPRE

Hypre does not support complex number or 64bit scalars, but like PETSc provides parallel support via matrix vector multiplication and high quality preconditioners. The BLOPEX LOBPCG solver is incorporated into Hypre programs `struct.c` and `ij.c` located in the Hypre directory `../src/test`. These programs have broad functionality and can setup and solve 3D-7pt Laplacians. They can also input matrix files in Hypre formats to construct a generalized eigenvalue problem. These files can be created in Matlab using the Matlab matlab2hype package available on http://www.mathworks.cn/matlabcentral/.

There is also a somewhat less intimidating example in `../src/examples/ex11.c` which solves a 2-D Laplacian eigenvalue problem with zero boundary conditions on an nxn grid.

3.5.3.3 Matlab

The Matlab interface consists of m files and c files available on the Google source site under directory `../blopex_matlab`. The BLOPEX abstract files must also be acquired from the Google source site. All c files are compiled under the Matlab Mex compiler. Complex numbers are supported along with 64-bit integers in the newer version of Matlab. Preconditioners are implemented in this interface as m files.

3.5.3.4 Serial

These are stand alone drivers and interfaces written in C. There are complex and real versions. Matrices created by the drivers are in standard Fortran format. They do not have any parallel support. They have been used primarily for BLOPEX development testing.

3.6 The Google Source Site

BLOPEX source code as of Version 1.1 is maintained on the Google source code site http://code.google.com/p/blopex/ under the SVN version manager. All source is downloadable.

This site also provides some Wiki documents that describe tests we have executed for all interfaces and various systems. Between the source code for the Drivers and the Wiki's we hope users will find BLOPEX accessable and usable.

3.7 Environments BLOPEX Tested On

BLOPEX has been tested in a wide variety of environments. The following is a partial list covered by one or more of the Wiki's described in section 3.6

Machines: UCD XVIB, UCAR Frost, NCAR Bluefire, Lawrence Livermore National Laboratory, IBM PC

Operating Systems: Linux, Fedora, IBM AIX, Cygwin under Windows 7

Compilers: gcc, IBM `blrts_xlc`, g++, pgcc, SUN mpcc, AMD OPEN64

Lapack Libraries: Lapack, AMD ACML, Intel MKL, IBM ESSL

MPI: openmpi, mpich2

3.8 Numerical Results

Some numerical tests for 3D 7-Point Laplacians of the BLOPEX implementation of LOBPCG in Hypre have previously been reported in [44] and in PETSc and Hypre in [41].

We report here on some results using Hypre and a few of the matrices that were analyzed by PRIMME as reported in [54]. These matrices are available from the University of Florida Sparse Matrix Collection at `http://www.cise.ufl.edu/research/sparse/matrices/`. A direct comparison to PRIMME's results is not possible since they are produced on a very different machine.

Matrix	Rows	nnz	nnz(L+U) AMD
Andrews	60,000	760,154	234,019,880
finan512	74,752	596,992	5,600,676
cdf1	70,656	1,825,580	71,684,224
cdf2	123,440	3,085,406	147,417,232

Table 3.1: Matrices Analyzed

All of the matrices used are symmetric positive definite. We use matrix **Andrews**, which has a "seemingly random" sparsity pattern and not much "structure", **finan512**, which is a stochastic matrix used for financial portfolio optimization and **cfd1** and **cfd2**, which are pressure matrices from structural engineering. Their characteristics are described in table 3.1. Note *nnz(L+U) AMD* is the number of nonzeros in L+U of the LU factorization using AMD.

Analysis was performed on a Fedora 10 OS, 4 Quad Core Opteron 2.0 Ghz CPUs, and 64 GB RAM. Hypre was configured using openmpi with gcc compiler and BLAS/LAPACK libraries.

To setup the matrices for processing by Hypre we downloaded the matlab versions and converted them using our matlab2hypreIJ.m program to Hypre formats. This file was then processed using the Hypre ij program. For example to find 5 eigenvalues of **finan512** to a tolerance of $1e-6$ using the BoomerAMG preconditioner we would execute:

./ij -lobpcg -vrand 5 -tol 1e-6 -pcgitr 0 -itr 200 -seed 1 -solver 0 -fromfile finan512

For the first experiment (Table 3.2), we process all of the files in single processor mode. Both the times to setup the preconditioner and to execute the LOBPCG solver are reported. The matrix **Andrews** has an eigenvalue very close to zero, which causes problems orthonormalizing the residual. To overcome this a shift of $1e-7$ was

Matrix	Setup	Eigenvalues to solve for							
		1	2	3	4	5	7	10	15
Andrews	29	4	18	34	62	@	*	*	*
finan512	1	5	10	22	37	43	66	90	203
cdf1	25	152	297	405	599	737	*	*	*
cdf2	36	335	644	1342	*	*	*	*	*

All times rounded to nearest second.

* Analysis not performed.

@ Failure of dsygv routine.

Table 3.2: Single Processor Setup and Solution Time

applied to the matrix. This was successful up to solution for 5 eigenvalues where dsygv routine failed. Also, note the relatively large preconditioner setup times for all matrices except **finan512**. This seems to reflected in the *nnz(L+U) AMD* values shown in Table 3.1. The setup times are independent of the number of eigenvalues to solve for.

The second experiment (Table 3.3) studies the effect of parallel processing on solution time for matrix **finan512**. It solves for 5 eigenvalues using openmpi varying the number of processors. For example to run with 2 processors, we would split finan512 into 2 Hypre files using matlab2hypreIJ.m and process as follows:

mpirun -np 2 ./ij -lobpcg -vrand 5 -tol 1e-6 -pcgitr 0 -itr 200 -seed 2 -solver 0 -fromfile finan5122

3.9 Summary

Version 1.1 of BLOPEX has been implemented in PETSC version 3.1-p3 and submitted to Hypre for inclusion. Version 1.1 incorporates the new features of complex

Processors	Setup Time	Solution Time
1	.42	42.86
2	.96	25.17
3	.63	15.00
4	.40	10.82
5	.31	8.35
6	.24	6.44
7	.20	5.20

Table 3.3: Multiple Processor Setup and Solution Time for finan512

numbers and 64 bit integers. The new Google code site for BLOPEX makes testing documentation available to the user. BLOPEX has interfaces to popular software packages PETSc, Hypre, and Matlab.

APPENDIX A. SpectralCluster Function

This is the primary function used for microarray analysis.

```
 function [partition, con] = spectralcluster(X,numclusters,options,plot,varargin)
% Spectral clustering
% Input: Fixed arguments
%        X              points (vertices) to cluster
%        numclusters   number of clusters to find
%        options        options for computation (as a string)
%                       ----- options for computing weights
%                       gauss  - use gaussian to compute weights
%                       euclidian - use euclidian distance (default)
%                       full   - make fully connected graph
%                       edges  - edges predefined, weights=1
%                       norm   - normalize point vectors
%                       ----- what solver to use
%                       eigs  - use eigs to compute e-val
%                       eig   - use eig (default)
%                       ----- which problem to solve
%                       ncut - solve L*x=lambda*D*x
%                       mass - solve Lx=lambda*M*x
%                       rcut - solve L*x=lambda*x - Rcut (default)
%                       ----- how to compute multiple clusters
%                       kmeans - cluster via kmeans
%                       bicluster - cluster via bicluster (default)

%        plot           plots to produce (as a string)
%                       ------------------------------
%                       nodes    - nodes in colors specified by cdx
%                       edges    - edges in colors specified by cdx
%                       eigval   - eigenvalues
%                       eigvec   - fiedler vector
%                       clusters - graph with clusters in different colors
%                       info     - list misc info
% Input: Variable arguments
```

```
%        Radius         value to use to limit weights
%        Sigma          local density or p-value to use in weights
%        Scale          multiplier for weight computation
%        Mass           vector of node masses
%        Edges          predefined edges
%        Rweak          Rvalue for weak sign graph clustering
%
% Oputput: partition    cell array of indexes to nodes showing partitioning
%          con          vector of connectivity values for clusters
% -------------------------------------
% setup defaults
% -------------------------------------
if nargin < 3
   error('At least 2 parameters are expected');
end
% set problem size and defaults
[xsize,dim]=size(X);
radius = 1;
sigma=ones(xsize,1); %uniform local density
scale = 1;
Mass = ones(xsize,1);
edges = 0;
rweak = 0;

vi=size(varargin,2);
i=1;
while i<vi
   if strfind(varargin{i},'Radius')
      radius=varargin{i+1};
      i=i+2;
   elseif strfind(varargin{i},'Scale')
      scale=varargin{i+1};
      i=i+2;
   elseif strfind(varargin{i},'Sigma')
      sigma=varargin{i+1};
      i=i+2;
   elseif strfind(varargin{i},'Edges')
      edges=varargin{i+1};
```

```
      i=i+2;
   elseif strfind(varargin{i},'Mass')
      Mass=varargin{i+1};
      i=i+2;
   elseif strfind(varargin{i},'Rweak')
      rweak=varargin{i+1};
      i=i+2;
   else
      i=i+1;
   end
end

global OPTIONS NUMCLUSTERS DW PLOT RWEAK;
OPTIONS = options;
PLOT = plot;
NUMCLUSTERS = numclusters;
RWEAK = rweak;

color=[0 0 1;  %blue
   1 0 0;  %red
   0 1 0;  %green
   0 0 0;  %black
   0 1 1;  %cyan
   1 0 1;  %magenta
   1 1 0]; %yellow
color=[color; color; color; color];
% -------------------------------------
% define similarity (weight) matrix  S(i,j)
% since diag is zero and symmetric we only need upper part
% -------------------------------------
if strfind(OPTIONS,'norm')
   % normalize X vectors
   tic
   for i=1:xsize
     X(i,:)=X(i,:)/norm(X(i,:));
   end
   sprintf('normalize %f',toc)
end
```

```
if strfind(OPTIONS,'gauss')
   % gaussian (adj by sigma)
   tic
   S=zeros(xsize,xsize);
   for i=1:xsize
      for j=i+1:xsize
         % calculate a local density sigma
         sc=min(sigma(i),sigma(j))^2/(sigma(i)*sigma(j));
         sc=sc*scale;
         % compute inverse gausian distance
         S(i,j)=exp(-sc*norm(X(i,:)-X(j,:))^2);
         % exclude wts that are too small
         if S(i,j) < radius
            S(i,j)=0;
         end
      end
   end
   sprintf('gauss mean %f',mean(mean(S)))
   sprintf('gauss time %f',toc)
elseif strfind(OPTIONS,'full')
   % fully connected graph wt 1 (adj by sigma)
   npt=size(X,1);
%    [xdum,edges]=completegraph(npt,0,0,1);
%    clear xdum
   S=zeros(npt);
   for i=1:npt-1
      for j=i+1:npt
         S(i,j)=scale*min(sigma(i),sigma(j))^2/(sigma(i)*sigma(j));
      end
   end
elseif strfind(OPTIONS,'edges')
   % edges supplied weight 1 for all edges
   if edges == 0
      error('Variable argument Edges is missing');
   end
   weights=ones(size(edges,1),1);
   S=adjacency(edges,weights);
```

```
   S=full(S);
else
   % wt 1 if inside radius (adj by sigma)
   S=spalloc(xsize,xsize,200*xsize);
   for i=1:xsize
      for j=i+1:xsize
         a=norm(X(i,:)-X(j,:));
         if a < radius
            S(i,j)=scale*min(sigma(i),sigma(j))^2/(sigma(i)*sigma(j));
         end
      end
   end

end

if strfind(PLOT,'info')
   sprintf('vertices %f',xsize)
   sprintf('pct nnz %f', (nnz(S)*100)/(xsize*xsize))
end

% ----------------------------
% compute dummy node weight
% this to be used later in solve function
% ----------------------------
% edge wt is min of .1 of smallest non zero wt in S or .001
DW=full(min(S(S>0)))*.1;
DW=min(DW,.001);
DW=DW/xsize;
% -------------------------------------
% define Graph (i.e. adjacency matrix)
% -------------------------------------
W=S+S';
clear S;
% plot graph without edges
if strfind(PLOT,'nodes')
   cdx=ones(xsize,1);
   figure
   scatter(X(:,1),X(:,2),30,color(cdx,:),'filled')
```

```
   title('Graph nodes')
end

% plot graph with edges
if strfind(PLOT,'edges')
   cdx=ones(xsize,1);
   figure
   i=xsize-1;
   gplot2(W,X,3,cdx);
   title(['Graph nodes & edges:' OPTIONS])
end

% --------------------------------
% do kmeans clustering on column vector
% defined by 1st k+1 eigenvectors
% --------------------------------
if strfind(OPTIONS,'kmeans')
   % solve the e-val problem
   k1=NUMCLUSTERS+10;
   [V,e,eidx]=solve(W,Mass,k1);
   % define row vectors of 1st numcluster rows of eigenvectors V
   % excluding first 0 eigenvalue
   k=numclusters;
   yidx=eidx(2:k+1);
   pidx=kmeans(V(:,yidx),k,'emptyaction','singleton','MaxIter',200);
   for i=1:k
      partition{i}=find(pidx==i);
   end
   % plot eigenvalues
   if strfind(PLOT,'eig')
      figure
      stairs(1:k1,e(1:k1));
      title(['Eigenvalues:' OPTIONS]);
      %   ploteig(W,V(:,idx),16);
   end
end

% ------------------------------------
```

```
% find clusters via successive bicluster
% -------------------------------------
if strfind(OPTIONS,'kmeans')
else
   % put all unconnected vertices in partition 1
   % and restrict analysis to remaining vertices
   sumW=sum(W);
   if sum(sumW==0) ~= 0
     partition{1}=find(sumW==0);
     partition{2}=find(sumW~=0);
     pi=2;
     psize=2;
     NUMCLUSTERS = NUMCLUSTERS + 1;
   else
     partition{1}=1:xsize;
     pi=1;
     psize=1;
   end
   pnz=pi;
   while psize < NUMCLUSTERS
      % bicluster least connected cluster
      % pass cluster pi to bicluster
      idx=partition{pi};
      [part1,part2]=bicluster(W(idx,idx),Mass(idx));
      % incorporate part1 and part2 into partition
      % part1 and part2 are indices wrt idx
      % so must convert to indices of original vertices
      partition{pi}=idx(part1);
      psize = psize + 1;
      partition{psize}=idx(part2);
      % find least connected cluster
      con=zeros(1,psize);
      % if partition 1 if for unconnected vertices exclude these
      if pnz~=1
         con(1)=999;
      end
      for i=pnz:psize
         ps=size(partition{i},2);
```

```
         if ps==0
            error('0 partition size in cluster')
         end
         ps=ps*(ps-1);
         if ps==0  %eliminate single nodes from consideration
            con(i)=999;
         else %compute connectivity relative to fully connected graph
            con(i)= sum(sum(W(partition{i},partition{i})));
            con(i)= con(i)/ps;
         end
      end
      % find partition least strongly connected
      m=min(con);
      pi=find(con==m);
      % if equal connectivity then take 1st one
      if size(pi,2)>1
         pi=pi(1);
      end
      if strfind(PLOT,'debug')
        con
        partition
        pi
      end
   end
end

if strfind(PLOT,'info')
   fprintf(1,'%s \n','Partition size and connectivity')
   for i=1:size(partition,2);
      fprintf(1,'%5d %8f \n',size(partition{i},2),con(i))
   end
end

% ---------------------------
% plot results of clustering
% ---------------------------
if strfind(PLOT,'clusters')
   figure
```

```
   if dim == 2 % vertices lie in a plane
      numpart=size(partition,2);
      for i=1:numpart
         ix=partition{i};
         if i < 8
            scatter(X(ix,1),X(ix,2),30,color(i,:),'o','filled')
         elseif i < 15
            scatter(X(ix,1),X(ix,2),30,color(i,:),'*')
         else
            scatter(X(ix,1),X(ix,2),20,color(i,:),'s','filled')
         end
         if i==1
            hold on
         end
      end
      title(['Clusters:' OPTIONS]);
      hold off
      %       figure
      %       gplot(W,X,'-*');
   else % vertices in more than 2 dimensions
% problem with indexes in plot stmt
% don't get this after return to calling pgm
%       for c = 1:NUMCLUSTERS
%          subplot(4,4,c);
%          Y=X(partition{c},:);
%          plot(Y)
%          axis tight
%          title(['Clusters:' OPTIONS]);
%       end
   end
end

   function [part1,part2]=bicluster(W,Mass)

      global OPTIONS
      % save the initial size
      origsize = size(W,1);
      % solve the e-val problem
```

```
   k1=4;
   [V,e,eidx]=solve(W,Mass,k1);
   % do a single bicluster
   for i=1:k1
      if e(i) > 1e-10
         part1 = find(V(:,eidx(i))>=-RWEAK);
         part2 = find(V(:,eidx(i))<=RWEAK);
         % if original partition is same as part1 and part2 then error
         if origsize==size(part1,1) & origsize==size(part2,1)
            error('Bicluster error. Same size partitions. r value possibly too large.
         end
         % dummy node may introduce empty partition when graph
         % is fully connected to begin with and after node is
         % removed from evec, in this case keep looking.
         % 1st evec where this does not occur is real fiedler vec
         if size(part1,1)>0 & size(part2,1)>0
           break
         end
      end
   end
   if strfind(PLOT,'eigvec')
      V(:,eidx(i))
   end
   % plot eigenvalues
   if strfind(PLOT,'eigval')
      figure
      stairs(1:k1,e(1:k1));
      title(['Eigenvalues:' OPTIONS]);
      %   ploteig(W,V(:,idx),16);
   end
end

function [V,e,eidx]=solve(W,Mass,k1)

   global OPTIONS DW

   % add dummy node to elim multi components
   if strfind(OPTIONS,'full')
```

```
   xs=size(W,1);
else
   xs=size(W,1)+1;
   for i=1:xs-1
      W(i,xs)=DW;
      W(xs,i)=DW;
   end
   W(xs,xs)=0;
end

SD = sum(W,2);
D = sparse(1:xs,1:xs,SD);
L = D - W;

% solve L for 1st (low to high) k1 e-value, e-vectors
% note: Matlab returns e-values in a diag matrix
k1=min(k1,xs);
if strfind(OPTIONS,'eigs')
   opts.issym=1;
   opts.disp=0;
   warning off MATLAB:nearlySingularMatrix
   if strfind(OPTIONS,'ncut')
      [V,E,flag]=eigs(L,D,k1,'sm',opts);
      if flag ~= 0; flag; end
   elseif strfind(OPTIONS,'mass')
      % add a dummy mass
      if strfind(OPTIONS,'full')
      else
         Mass=[Mass; DW]; %dummy mass
      end
      M=sparse(diag(Mass));
      [V,E,flag]=eigs(L,M,k1,'sm',opts);
      if flag ~= 0; flag; end
   else
      % warning: setting the sigma in eigs too low can result in
      % 'matrix is singular to working precision' and e-val of NaN
      % .001 is too low
      [V,E,flag]=eigs(L,k1,.01,opts);
```

```
            if flag ~= 0; flag; end
         end
         warning on MATLAB:nearlySingularMatrix
      else
         L=full(L);
         D=full(D);
         if strfind(OPTIONS,'ncut')
            [V,E]=eig(L,D);
         elseif strfind(OPTIONS,'mass')
            if strfind(OPTIONS,'full')
            else
               Mass=[Mass; DW]; %dummy mass
            end
            M=full(diag(Mass));
            [V,E]=eig(L,M);
         else
            [V,E]=eig(L);
         end
      end
      % force eigenvalues into low to high order
      E=sum(E);
      [e,eidx]=sort(E);
      e(1:k1)
      %don't include the dummy node
      if strfind(OPTIONS,'full')
      else
         V=V(1:xs-1,:);
      end
   end

end
```

APPENDIX B. SpectralClusterTest Driver

This is a Matlab program used to produce some of the examples presented in the paper. It demonstrates how to call the spectralculster function to perform recursive biclustering.

```
 % spectralcluster tests
test=3;
switch test
   case 1
% four clusters normally distributed around
% found points in R^2
randn('state',5)
X=sample([2,3],1,50);
Y=sample([8,9],2,80);
X=[X; Y];
Y=sample([2,9],1,30);
X=[X; Y];
Y=sample([5,6],.5,40);
X=[X;Y];
n=size(X,1);
 cdx = spectralcluster(X,5,'eig,gauss','edges,clusters,info,debug','Radius',.02,'Rweak',
   case 2
% yeastvalues from Matlab demo "Gene Expression Profile Analysis
% these are after filtering to eliminate genes with low expression
% this is a set of 7 microarrays, taken in a time sequence for
% metabolic shift from fermentation to respiration
load c:\cnsdemo\yeastvalues.mat
X=yeastvalues;
[cdx,con] = spectralcluster(X,16,'eigs,norm','info','Radius',.2,'Rweak',.0000001);
% [cdx,con] = spectralcluster(X,16,'eigs','info','Radius',2,'Rweak',.0000001);
[xsize,dim]=size(X);
% we don't plot the first partition since this is the collection of
% isolated vertices and carries no obvious information
figure('Color','white')
```

```
for c = 2:17
   subplot(4,4,c-1);
   plot(1:dim,X(cdx{c},:))
   t=sprintf('%d %f',size(cdx{c},2),con(c));
   title(t)
   axis tight
end
   case 3
% 5pt complete + 3pt complete+ 4pt complete
[p1,e1]=complete_graph(5,0,0,2);
[p2,e2]=complete_graph(3,5,0,2);
[p3,e3]=complete_graph(4,0,6,2);
points=[p1; p2; p3;2.5 3];
e2=e2+5;
e3=e3+5+3;
edges=[e1;e2;e3;1 13;7 13;9 13];
cdx = spectralcluster(points,3,'edges,eig,ncut','clusters,edges','Edges',edges);
   otherwise
end
```

APPENDIX C. Subroutines Used by SpectralCluster Funtion

These are the subroutines call by the Matlab spectralcluster function.

```
 function gplot2(W,points,range,Idx,area)
% plot the partition of a graph with different edge weights
%
% inputs W      weighted adj matrix
%        points x,y coord of graph nodes
%        range  max graphed size of an edge
%        Idx    partition of the graph, values 1,2,3,...
%        area   size of nodes

n=size(points,1);
if nargin < 5
   area = 30;
end
if nargin < 4
   Idx=ones(1,n);
end
if nargin < 3
   range = 3;
end

hold on
% find range of vertex coord and adj axes
xmin=min(points(:,1));
xmax=max(points(:,1));
ymin=min(points(:,2));
ymax=max(points(:,2));
axis([xmin-1 xmax+1 ymin-1 ymax+1] );

% plot nodes
color=[0 0 1;  %blue
       1 0 0;  %red
       0 1 0;  %green
```

```
       0 0 0;  %black
       0 1 1;  %cyan
       1 0 1;  %magenta
       1 1 0]; %yellow
color=[color; color; color; color];

scatter(points(:,1),points(:,2),area,color(Idx,:),'filled')
% for i=1:n
%    plot(points(i,1),points(i,2),' *b')
% end

% find edges
[x,y]=find(triu(W));
edges=[x,y];

n=size(edges,1);
% plot edges
idx=triu(W)>0;
xmin=min(W(idx));
xmax=max(W(idx));

for i=1:n
   X=[ points(edges(i,1),1) points(edges(i,2),1) ];
   Y=[ points(edges(i,1),2) points(edges(i,2),2) ];
   % compute line width in points
   if xmax==xmin
      width = .5;
   else
      width=(W(edges(i,1),edges(i,2))-xmin)/(xmax-xmin);
      width=width*range+.3;
   end
   line(X,Y,'LineStyle','-','Color','b','LineWidth',width);
end
hold off



 function W=adjacency(edges,weights)
```

```
% Produce adjacency matrix of graph
% defined by input parameters edges and weights
% Graph nodes are numbered from 1 to N.
% The highest order node should have an edge.
% Input paramater edges has an entry for each graph edge
% edges(1,1) is node with connection to edges(1,2).
% weights(1) is weight to assign to edge 1.

% Get number of graph nodes
N=max(max(edges));

%Build sparse adjacency matrix
%Note that matrix is symmetric
r=[edges(:,1);edges(:,2)];
c=[edges(:,2);edges(:,1)];
v=[weights weights];

% Build NxN sparse matrix
% W(r(i),c(i))=v(i)
W=sparse(r,c,v,N,N);
```

APPENDIX D. List of Genes by Cluster

Gene NCBI locus tags corresponding to the Clusters extracted from the Matlab demo "Gene Expression Profile Analysis".

```
---- Cluster Sequence 2, Number of Genes 2,  Connectivity 1.000000
YGL059W  YOR177C


---- Cluster Sequence 3, Number of Genes 3,  Connectivity 1.000000
YCR036W  YMR104C  YOR032C


---- Cluster Sequence 4, Number of Genes 12,  Connectivity 0.121212
YBR050C  YJL164C  YJR008W  YKL091C  YPL256C
YBR051W  YPR002W  YBR056W  YDL234C  YFR055W
YHL039W  YGR052W


---- Cluster Sequence 5, Number of Genes 31,  Connectivity 0.150538
YAL034C  YBL043W  YBL049W  YBR046C  YBR285W
YCR091W  YDL204W  YDL218W  YDR330W  YKL093W
YLR164W  YBL048W  YDR043C  YDR313C  YGR236C
YIL097W  YIL101C  YJL067W  YJR155W  YKL016C
YNR007C  YOR097C  YPL185W  YGR243W  YNL093W
YEL039C  YGR146C  YIL113W  YKL217W  YMR107W
YPR150W


---- Cluster Sequence 6, Number of Genes 85,  Connectivity 0.206162
YAL003W  YAL012W  YBR048W  YCL054W  YDL148C
YDR144C  YDR384C  YEL026W  YGR155W  YGR159C
YJR063W  YLR196W  YLL047W  YMR131C  YNL111C
YNL175C  YNL207W  YNL303W  YNR050C  YNR054C
YPL012W  YPR137W  YCL053C  YCLX02C  YDL083C
```

YDR025W YGR092W YGR160W YHR128W YMR049C
YMR229C YMR290C YNL060C YNL110C YNL132W
YNL182C YNL256W YOR361C YPL043W YPL093W
YPR144C YAL036C YBR247C YDL182W YDR206W
YDR398W YEL040W YER036C YGL076C YGL078C
YGR103W YIL053W YJL122W YJL148W YJR041C
YJR071W YKL009W YKL081W YLR186W YLR056W
YMR037C YMR217W YMR239C YNL075W YNL141W
YNL313C YOR116C YPL126W YPL226W YAL025C
YDL063C YDL213C YGL029W YKL078W YKL082C
YLR009W YLR129W YLL008W YLR355C YLR449W
YMR093W YNL002C YNL120C YNR067C YOL010W

---- Cluster Sequence 7, Number of Genes 6, Connectivity 0.200000
YCR019W YDR436W YJR006W YNR034W YBR069C
YDR101C

---- Cluster Sequence 8, Number of Genes 22, Connectivity 0.129870
YAL054C YER024W YGR067C YLR142W YKR097W
YDR505C YER065C YJL089W YCR005C YFL030W
YIL057C YMR118C YNL117W YNL195C YPL054W
YCR010C YDL215C YDR009W YKL171W YLR377C
YPR030W YDL215C

---- Cluster Sequence 9, Number of Genes 35, Connectivity 0.159664
YBR116C YDR096W YDR216W YML042W YNL009W
YNR002C YPL134C YPL262W YBR117C YDL199C
YDL245C YEL012W YER096W YER098W YGL153W
YGR110W YHL032C YHR096C YJL045W YKL107W
YKL187C YLR267W YOR027W YPL109C YPL135W
YER015W YML054C YOL084W YBR298C YDL233W
YDR262W YGR224W YJR095W YOR019W YPL201C

---- Cluster Sequence 10, Number of Genes 23, Connectivity 0.189723
YBR241C YDR148C YDR306C YGR043C YGR201C
YGR231C YJL144W YML131W YOR120W YBR280C
YMR068W YBR203W YDR030C YDR494W YJL170C
YLR254C YLR080W YMR030W YBL086C YDL169C
YDR275W YNR071C YNR073C

---- Cluster Sequence 11, Number of Genes 8, Connectivity 0.250000
YNL174W YPL183C YMR108W YBR155W YJL109C
YKL191W YNL216W YPR136C

---- Cluster Sequence 12, Number of Genes 269, Connectivity 0.118127
YBL015W YBR052C YBR072W YBR169C YBR183W
YDL004W YDL124W YDR070C YDR074W YDR258C
YDR272W YDR358W YEL011W YEL024W YER141W
YFL014W YGR019W YGR111W YIL124W YIL162W
YJL166W YJR104C YKL065C YKL067W YKL085W
YLR168C YLR216C YLL041C YLL023C YLR270W
YLR290C YLR356W YML100W YML120C YMR173W
YMR181C YNL015W YNL037C YNL173C YOL126C
YOL053C YOR052C YOR220W YOR244W YPL186C
YPL230W NORF 7 YAL060W YAR028W YBL050W
YBL064C YBR139W YBR147W YBR214W YBR256C
YCL035C YCR021C YDL181W YDR001C YDR077W
YDR125C YDR171W YDR272W YDR453C YDR529C
YDR533C YER067W YGL037C YGL187C YGL259W
YGR044C YGR088W YGR130C YGR132C YGR182C
YGR250C YHR051W YHR104W YHR195W YIL169C
YIR038C YJL137C YJL161W YJL185C YJR034W
YJR080C YKL036C YKL141W YLR193C YLR217W
YLR219W YLR093C YLR149C YKR058W YKR076W
YLL026W YLR271W YLR295C YML128C YMR105C
YMR311C YNL134C YNL160W YNL200C YNL252C
YNL274C YOL117W YOR031W YOL032W YOL048C
YOL071W YOR049C YOR215C YOR273C YOR289W

YOR317W YPL087W YPL165C YPR020W YPR026W
YPR098C NORF 4 NORF 8 NORF 54 YBL075C
YBL099W YBL107C YBR054W YBR126C YBR269C
YDL022W YDR032C YDR178W YDR342C YER053C
YFL054C YFR033C YGL006W YGL198W YGR149W
YHL021C YHR087W YJL102W YJR019C YJR073C
YJR096W YKL148C YKL150W YLR178C YLR252W
YLR258W YLR038C YKR067W YLR304C YMR090W
YMR110C YMR133W YMR145C YMR196W YMR271C
YNL045W YNL115C YNL305C YOR136W YOR178C
YOR374W YPL004C YPL078C YPL154C YPL196W
YPR149W YDR258C YAL017W YBL030C YBL038W
YBL078C YBL100C YBL108W YBR101C YBR149W
YBR222C YBR230C YCL025C YCR097W YDL021W
YDL023C YDL067C YDL091C YDR031W YDR059C
YDR085C YDR231C YDR277C YDR329C YDR343C
YDR377W YDR513W YER035W YER079W YER150W
YER158C YER182W YFR015C YGL045W YGL047W
YGL121C YGL191W YGL199C YGR008C YGR028W
YGR070W YGR142W YGR174C YGR194C YGR238C
YGR244C YGR248W YHL024W YHR016C YHR092C
YHR209W YIL087C YIL107C YJL079C YJL103C
YJL151C YJL155C YJR048W YJR121W YKL026C
YKL151C YKL193C YKR016W YKR046C YLR251W
YLR081W YLR299W YLR327C YLR345W YLR395C
YLR423C YML004C YMR031C YMR056C YMR081C
YMR136W YMR170C YMR188C YMR195W YMR197C
YMR250W YMR297W YNL052W YNL100W YNL144C
YNL194C YNR001C YOL153C YOR035C YOR041C
YOL083W YOR065W YOR089C YOR161C YOR347C
YPL123C YPL223C YPR184W NORF 46

---- Cluster Sequence 13, Number of Genes 143, Connectivity 0.140451
YAR073W YBR092C YBR187W YBR189W YBR191W
YDL082W YDL130W YDR382W YDR450W YEL054C
YER070W YER074W YER117W YGR085C YHL001W
YHL033C YHR141C YHR216W YIL069C YJL136C

YJL190C YLR198C YLR212C YLL045C YLR044C
YLR048W YLR076C YKR057W YKR059W YLR264W
YLR340W YLR384C YLR432W YMR121C YNL013C
YNL247W YNL301C YNL327W YOL120C YOR224C
YOR310C YOR312C YPL142C YPL160W YPR145W
YBL024W YDR165W YDR341C YDR365C YER002W
YGL135W YGR034W YHR215W YIL018W YIL052C
YJL189W YKL181W YLR175W YLL044W YLR029C
YLR075W YLR339C YLR367W YLR409C YLR413W
YML123C YNL178W YNL302C YNR053C YOL121C
YOL127W YOL077C YOR153W YOR335C YOR369C
YAL038W YBL027W YBR032W YBR106W YBR181C
YBR249C YDL136W YDL208W YDR012W YDR060W
YDR064W YDR418W YER131W YGL030W YGL102C
YHL015W YHR089C YHR208W YLR180W YLR060W
YLR062C YLR344W YLR372W YLR448W YML063W
YMR318C YNL065W YNL069C YNL119W YOR234C
YPL198W YPL220W NORF 17 YBL076C YDL167C
YDR037W YDR321W YDR417C YDR447C YDR449C
YDR471W YER110C YFR031BC YGL031C YGL103W
YGL123W YGR148C YGR214W YGR264C YHR203C
YIL133C YJL177W YJR123W YJR145C YKL006W
YLR249W YLR325C YLR441C YMR242C YNL096C
YOL040C YOR063W YOR309C YPL081W YPL131W
YPR102C YPR132W NORF 20

---- Cluster Sequence 14, Number of Genes 10, Connectivity 0.466667
YBL045C YBR067C YIL125W YJL163C YKL109W
YLR173W YOL053W YGL192W YMR191W YBR039W

---- Cluster Sequence 15, Number of Genes 9, Connectivity 0.138889
YCLX09W YPR043W YBR218C YLR341W YPR116W
YNL067W YDR039C YDR491C YGR094W

```
---- Cluster Sequence 16, Number of Genes 13,  Connectivity 0.269231
YAL026C  YAR027W  YKL035W  YER044C  YDR516C
YFR053C  YKL142W  YMR278W  YOL082W  YIL111W
YKL103C  YLR257W  YOR285W


---- Cluster Sequence 17, Number of Genes 4,  Connectivity 0.333333
YGL158W  YCR039C  YMR232W  YLR297W

end
```

REFERENCES

[1] Charles J. Alpert, Andrew B. Kahng, and So-Zen Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90:3–26, 1999.

[2] Charles J. Alpert and So-Zen Yao. Spectral partitioning: The more eigenvectors, the better. In *Proc. ACM/IEEE Design Automation Conf*, pages 195–200, 1994.

[3] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Transactions on Mathematical Software*, 36(3):13:1–13:??, July 2009.

[4] Turker Biyikoglu, Josef Leydold, and Peter F. Stadler. *Laplacian Eigenvectors of Graphs*. Springer-Verlag, Berlin Heidelberg, 2007.

[5] Benjamin Milo Bolstad. *Low-level Analysis of High-density Oligonucleotide Array Data*. PhD thesis, University of Waikato, 2004.

[6] Edited by Charles-Edmond BIchot and Patrick Siarry. *Graph Partitioning*. Wiley, New York, 2011.

[7] Tony F. Chan, Tony Chan Ciarlet, and W. K. Szeto. On the optimality of the median cut spectral bisection graph partitioning method. *SIAM Journal on Scientific Computing*, 18:943–948, 1997.

[8] Duhong Chen, J. Gordon Burleigh, and David Fernandez-Baca. Spectral partitioning of phylogenetic data sets based on compatibility. *Syst. Biol.*, 56(4):623–632, 2007.

[9] S.Y. Cheng. Eigenfunctions and nodal sets. *Comment. Math. Helvetici*, 51:43–55, 1976.

[10] Yun Chi, Xiaodan Song, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 153–162, New York, NY, USA, 2007. ACM.

[11] Fan R. K. Chung. *Spectral Graph Theory*, chapter 2.2. A.M.A. CBMS, Providence, Rhode Island, 1997.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Edition*. MIT Press, Cambridge, Massachusetts, 2001.

[13] R. Courant and D. Hilbert. *Methods of Mathematical Physics, Vol. 1*. Interscience, New York, 1953.

[14] E. Brian Davies, Graham M.L. Gladwell, Josef Leydold, and Peter F. Stadler. Discrete nodal domain theorems. *Linear Algebra and its Applications*, 336:51–60, 2001.

[15] Harry F. Davis. *Fourier Series and Orthogonal Functions*, chapter 4.2. Dover Publications, Inc., New York, 1963.

[16] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. A unified view of kernal k-means, spectral clustering and graph cuts. *UTCS Technical Report TR-04-25*, 2005.

[17] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell*, 29, 2007.

[18] Chris Ding, Xiaofeng He, and Horst D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc. SIAM Data Mining Conf*, pages 606–610, 2005.

[19] Chris Ding, Xiaofeng He, and Hongyuan Zha. A specral method to separate disconnected and nearly-disconnected web graph components. *Proc 7th Int'l Conf. on Knowledge Discovery and Data Mining*, KDD 2001:275–280, 2001.

[20] W. Donath and A. Hoffman. Algorithms for partitioning graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15 no.3:938–944, 1972.

[21] W. Donath and A. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, pages 420–425, 1973.

[22] Art M. Duval and Victor Reiner. Perron-frobenius type results and discrete versions of nodal domain theorems. *Linear Algebra and its Applications*, 294:259–268, 1999.

[23] Stanley J. Farlow. *Partial Differential Equations for Scientists and Engineers*. Dover Publications, Inc., New York, 1982.

[24] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59 no. 2:167–181, 2004.

[25] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23 no.2:298–305, 1973.

[26] Miroslav Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czech. Math. J.*, 25, no. 100:619–633, 1975.

[27] Miroslav Fiedler. *Special Matrices and Their Applications in Numerical Mathematics*. Dover edition, Boston, 2008.

[28] Igor Fischer and Jan Poland. New methods for spectral clustering. Dalle Molle Institute for Artificial Intelligence, 2004.

[29] J. Friedman. Some geometric aspects of graphs and their eigenfunctions. *Duke Math J.*, 69(3):487–525, 1993.

[30] G.M.L. Gladwell and H. Zhu. Courant's nodal line theorem and its discrete counterparts. *Q. Jl Mech. Appl. Math.*, 55:1–15, 2002.

[31] Leo Grady. Graph analysis toolbox matlab code. http://eslab.bu.edu/software/graphanalysis/, August 2003.

[32] Leo Grady and Eric L. Schwartz. Isoperimetric graph partitioning for image segmentation. *IEEE Trans. on Pat. Anal. and Mach. Int*, 28:469–475, 2006.

[33] D.H. Griffel. *Applied Functional Analysis*. Halsted Press, New York, 1981.

[34] Ji-Ming Guo. The algebraic connectivity of graphs under perturbation. *Linear Algebra and its Applications*, 433(6):1148 – 1153, 2010.

[35] U. Hetmaniuk and R. Lehoucq. Basis selection in lobpcg. *J. Comput. Phys.*, 218:324–332, 2006.

[36] Desmond J. Higham, Gabriela Kalna, and Milla Kibble. Spectral clustering and its use in bioinformatics. *Journal of Computational and Applied Mathematics*, 204:25–37, 2007.

[37] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, NY, 2005.

[38] D. Jerison and C. Kenig. Unique continuation and absence of positive eigenvalues for schrodinger operators. *Ann. Math.*, 121:159–268, 1999.

[39] Claes Johnson. *Numerical Solutions of Partial Differential Equations by the Finite Element Method*, chapter 24.

[40] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM*, 51 no. 3:497–515, 2004.

[41] A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov. Block locally optimal preconditioned eigenvalue xolvers (blopex) in hypre and petsc. *SIAM J. Sci. Comput*, 29:2224–2239, 2007.

[42] Andrew V. Knyazev. Preconditioned eigensolvers – an oxymoron? *Electron. Trans. Numer. Anal.*, 7:104–123, 1998.

[43] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput*, 23:517–541, 2001.

[44] Andrew V. Knyazev and Merico E. Argentati. Implementation of a preconditioned eigensolver using hypre. Technical Report UCD-CCM 220, Center for Computational Mathematics, University of Colorado Denver, 2005.

[45] Anna Matsekh, Alexei Skurikhin, Lakshman Prasad, and Edward Rosten. Numerical aspects of spectral segmentation. In *Applied Parallel and Scientific Computing*, volume LNCS 7133, pages 193–203, 2012.

[46] Marina Meila and Jianbo Shi. Learning segmentation by random walks. In *In Advances in Neural Information Processing*, pages 470–477. MIT Press, 2000.

[47] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *AI and STATISTICS (AISTATS) 2001*, 2001.

[48] Boaz Nadler, Stphane Lafon, Ronald R. Coifman, and Ioannis G. Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of fokker-planck operators. In *in Advances in Neural Information Processing Systems 18*, pages 955–962. MIT Press, 2005.

[49] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[50] Pekka Orponen and Satu Elisa Schaeffer. Local clustering of large graphs by approximate fiedler vectors. In *Proceedings of the Fourth International Workshop on Efficient and Experimental Algorithms (WEA05), volume 3505 of Lecture Notes in Computer Science*, pages 524–533. Springer-Verlag GmbH, 2005.

[51] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice–Hall, Inc., Englewood Cliffs, N.J., 1980.

[52] Eitan Sharon, Meirav Galun, Dahlia Sharon, Ronen Basri, and Achi Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442(17):810–813, 2006.

[53] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[54] Andreas Stathopoulos and James R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Transactions on Mathematical Software*, 37(2):21:1–21:30, April 2010.

[55] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, July 1997.

[56] Ulrike von Luxburg. A tutorial on spectral clustering. Technical Report No. Tr-149, Max Planck Institute for Biological Cybernetics, August 2006.

[57] Y. Weiss. Segmentation using eigenvectors: A unifying view. *Internation Conference on Computer Vision*, pages 974–982, September 1999.

[58] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In *Siam Conference on Data Mining*, 2005.

[59] Lihi Zelnik-manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, 2004.

[60] Shu-BO Zhang, Song-Yu Zhou, Jian-GuO He, and Jian-Huang Lai. Phylogeny inference based on spectral graph clustering. *Journal of Computational Biology*, 18 no. 4:627–637, 2011.