Spectral Graph Theory Lecture 1

Introduction

Daniel A. Spielman September 2, 2009

1.1 A quick introduction

First of all, please call me "Dan". If such informality makes you uncomfortable, you can try "Professor Dan". Failing that, I will also answer to "Prof. Spielman".

As the title suggests, this course is about the eigenvalues and eigenvectors of matrices associated with graphs, and their applications. I will never forget my amazement at learning that combinatorial properties of graphs could be revealed by an examination of the eigenvalues and eigenvectors of their associated matrices. I hope to both convey my amazement to you, but then to make it feel like common sense. I'm now shocked when any important property of a graph is not revealed by its eigenvalues and eigenvectors.

This class will fundamentally be a math class, but my emphasis is on material that I find useful. I'll present a lot of theorems, a few algorithms, and a bunch of open problems.

In this lecture, I hope to survey everything that we will cover during the semester. These notes should contain a few details that I will skip during lecture. Please note that these notes have not been written very carefully, so every formal statement may be slightly wrong.

1.2 Graphs

First, we recall that a graph $G = (V, E)$ is specified by its vertex set, V, and edge set E. In an undirected graph, the edge set is a set of unordered pairs of vertices. Unless otherwise specified, all graphs will be undirected, and finite.

Graphs are typically used to model connections or relations between things, where "things" are vertices. Common "natural" examples are:

- Friendship graphs: people are vertices, edges exist between pairs of people who are friends (assuming the relation is symmetric).
- Network graphs: devices, routers and computers are vertices, edges exist between pairs that are connected.
- Circuit graphs: electronic components, such as transistors, are vertices: edges exist between pairs connected by wires.

Of course, we also study many abstract, mathematically defined graphs. For example,

- The path on n vertices. The vertices are $\{1, \ldots n\}$. The edges are $(i, i+1)$ for $1 \leq i < n$.
- The ring on n vertices. The vertices are $\{1, \ldots n\}$. The edges are all those in the path, plus the edge $(1, n)$.
- The hypercube on 2^k vertices. The vertices are elements of $\{0,1\}^k$. Edges exist between vertices that differ in only one coordinate.

While we think of the hypercube as an abstract graph, it will also show up in applications. When studying probabilities involving k independent random variables taking values in $\{0,1\}$, we will find it convenient to identify them with the vertices of the hypercube. Similarly, when studying random sets of size s of $\{1, \ldots m\}$, we will create one vertex for each set, and put an edge between two sets that differ in exactly two elements.

1.3 Matrices for Graphs

Typically, and without loss of generality, we will assume that $V = \{1, \ldots, n\}$. The most natural matrix to associate with a graph G is its adjacency matrix, A_G, whose entries $a_{i,j}$ are given by

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

While the adjacency matrix is the most natural matrix to associate with a graph, I also find it the least useful. Eigenvalues and eigenvectors are most meaningful when used to understand a natural operator or a natural quadratic form. The adjacency matrix provides neither.

The most natural operator associated with a graph G is probably its diffusion operator. To construct this, let D_G be the diagonal matrix in which $D_G(i,i)$ is the degree of vertex i. We then set

$$W_G = D_G^{-1} A_G.$$

This matrix describes the behavior of random walks in the graph G. If $p \in \mathbb{R}^V$ is a row-vector giving a probability that some particle is at each vertex ($p(i)$ is the probability of vertex i), and the particle moves to a random neighbor of its present location, then pW_G is the new probability distribution of the particle. Of course, when the graph is *regular*, that is when every vertex has the same degree, W_G is merely a rescaling of A_G. I think this is why researchers got away with studying the adjacency matrix for so long.

The most natural quadratic form associated with a graph is defined in terms of its Laplacian matrix,

$$L_G \stackrel{\text{def}}{=} D_G - A_G.$$

Given a function on the vertices, $x \in \mathbb{R}^V$, the Laplacian quadratic form is

$$x^T L_G x = \sum_{(i,j) \in E} (x(i) - x(j))^2. \tag{1.1}$$

This form measures the smoothness of the function x. It will be small if the function x does not jump too much over any edge.

1.4 Eigenvectors

Let's get our feet wet by examining the Laplacian spectra of the path graph on 4 vertices. We have $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (2, 3), (3, 4)\}$. Here is a picture of the graph, and its Laplacian matrix:

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

While obvious, it is important to see that the ith row and ith column in the Laplacian matrix correspond to the ith vertex. Really, we should identify the rows and columns with V. Similarly, an eigenvector should be viewed as a vector in $\mathbb{R}^V$, or better yet a function from the vertices to the Reals. Here is an eigenvector, to three digits of precision

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} .653 \\ .271 \\ -.271 \\ -.653 \end{pmatrix} = .586 \begin{pmatrix} .653 \\ .271 \\ -.271 \\ -.653 \end{pmatrix}$$

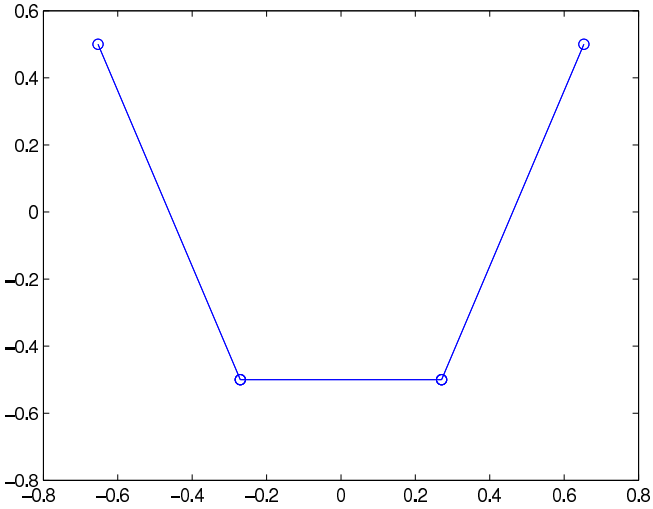
1.5 Spectral Embeddings

The quickest way to convince you that eigenvectors tell you something about a graph is to show you an example. It won't tell you why, but it will convince you that something interesting is going on. Let's begin by computing another eigenvector of the path on 4 vertices:

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} .5 \\ -.5 \\ -.5 \\ .5 \end{pmatrix} = 2 \begin{pmatrix} .5 \\ -.5 \\ -.5 \\ .5 \end{pmatrix}$$

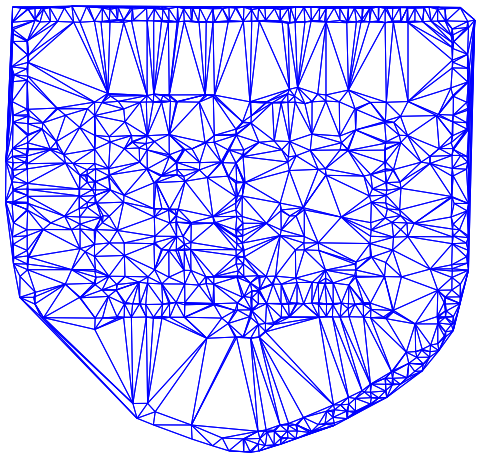
These two eigenvectors provide us two functions from V to the Reals. We will call the first vector v_2 and the second v_3. I will now use these two vectors to draw the graph, by locating vertex i at point $(v_2(i), v_3(i))$. If I then fill in the edges, I get the following figure:

```
A = diag(ones(1,3),1);
A = A + A';
L = lap(A);
[v,d] = eig(L);
gplot(A,[v(:,2),v(:,3)],'o')
hold on
gplot(A,[v(:,2),v(:,3)])
```



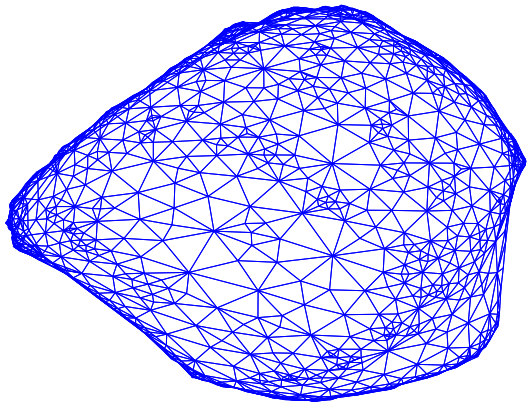
Let's do a more interesting example. I generated the following graph by sampling pixels in the Yale Coat of Arms with probability proportional to their darkness. Those pixels became vertices. I then generated the Delaunay triangulation of these points to form the following graph:

```
load yaleShieldBig
gplot(A,xy)
```



Since the vertices came with coordinates, it was easy to draw a nice picture of the graph. But, what if we just knew the graph, and not the coordinates? We could generate coordinates by computing two eigenvectors, and using each as a coordinate. In particular, I will take the eigenvectors corresponding to the second and third-smallest eigenvalues of the graph Laplacian, call them v_2 and v_3, and plot vertex i at $(v_2(i), v_3(i))$. I then draw the edges as straight lines between their corresponding vertices.

```
L = lap(A);
[v,d] = eigs(L, 3, 'sm');
gplot(A,v(:,[1 2]))
```



That's a great way to draw a graph if you start out knowing nothing about it. It's the first thing I do whenever I meet a strange graph. Note that the middle of the picture is almost planar, although edges do cross near the boundaries.

1.6 Graph Isomorphism

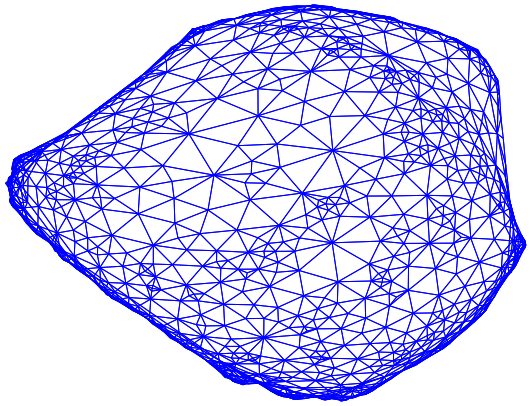
It is important to note that the eigenvalues do not change if we relabel the vertices. Moreover, if we permute the vertices then the eigenvectors are similarly permuted. That is, if P is a permutation matrix, then

$$Lv = \lambda v \quad \text{if and only if} \quad (PLP^T)(Pv) = \lambda(Pv),$$

because $P^T P = I$.

To prove it by experiment, let's randomly permute the vertices, and plot the permuted graph.

```
p = randperm(length(A));
Ap = A(p,p);
Lp = lap(Ap);
[vp,dp] = eigs(Lp, 3, 'sm');
gplot(Ap,vp(:,[1 2]))
```



Note that this picture is slightly different from the previous one: it has flipped vertically. That's because eigenvectors are only determined up to signs, and that's only if they have multiplicity 1. This gives us a very powerful heuristic for testing if one graph is a permutation of another (this is the famous "Graph Isomorphism Testing Problem"). First, check if the two graphs have the same sets of eigenvalues. If they don't, then they are not isomorphic. If they do, and the eigenvalues have multiplicity one, then draw the pictures above. If the pictures are the same, up to horizontal or vertical flips, and no vertex is mapped to the same location as another, then by lining up the pictures we can recover the permutation.

As some vertices can map to the same location, this heuristic doesn't always work. We will learn about it to the extent to which it does. In particular, we will see that if every eigenvalue of two graphs G and H have multiplicity 1, then in polynomial time we can test whether or not they are isomorphic. If every eigenvalue has multiplicity at most k, then we will see how to test isomorphism in time $n^{O(k)}$.

Even these algorithms don't kill off the graph isomorphism problem. There are graphs in which every non-trivial eigenvalue has large multiplicity. We will learn how to construct and analyze these, as they constitute fundamental examples and counter-examples to many natural conjectures.

```
>> load lsg25
>> L = lap(A);
>> e = eig(L)'

e =

  Columns 1 through 8

    0.0000   10.0000   10.0000   10.0000   10.0000   10.0000   10.0000   10.0000
```

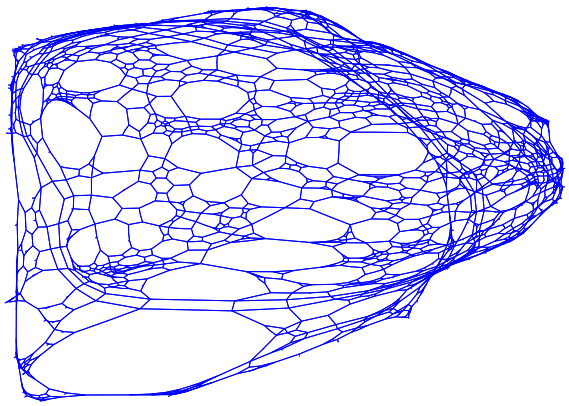
```
  Columns 9 through 16

  10.0000   10.0000   10.0000   10.0000   10.0000   15.0000   15.0000   15.0000

  Columns 17 through 24

  15.0000   15.0000   15.0000   15.0000   15.0000   15.0000   15.0000   15.0000

  Column 25
  15.0000
```

Determining the complexity of graph isomorphism is a maddening task. For every pair of graphs I have ever seen, it has been easy to tell whether or not they are isomorphic. However, the best bound on the worst-case complexity for the problem is $2^{O(\sqrt{n})}$.

1.7 More Spectral Embeddings

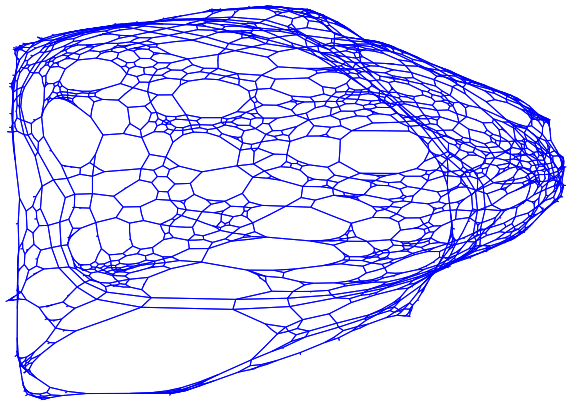
Here is a graph I found in which vertices represent intersections in Rome.

```
load rome
L = lap(A);
[v,d] = eigs(L,3,'sm');
gplot(A,v(:,[1 2]))
```



The next one comes by associating a vertex with everyone who as co-authored a paper with Paul Erdös, and placing edges between people who were co-authors on the same paper with Erdös. This graph had multiple zero eigenvalues, which gave Matlab some trouble. The extra code is a work-around.

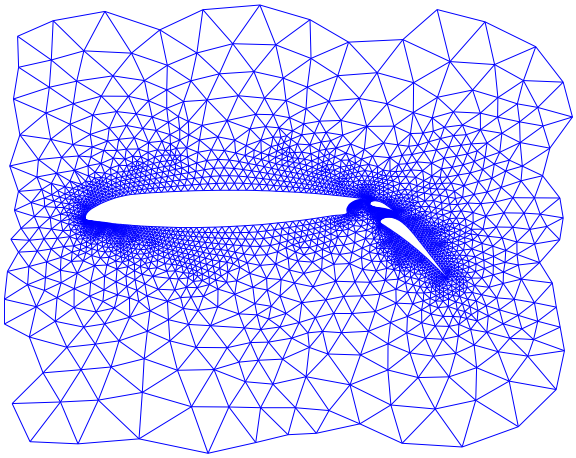
```
load erdosGraph
L = lap(A);
L = L + eye(length(L))/10^8;
[v,d] = eigs(L,7,'sm');
gplot(A,v(:,[1 2]))
```



That graph didn't look too good. That's to be expected. Not all graphs were meant to be drawn in the plane.

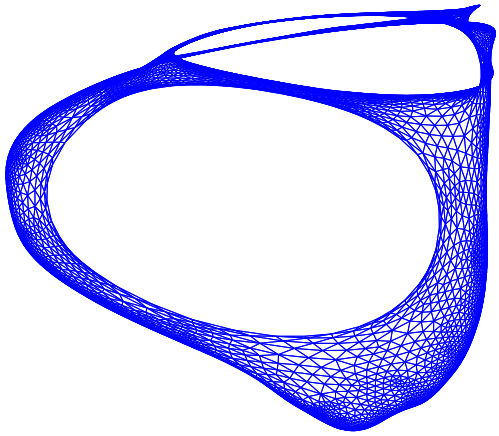
Here's the famous airfoil graph.

```
load airfoil
A = sparse(i,j,1,4253,4253);
A = A + A';
gplot(A,[x,y])
```



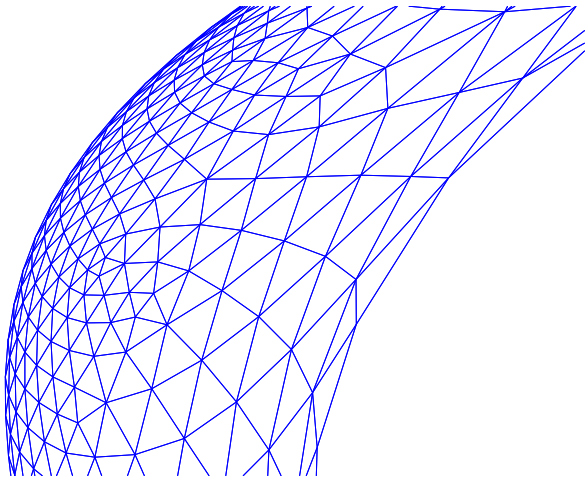
And, here's its spectral embedding.

```
L = lap(A);
[v,d] = eigs(L,3,'sm');
gplot(A,v(:,[1 2]))
```



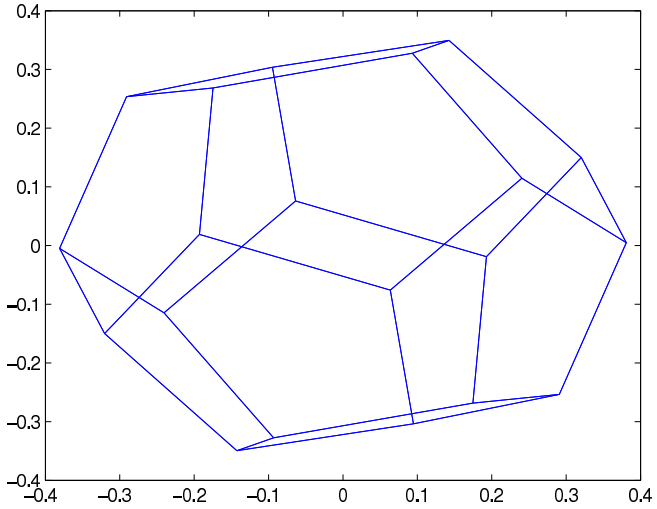
Let's zoom in on a small portion, so that we can see how nice the embedding is.

```
set(gca,'XLim',[-.032, -.024])
set(gca,'YLim',[-.0023, .0052])
```



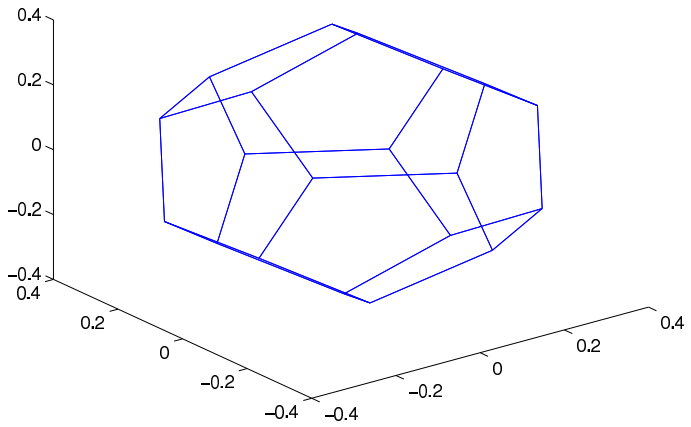
Finally, let's look at a spectral embedding of the edges of the dodecahedron.

```
load dodec
L = lap(A);
[v,d] = eigs(L,3,'sm');
gplot(A,v(:,[1 2]))
```



You will notice that this looks like what you would get if you squashed the dodecahedron down to the plane. The reason is that we really shouldn't be drawing this picture in two dimensions: the smallest non-zero eigenvalue of the Laplacian has multiplicity three. So, we can't reasonably choose just two eigenvectors. We should be choosing three that span the eigenspace. If we do, we would get the canonical representation of the dodecahedron in three dimensions.

```
[v,d] = eigs(L,4,'sm');
gplot3(A,v(:,[1 2 3]))
```



As you would guess, this happens for all Platonic solids. In fact, if you properly re-weight the edges, it happens for every graph that is the one-skeleton of a convex polytope. Let me state that more concretely. A weighted graph is a graph along with a weight function, w, mapping every edge to a positive number. The adjacency matrix of a weighted graph has the weights of edges as its entries, instead of 1s. The diagonal degree matrix of a weighted graph, D_G, has the weighted degrees on

its diagonal. That is,

$$D_G(i,i) = \sum_{j:(i,j)\in E} w(i,j).$$

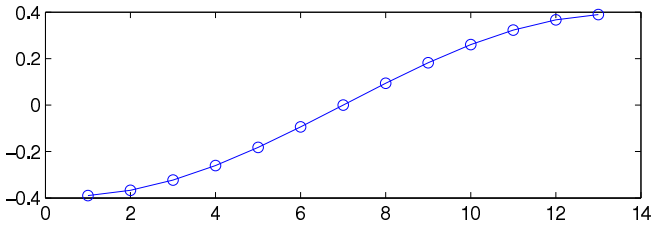
The Laplacian then becomes $L_G = D_G - A_G$. Given a convex polytope in $\mathbb{R}^d$, we can treat its 1-skeleton as a graph on its vertices. We will prove that there is a way of assigning weights to edges so that the second-smallest Laplacian eigenvalue has multiplicity d, and so that the corresponding eigenspace is spanned by the coordinate vectors of the vertices of the polytope.

This brings us to the study of the Colin de Verdière parameter of graph. Colin de Verdière's parameter of a graph is the maximum multiplicity of the second-smallest Laplacian eigenvalue over all edge-weight functions. Colin de Verdière proved that it is three if and only if the graph is planar! It is 2 if and only if the graph is outer-planar (planar, with all vertices on the convex hull), and 4 if and only if the graph is linkless embeddable in $\mathbb{R}^3$. The world is still waiting to find out what it means when it is 5.

1.8 Physical Intuition

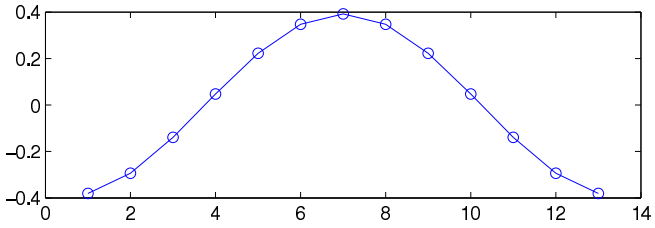
One can gain some intuition for the spectra of graphs by associating them with physical systems. For example, the path graph can be understood as a discretization of a string, and its Laplacian eigenvectors as discretizations of its fundamental modes of vibration. For special graphs, this connection can be made rigorous through the Finite Element Method. In the following figure, I construct the path graph on 13 vertices, and then plot the second Laplacian eigenvector v_2 by plotting the points $(i, v_2(i))$. I have neglected the first Laplacian eigenvector, as it is a constant vector.

```
A = diag(ones(1,12),1);
A = A + A';
L = lap(A);
[v,d] = eig(L);
plot(v(:,2));
hold on; plot(v(:,2),'o');
```



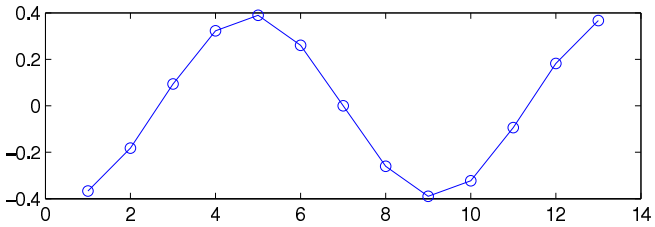
Here is the third eigenvector:

```
plot(v(:,3));
hold on; plot(v(:,3),'o');
```



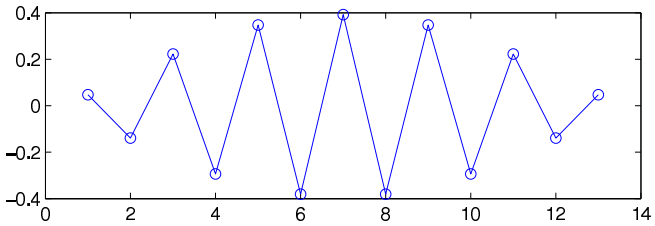
The fourth:

```
plot(v(:,4));
hold on; plot(v(:,4),'o');
```



And, the thirteenth:

```
plot(v(:,13));
hold on; plot(v(:,13),'o');
```



1.9 The Fiedler Value

The multiplicity of zero in the spectrum of the Laplacian of a graph is equal to the number of connected components in the graph. If we let the eigenvalues of the Laplacian be

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n,$$

then $\lambda_2 = 0$ if and only if the graph is disconnected.

Whenever you see a qualitative statement like this, you should try to make it quantitative. This was Fiedler's inspiration for calling λ_2 the "algebraic connectivity of a graph". He proved that the further λ_2 is from 0, the better connected the graph is. We will cover the ultimate extension of this result: Cheeger's inequality.

In short, we say that a graph is poorly connected if one can cut off many vertices by removing only a few edges. We measure how poorly connected it is by the ratio of these quantities (almost). Cheeger's inequality gives a tight connection between this ratio and λ_2. If λ_2 is small, then for some t, the set of vertices

$$S_i \stackrel{\text{def}}{=} \{i : v_2(i) < t\}$$

may be removed by cutting much less than $|S_i|$ edges. This spectral graph partitioning heuristic has proved very successful in practice.

1.10 Expanders

We will be particularly interested in graphs that are very well connected. These are called *expanders*. Roughly speaking, expanders are sparse graphs (say a number of edges linear in the number of vertices), in which λ_2 is bounded away from zero by a constant. They are among the most important examples of graphs, and play a prominent role in Theoretical Computer Science.

Expander graphs have numerous applications. We will see how to use expander graphs to construct pseudo-random generators *about which one can actually prove something*. We will also use them to construct good error-correcting codes.

Conversely, we will learn how to construct expanders. We will see the group theoretic constructions of Margulis and Lubotzky, Phillips and Sarnak, although their analysis will be beyond the scope of this class. Instead, we will cover a new, very simple, construction of expanders that it is easy to understand (along with it's slightly more complicated cousin: the Zig-Zag product). We will also see how to construct expanders from error-correcting codes.

1.11 Approximations of Graphs

We will ask what it means for one graph to approximate another. Given graphs G and H, we will measure how well G approximates H by the closeness of their Laplacian quadratic forms. We will see that expanders are precisely the sparse graphs that provide good approximations of the complete graph, and we will use this perspective for most of our analysis of expanders. We will show that every graph can be well-approximated by a sparse graph through a process called *sparsification*.

We will also ask how well a graph can be approximated by a tree, and see that low-stretch spanning-trees provide good approximations under this measure.

My motivation for this material is not purely graph-theoretic. Rather, it is inspired by the need to design fast algorithms for computing eigenvectors of Laplacian matrices and for solving linear equations in Laplacian matrices. This later problem arises in numerous contexts, including the solution of elliptic PDEs by the finite element method, the solution of network flow problems by interior point algorithms, and in classification problems in Machine Learning.

In fact, our definition of graph approximation is designed to suit the needs of the Preconditioned Conjugate Gradient algorithm, and we will see how good graph approximations lead to fast algorithms for solving linear equations.

1.12 Diameter and Concentration of Measure

We will also use the analysis of the Conjugate Gradient algorithm to derive results in graph theory. We will first use it to derive bounds on the diameter of a graph in terms of eigenvalues.

More interestingly, we will use it to derive tails bounds on random variables. Imagine that we have a function f of k independent random $\{0,1\}$-valued random variables. If f satisfies a Lipschitz condition (that is it does not change too much when one variables changes value), then one can prove statements like

$$\Pr\left[f(x_1,\ldots,x_k) > t + \text{median}(f)\right] < \text{somethingSmall}(t).$$

From just knowing the eigenvalues of the hypercube, we can prove almost optimal statements of this form.

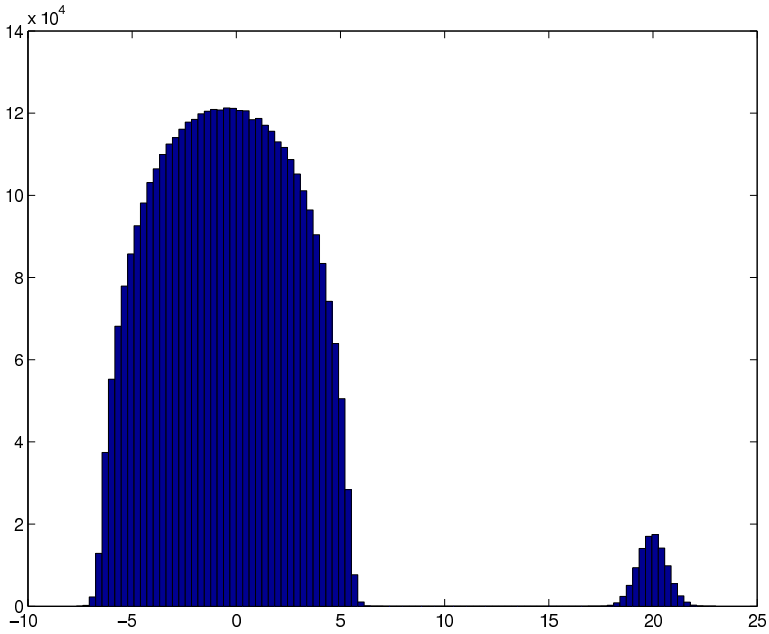
The nice thing about this graph-theoretic approach is that it immediately carries over to many other graphs. For example, we will be able to make similar statements for functions on the unit sphere, hyper-torii, and size-k subsets of another set.

1.13 Spectra of Random Graphs

There are two fancy ways to construct graphs: by algebraic constructions and at random. It turns out that random graphs are almost as nice as algebraic graphs, and are sometimes nicer. If they are large enough, random graphs should really be thought of as a special family of graphs. We will see that the spectra of random graphs are very well behaved.

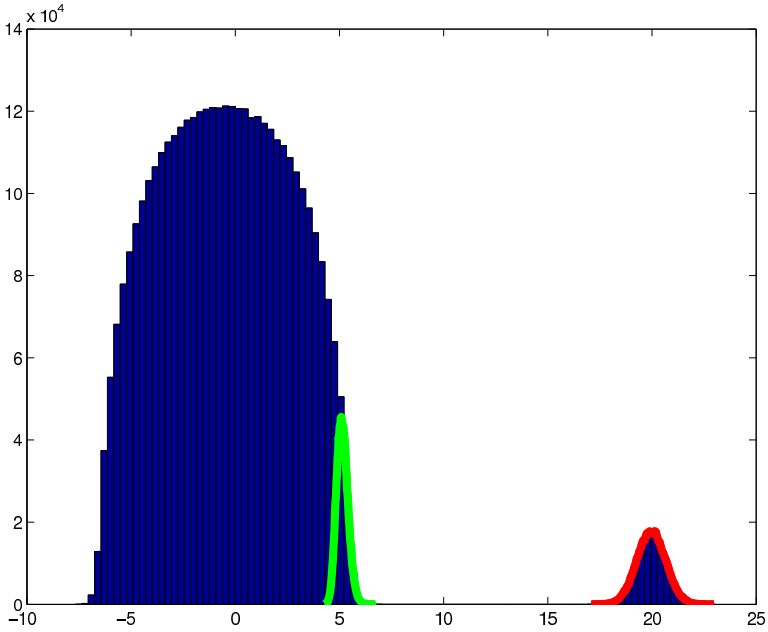
The most natural random graph is one in which each edge is chosen to exist with probability 1/2, independently from all the others. We will consider the adjacency matrix of such a graph, and the density function of its eigenvalues. To generate this density, we will compute a histogram. That is, we will generate many adjacency matrices, compute their eigenvalues, and then histogram all the eigenvalues (lumping them all together). Here's the Matlab code.

```
d = 40;
its = 100000;
E = zeros(d,its);
for i = 1:its,
a = rand(d) > .5;
a = triu(a,1);
a = a + a';
e = sort(eig(a));
E(:,i) = e;
end
[y,x] = hist(E(:),100);
hist(E(:),100);
```



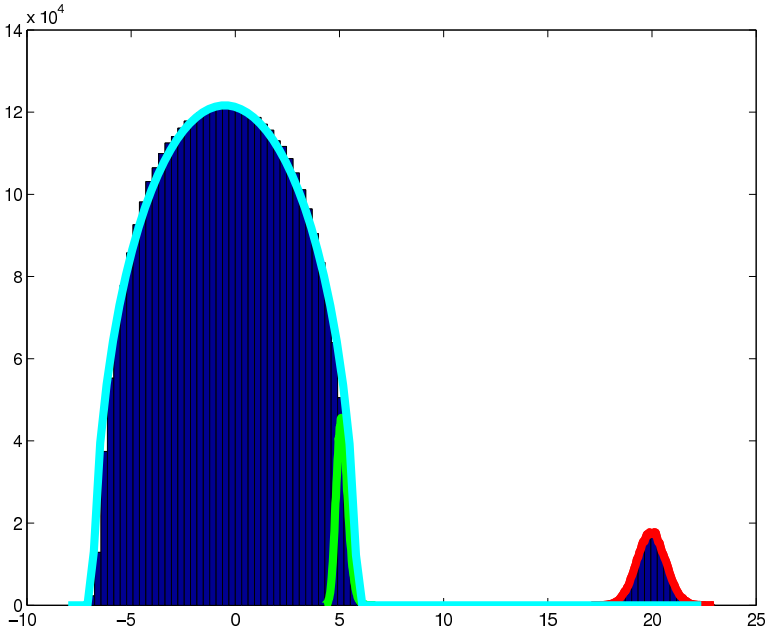
Now, I'll add in histograms of just the largest and second-to-largest eigenvalues.

```
[y1,x1] = hist(E(40,:),100);
sc = x(2) - x(1);
sc1 = x1(2) - x1(1);
hold on
plot(x1,y1 *sc / sc1, 'r',...
   'LineWidth',5)
[y2,x2] = hist(E(39,:),100);
sc2 = x2(2) - x2(1);
plot(x2,y2 *sc / sc2, 'g',...
   'LineWidth',5)
```



Note that these distributions are very tightly concentrated. Almost everything that we are seeing here can be proved analytically. This includes the form of the big hump in the limit.

```
x1 = x / sqrt(d);
s1 = sqrt(1-x1.^2);
z = s1 / s1(23) * y(23);
plot(x-1/2,z,'c','LineWidth',5)
```

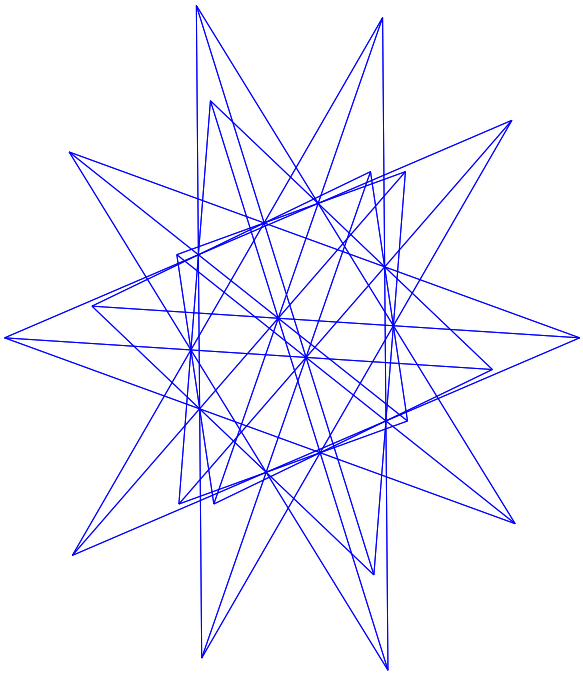


1.14 Cliques and Graph Coloring

Some of the earliest applications of spectral graph theory were in bounding the clique and chromatic numbers of graphs. Recall that a coloring of a graph is function $c : V \to \{1, \ldots, k\}$, for some k, so that for all $(i, j) \in E$, $c(i) \neq c(j)$. The chromatic number of a graph is the smallest k for which

the graph has a coloring. As colorings try to make adjacent vertices far apart, it is natural to look for graph colorings in the high-frequency eigenvectors of the Laplacian. Here is a drawing of the dodecahedron using its highest-frequency eigenvectors:

```
L = lap(A);
[v,d] = eig(full(L));
gplot(A,v(:,[19 20]))
```



We will see relations between the eigenvalues of a graph and its clique and chromatic numbers. We will also examine coloring heuristics that exploit the eigenvectors. While these heuristics do not always work, we will use our analysis of the spectra of random graphs to see that they work on quasi-random problems. These are problems in which we design a k-colorable graph at random by first dividing the vertices into k sets, and then choosing edges between vertices in different sets at random.

We will use the same analysis to see how to find cliques planted in quasi-random graphs, and to analyze spectral partitioning heuristics for quasi-random graphs.

1.15 Directed Graphs

We will finish the semester with a few lectures on the spectral theory of directed graphs. This is a theory in its infancy.

1.16 Mechanics

I expect to continue to produce lecture notes for every lecture. There will be around 6 problem sets during the semester. You will be allowed to work on them in small groups. There will be no final exam.