Homework 1

CS 292F : Laplacian Matrices and Spectra : Spring 2022

Out: Monday, April 4. Due: Monday, April 18.

Homework policy. Please turn in all your homework through GradeScope as a single .pdf file. For computational problems, turn in listings of any code you wrote; any sample output and plots; and an interactive session transcript as a Matlab diary file, Jupyter notebook .pdf, etc.

You are allowed to discuss the problems in groups of up to three (you and up to two others), but you must write up the solutions on your own. If you do work with anyone, you must acknowledge your collaborators. Also, you must cite any references you use other than the textbook and your class notes.

You are forbidden from searching the web in any way to find answers to these problems. However, you are welcome to search the web for advice on using Matlab, Julia, and other software tools. Stackoverflow is great.

Problem 1. Incidence matrix. Let G be an undirected graph with n vertices and m edges. An *incidence matrix* of G (sometimes called a *vertex-edge incidence matrix*) is an n-by-m matrix U, not square in general, with one column for each edge of G. The column corresponding to edge (i, j) has just two nonzeros, one in row i and one in row j, one with value 1 and one with value -1. Notice that the incidence matrix isn't unique—I didn't specify which entry in each column is positive and which is negative, and I also didn't specify which column corresponds to each edge.

1.1. Choose any connected 5-vertex graph. Draw a picture of that graph, with vertices labeled 1 through 5. Display both the Laplacian matrix and an incidence matrix of that graph.

1.2. Prove that if U is any incidence matrix for a graph G, then $L = UU^T$ is the (unique) Laplacian matrix of G.

1.3. Suppose graph G is connected. What is the rank of U? Justify your answer.

1.4. A *tree* is a connected, undirected graph with no cycles. Any tree with n vertices has $n - 1$ edges. A *spanning tree* of a connected graph G is a subgraph of G that is a tree and that includes all the vertices of G.

Let G and U be as above, let T be some spanning tree of G, and let V be an incidence matrix of T. Note that V is n-by-$(n - 1)$. Give two proofs that there exists an $(n - 1)$-by-m matrix W with $U = VW$: first, by reasoning about the ranks and column spaces of the matrices; and, second, by an explicit combinatorial construction of W, one column at a time, using the fact that an edge of G corresponds to both a column of U and a column of W.

Problem 2. Augmented matrix. Let G be an undirected graph with n vertices and m edges, let L be the Laplacian matrix of G, and let U be an incidence matrix of G. An *augmented matrix* of G is a square, symmetric $(n+m)$-by-$(n+m)$ matrix B defined in block form as

$$B = \begin{pmatrix} I & U^T \\ U & 0 \end{pmatrix}.$$

Suppose we are given an n-vector b and we want to solve the Laplacian linear system $Lx = b$ for the n-vector x. Give a linear system $By = c$ with the augmented matrix whose solution y contains the solution to $Lx = b$.

Problem 3. Another explanation of graph drawing. Let G be any graph on n vertices, and let $K = K_n$ be the complete graph on n vertices. Prove that

$$\frac{\lambda_2(G)}{n} = \min \frac{x^T L_G x}{x^T L_K x},$$

where the min is over vectors x with $x^T \mathbf{1} = 0$. This leads to another way to get some intuition about Hall's Laplacian-eigenvector graph drawing method. Since $x^T L_K x = \sum_{1 \le a < b \le n} (x(a) - x(b))^2$, the denominator may be viewed as a measure of the average closeness of the points. So, the quotient

$$\frac{x^T L_G x}{x^T L_K x}$$

measures the average distance between vertices connected by edges divided by the average distance between all pairs of vertices.

Problem 4. Graph automorphisms (harder). Let A be a symmetric matrix and let P be a permutation matrix such that $PAP^T = A$ and $P^2 \neq I$. Prove that A has an eigenvalue of multiplicity at least 2, that is, an eigenvalue λ for which A has two linearly independent eigenvectors. (Hint: If w is an eigenvector for λ, then so is Pw. Thus you want to find an eigenvector such that w and Pw are linearly independent, i.e. Pw is not in the span of w.)

Note that Laplacians of cycle graphs admit such permutations, but those of path graphs do not.

Problem 5. Setting up tools. The purpose of this problem is just to get an interactive environment of your choice set up so that you can play with sparse matrices and graphs. There are several possibilities:

- **Matlab** has efficient sparse matrix computation built in, and the CS 292F GitHub page contains a Matlab toolbox called "meshpart" that can compute Laplacians, plot pictures of graphs, and generate various kinds of sample graphs. Matlab is kind of old-fashioned as a language, but it is the environment that I know best. You can run Matlab at CSIL or install it on your own machine; see `https://www.software.ucsb.edu/info/matlab`.

  If you've never used Matlab before, it's pretty simple. Try running it and reproducing some of the demos I've done in class; the transcripts are on the class web site. Saying "help" gets you some general help; "help foo" gets you information on the function `foo()`. Try saying "help meshpart", and then take a look at the source code in the `Matlab/Meshpart` directory (on the course GitHub site) to see what it looks like.

- **Julia** has a lot of packages for computing with sparse matrices (including for example the Arpack package for eigenvalues and eigenvectors), and Dan Spielman, the author of our textbook, has a nifty package called "Laplacians.jl" with lots of algorithms on Laplacians, plus plots and sample graphs. Julia is a very cool language, two of whose authors came from UCSB, but I myself am currently very much a Julia novice. You can download Julia from `https://julialang.org/downloads/` and run it as a Docker image, from an interactive shell prompt, or in a Jupyter notebook. Our course GitHub site has a link to Laplacians.jl and a sample Jupyter notebook run called `SpielmanLecture1.ipynb`.

- **Python** has sparse matrix support through its well-documented "scipy" package, and some graph tools in its "networkx" package. It's probably got the most complete ecosystem around graphs, matrices, plotting, and so forth, most of which comes bundled in Anaconda, from `https://www.anaconda.com/products/individual#Downloads`. I run Python both from a shell prompt (with `ipython`) and in a Jupyter notebook. I don't know of any Python packages specifically designed for graph Laplacians.

- If there's another environment where you're comfortable and can find suitable graph and sparse matrix tools (Mathematica, R, etc.), feel free to use it instead.

You can do the rest of this problem in the environment of your choice. There are no limits on collaboration for this problem. On the contrary, I encourage you to post to Piazza any questions you have or obstacles you encounter while getting things set up, and also any advice or recipes you have for others in the class. I give some hints for Matlab below, but I'd love to see people do this in other environments as well.

5.1. In your environment of choice, type in the Laplacian matrix L of your 5-vertex graph from problem 1.1, and find its eigenvalues and eigenvectors (`eig()` in Matlab; say "`help eig`"). Either by hand or by writing code, create the incidence matrix U of L. Verify by computer that $UU^T = L$.

Form the augmented matrix B, by using this Matlab code or your environment's version thereof:

```
I = speye(something);   % sparse identity matrix, you choose "something"
Z = sparse(something);  % sparse zero matrix, you choose "something"
B = [I U' ; U Z];       % the augmented matrix
B                       % print B in sparse format
full(B)                 % print B in dense format
```

Compute the eigenvalues and eigenvectors of B (`eig(full(B))` in Matlab), and compare them to the eigenvalues and eigenvectors of L.

5.2. Load some of the sample Laplacian matrices from the `Data` directory on the coure GitHub site. (You can use `load matname` in Matlab to load a Laplacian matrix from `matname.mat`, or `@load matname.jld2` in Julia to load it from `matname.jld2`; in other environments you might want to start from the edge list in `matname.txt`.) Find the Fiedler value and vector, i.e. the second smallest eigenvalue of the Laplacian matrix and its eigenvector. (In Matlab, you can try both the built-in `eigs()` routine and the `fiedler()` routine from Meshpart toolbox. In Julia, you can get `eigs()` from the Arpack package, or you can use Spielman's routines from Laplacians.jl.)

You should check your results against `matname.v`, whose first column is the Fiedler vector. It's only 8 significant digits so you won't get exact agreement. But you can compare the Rayleigh quotient $w_2^T L w_2$ with your computed λ_2 for both your computed Fiedler vector and the one in `matname.v`, and those should agree to at least 6 digits or so.

5.3. Go to the SuiteSparse Matrix Collection at `https://sparse.tamu.edu`. The Collection has groups of matrices from many sources; some of them come from graphs. The website has tools to search and download the matrices in several formats.

Browse the collection and find some matrices that come from undirected graphs. One place to start is by filtering for the "Pajek" group. That group contains both undirected and directed graphs (symmetric and unsymmetric matrices); stick to the undirected ones, and ignore any edge weights. Convert each symmetric matrix to the Laplacian matrix of the corresponding unweighted graph (`laplacian()` in Matlab, `lap()` in Laplacians.jl, or it's easy enough to write your own converter). If one of your matrices is not connected, its Fielder value will be 0; you should get its largest connected component (e.g. Matlab/Meshpart routine `components()`) and use that.

What's the largest graph from the collection you can find a Fiedler vector for within a minute of wall-clock time on your computer?