TOPIC 2

Spectral and singular value decompositions

4 Introduction to spectral graph theory

Course: Math 535 (http://www.math.wisc.edu/~roch/mmids/) - Mathematical Methods in Data Science (MMiDS)
Author: Sebastien Roch (http://www.math.wisc.edu/~roch/), Department of Mathematics, University of Wisconsin-Madison
Updated: Sep 21, 2020

We give an introduction to spectral graph theory.

4.1 Matrices associated to graphs

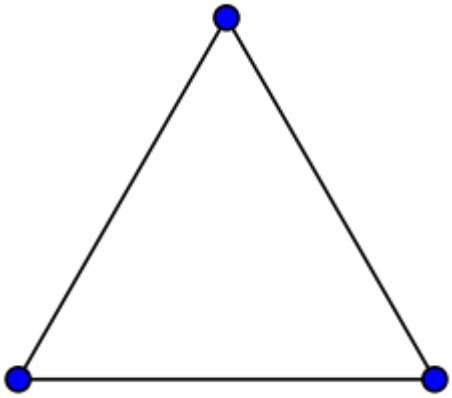
A convenient way of specifying a graph is the following matrix representation.

Definition (Adjacency Matrix): Assume the undirected graph $G = (V, E)$ has $n = |V|$ vertices. The adjacency matrix A of G is the $n \times n$ symmetric matrix defined as

$$A_{xy} = \begin{cases} 1 & \text{if } \{x, y\} \in E \\ 0 & \text{o.w.} \end{cases}$$

◁

Example (Triangle): The adjacency matrix of a triangle:



(Source (https://commons.wikimedia.org/wiki/File:Complete_graph_K3.svg))

that is, 3 vertices with all possible edges between them, is

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

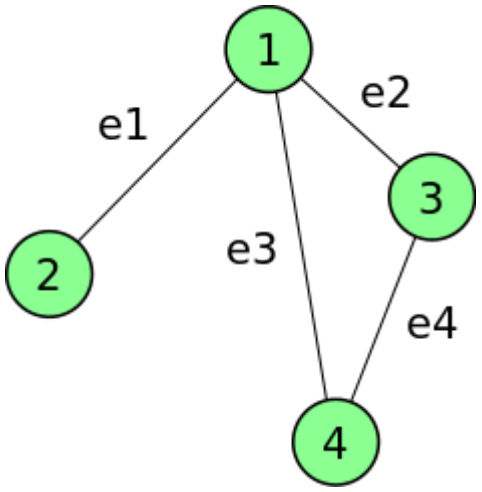
◁

Exercise: Let A^n be the n-th matrix power of the adjacency matrix A of a graph $G = (V, E)$. Prove that the (i, j)-th entry a_{ij}^n is the number of paths of length exactly n between vertices i and j in G. (Hint: Use induction on n.) ◁

Another useful matrix associated to a graph is its incidence matrix. For convenience, we assume that the vertices of $G = (V, E)$ are numbered $1, \ldots, |V|$ and that the edges are labeled arbitrarily as $e_1, \ldots, e_| E|$.

Definition (Incidence Matrix): The incidence matrix of an undirected graph $G = (V, E)$ is the $n \times m$ matrix B, where $n = |V|$ and $m = |E|$ are the numbers of vertices and edges respectively, such that $B_{ij} = 1$ if the vertex i and edge e_j are incident and 0 otherwise. $\triangleleft$

Example: The incidence matrix of the following graph:



(Source (https://commons.wikimedia.org/wiki/File:Labeled_undirected_graph.svg))

is given by

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

$\triangleleft$

In the digraph case, we have the definitions are adapted as follows. The adjacency matrix A of a digraph $G = (V, E)$ is the matrix defined as

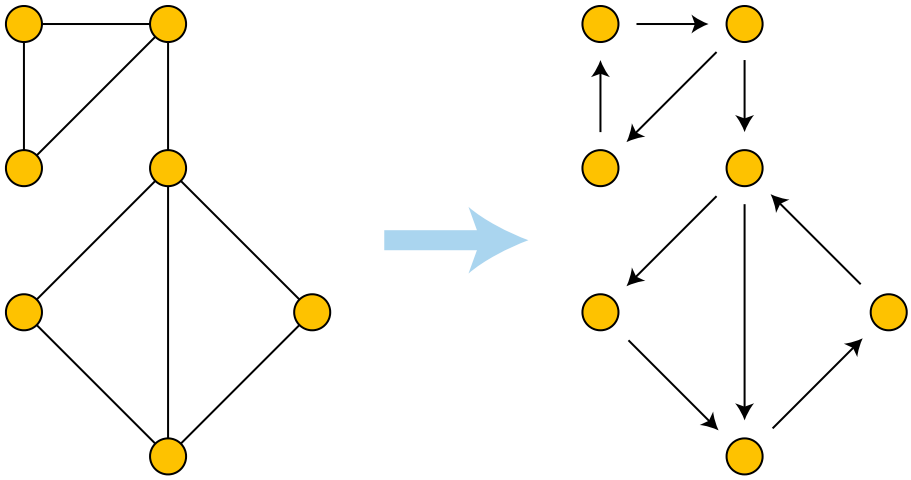
$$A_{xy} = \begin{cases} 1 & \text{if } (x, y) \in E \\ 0 & \text{o.w.} \end{cases}$$

The oriented incidence matrix of a digraph G with vertices $1, \ldots, n$ and edges $e_1, \ldots, e_m$ is the matrix B such that $B_{ij} = -1$ if egde e_j leaves vertex i, $B_{ij} = 1$ if egde e_j enters vertex i, and 0 otherwise.

An orientation of an (undirected) graph $G = (V, E)$ is the choice of a direction for each of its edges, turning it into a digraph.

Definition (Oriented Incidence Matrix): An oriented incidence matrix of an (undirected) graph $G = (V, E)$ is the incidence matrix of an orientation of G. ◁

Example: Here is an example of an orientation:



(Source (https://11011110.github.io/blog/2019/01/17/orientations-infinite-graphs.html))

◁

NUMERICAL CORNER Using LightGraphs.jl (https://github.com/JuliaGraphs/LightGraphs.jl), the adjacency matrix of a graph can be obtained with `adjacency_matrix` (https://juliagraphs.github.io/LightGraphs.jl/latest/linalg/#LightGraphs.LinAlg.adjacency_matrix). By default, it returns a `SparseArrays` (https://docs.julialang.org/en/v1/stdlib/SparseArrays/) array, which can be converted to a regular array with `Array` (https://docs.julialang.org/en/v1/base/arrays/#Core.Array).

In [4]:
```
#Julia version: 1.5.1
using LightGraphs, GraphPlot, LinearAlgebra
```

In [2]:
```
g = complete_graph(3)
```

Out[2]:
```
{3, 3} undirected simple Int64 graph
```

In [3]:
```
A = adjacency_matrix(g)
```

Out[3]:
```
3×3 SparseArrays.SparseMatrixCSC{Int64,Int64} with 6 stored entries:
  [2, 1]  =  1
  [3, 1]  =  1
  [1, 2]  =  1
  [3, 2]  =  1
  [1, 3]  =  1
  [2, 3]  =  1
```

In [4]:
```
Array(A)
```

Out[4]:
```
3×3 Array{Int64,2}:
 0  1  1
 1  0  1
 1  1  0
```

In [5]:
```
g = smallgraph(:petersen)
A = Array(adjacency_matrix(g))
```

Out[5]:
```
10×10 Array{Int64,2}:
 0  1  0  0  1  1  0  0  0  0
 1  0  1  0  0  0  1  0  0  0
 0  1  0  1  0  0  0  1  0  0
 0  0  1  0  1  0  0  0  1  0
 1  0  0  1  0  0  0  0  0  1
 1  0  0  0  0  0  0  1  1  0
 0  1  0  0  0  0  0  0  1  1
 0  0  1  0  0  1  0  0  0  1
 0  0  0  1  0  1  1  0  0  0
 0  0  0  0  1  0  1  1  0  0
```

The incidence matrix is obtained with `incidence_matrix` (https://juliagraphs.github.io/LightGraphs.jl/latest/linalg/#LightGraphs.LinAlg.incidence_matrix) - again as a `SparseArrays`.

In [6]:
```
Array(incidence_matrix(g))
```

Out[6]:
```
10×15 Array{Int64,2}:
 1  1  1  0  0  0  0  0  0  0  0  0  0  0  0
 1  0  0  1  1  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  0  1  1  0  0  0  0  0  0  0  0
 0  0  0  0  0  1  0  1  1  0  0  0  0  0  0
 0  1  0  0  0  0  0  1  0  1  0  0  0  0  0
 0  0  1  0  0  0  0  0  0  0  1  1  0  0  0
 0  0  0  0  1  0  0  0  0  0  0  0  1  1  0
 0  0  0  0  0  0  1  0  0  0  1  0  0  0  1
 0  0  0  0  0  0  0  0  1  0  0  1  1  0  0
 0  0  0  0  0  0  0  0  0  1  0  0  0  1  1
```

The numbering above is indeed consistent with:

In [7]:
```
j = 0 # initialize edge counter
for e_j in edges(g)
    j += 1
    @show j, src(e_j), dst(e_j)
end
```

```
(j, src(e_j), dst(e_j)) = (1, 1, 2)
(j, src(e_j), dst(e_j)) = (2, 1, 5)
(j, src(e_j), dst(e_j)) = (3, 1, 6)
(j, src(e_j), dst(e_j)) = (4, 2, 3)
(j, src(e_j), dst(e_j)) = (5, 2, 7)
(j, src(e_j), dst(e_j)) = (6, 3, 4)
(j, src(e_j), dst(e_j)) = (7, 3, 8)
(j, src(e_j), dst(e_j)) = (8, 4, 5)
(j, src(e_j), dst(e_j)) = (9, 4, 9)
(j, src(e_j), dst(e_j)) = (10, 5, 10)
(j, src(e_j), dst(e_j)) = (11, 6, 8)
(j, src(e_j), dst(e_j)) = (12, 6, 9)
(j, src(e_j), dst(e_j)) = (13, 7, 9)
(j, src(e_j), dst(e_j)) = (14, 7, 10)
(j, src(e_j), dst(e_j)) = (15, 8, 10)
```

4.2 Laplacian matrix

Our main matrix of interest is the Laplacian matrix.

It is a graph analogue of the Laplace-Beltrami operator (https://en.wikipedia.org/wiki/Laplace–Beltrami_operator) in differential geometry. We will show in particular that it contains useful information about the connectedness of the graph and we will describe an application to graph partitioning in the next section. But first some theory.

4.2.1 Definition and basic properties

Recall that, given a graph $G = (V, E)$, the quantity $\delta(v)$ denotes the degree of $v \in V$. The key definition is the following.

Definition (Laplacian Matrix): Let $G = (V, E)$ be a graph with vertices $V = \{1, \dots, n\}$ and adjacency matrix $A \in \mathbb{R}^{n \times n}$. Let $D = \mathrm{diag}(\delta(1), \dots, \delta(n))$ be the degree matrix. The Laplacian matrix associated to G is defined as $L = D - A$. Its entries are

$$l_{ij} = \begin{cases} \delta(i) & \text{if } i = j \\ -1 & \text{if } \{i, j\} \in E \\ 0 & \text{o.w.} \end{cases}$$

◁

Observe that the Laplacian matrix L of a graph G is symmetric:

$$L^T = (D - A)^T = D^T - A^T = D - A$$

where we used that both D and A are themselves symmetric. The associated quadratic form is particularly simple and will play an important role.

Lemma (Laplacian Quadratic Form): Let $G = (V, E)$ be a graph with $n = |V|$ vertices. Its Laplacian matrix L is PSD and furthermore we have the following formula for the Laplacian quadratic form

$$\mathbf{x}^T L\mathbf{x} = \sum_{e=\{i,j\}\in E} (x_i - x_j)^2$$

for any $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$.

Proof: Let B be an oriented incidence matrix of G. We claim that $L = BB^T$. Indeed, for $i \neq j$, entry (i, j) of BB^T is a sum over all edges containing i and j as endvertices, of which there is at most one. When $\{i, j\} \in E$, that entry is -1, since one of i or j has a 1 in the column of B corresponding to e and the other one has a -1. For $i = j$, letting b_{xy} be entry (x, y) of B,

$$(BB^T)_{ii} = \sum_{e=\{x,y\}\in E: i\in e} b_{xy}^2 = \delta(i).$$

Thus, for any $\mathbf{x}$, we have $(B^T\mathbf{x})_k = x_v - x_u$ if the edge $e_k = \{u, v\}$ is oriented as (u, v) under B. That implies

$$\mathbf{x}^T L\mathbf{x} = \mathbf{x}^T BB^T\mathbf{x} = \|B^T\mathbf{x}\|^2 = \sum_{e=\{i,j\}\in E} (x_i - x_j)^2.$$

Since the latter is always nonnegative, it also implies that L is PSD. □

As a convention, we denote the eigenvalues of a Laplacian matrix L by

$$0 \leq \mu_1 \leq \mu_2 \leq \cdots \leq \mu_n.$$

Another important observation:

Lemma ($\mu_1 = 0$): Let $G = (V, E)$ be a graph with $n = |V|$ vertices and Laplacian matrix L. The constant unit vector

$$\mathbf{y}_1 = \frac{1}{\sqrt{n}}(1, \ldots, 1)^T$$

is an eigenvector of L with eigenvalue 0.

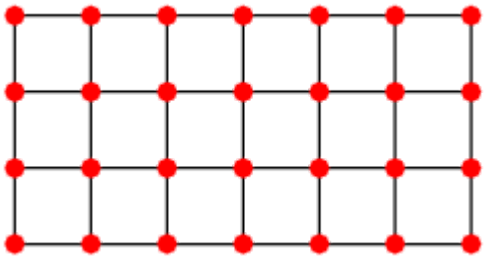
Proof: Let B be an oriented incidence matrix of G recall that $L = BB^T$. By construction $B^T\mathbf{y}_1 = \mathbf{0}$ since each column of B has exactly one 1 and one -1. So $L\mathbf{y}_1 = BB^T\mathbf{y}_1 = \mathbf{0}$ as claimed. □

In general, the constant vector may not be the only eigenvector with eigenvalue one.

Exercise: Let G be a graph with two connected components. Show that its Laplacian L has at least two linearly independent unit eigenvectors with eigenvalue zero. [Hint: Write L as a block matrix and use the lemma above.] ◁

NUMERICAL CORNER One use of the spectral decomposition of the Laplacian matrix is in graph drawing. We illustrate this next. Given a graph $G = (V, E)$, it is not clear a priori how to draw it in the plane since the only information available are adjacencies of vertices. One approach is just to position the vertices at random. The function `gplot` provides a functionality to supply a `layout` function that returns an x and y-coordinate for each vertex.

We will test this on a grid graph. Sometimes a picture is worth a thousand words. This is an example of a 4×7-grid graph.



(Source (https://mathworld.wolfram.com/GridGraph.html))

We use `grid` (https://juliagraphs.github.io/LightGraphs.jl/latest/generators/#LightGraphs.SimpleGraphs.grid-Union{Tuple{AbstractArray{T,1}},%20Tuple{T}}%20where%20T<:Integer) to construct such a graph.

In [5]:
```
g = grid([4,7])
```

Out[5]:
```
{28, 45} undirected simple Int64 graph
```

Next, we write a function that returns independent uniform coordinates in $[0, 1]$ for the vertices.

In [6]:
```
function rand_locs(g)
    return rand(nv(g)), rand(nv(g)) # random x and y coord for each vert
ex
end
```

Out[6]:
```
rand_locs (generic function with 1 method)
```

Here is what it looks like on the graph `g` . Each row below gives the x and y-coordinates of one vertex.

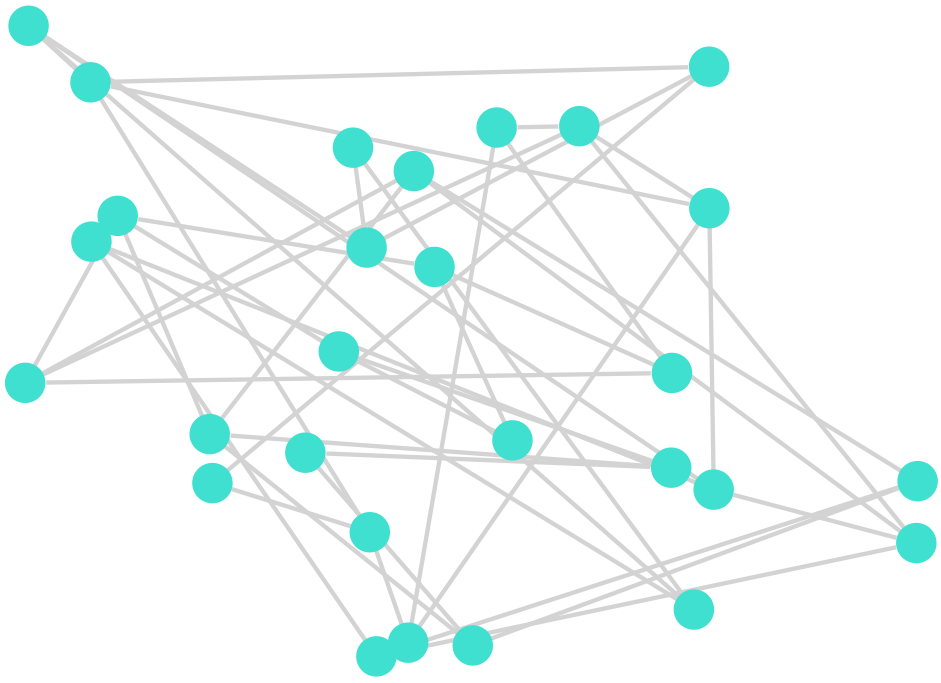
In [7]:
```
(x, y) = rand_locs(g)
hcat(x,y)
```

Out[7]:
```
28×2 Array{Float64,2}:
 0.191692   0.606737
 0.989004   0.352409
 0.189396   0.51508
 0.816165   0.587919
 0.436631   0.0819465
 0.50328    0.586298
 0.3384     0.835183
 0.740025   0.818151
 0.754872   0.119381
 0.699313   0.38022
 0.437357   0.593008
 0.771573   0.61867
 0.0918346  0.227644
 ⋮
 0.2729     0.787315
 0.0667081  0.801595
 0.391096   0.387632
 0.14777    0.355164
 0.247434   0.194785
 0.328025   0.0842352
 0.406501   0.271673
 0.452194   0.910649
 0.699653   0.304761
 0.736827   0.244049
 0.791831   0.205615
 0.854796   0.181293
```

Providing these coordinates to `gplot` through the `layout` option gives the following drawing of the graph.

In [9]:
```
gplot(g, layout=rand_locs)
```

Out[9]:



Clearly, this is a lot harder to read than the original graph above.

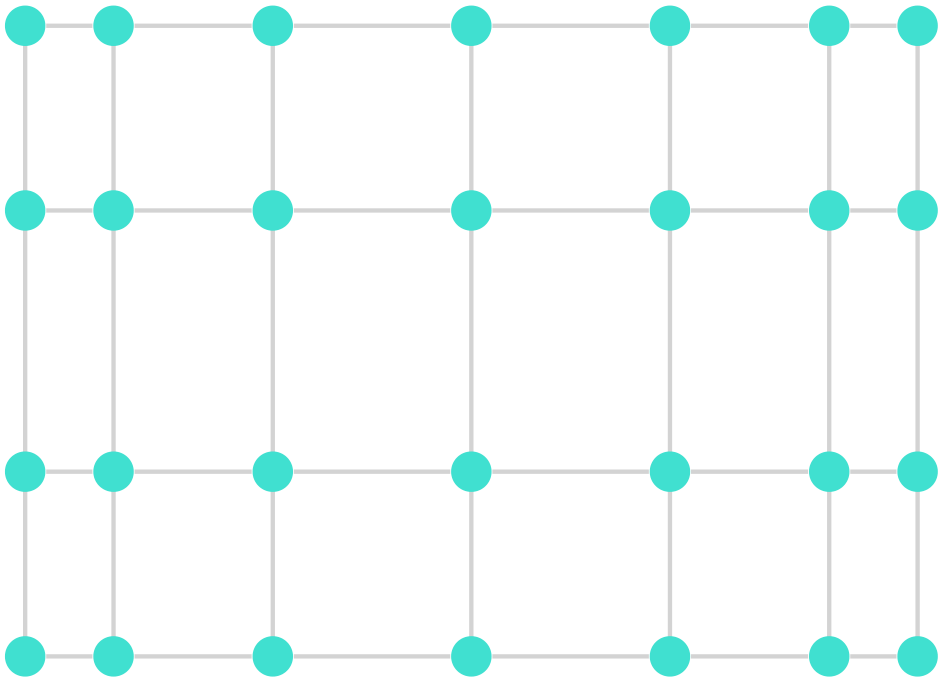
Another approach is to map the vertices to two eigenvectors, similarly to what we did for dimensionality reduction. The eigenvector associated to μ_1 is constant and therefore not useful for drawing. We try the next two. We use `laplacian_matrix` (https://juliagraphs.github.io/LightGraphs.jl/latest/linalg/#LightGraphs.LinAlg.laplacian_matrix-Union{Tuple{AbstractGraph{U}},%20Tuple{U},%20Tuple{AbstractGraph{U},DataType}}%20where%20U) to compute the Laplacian matrix of the graph. By default, the function `eigen` (https://docs.julialang.org/en/v1/stdlib/LinearAlgebra/#LinearAlgebra.eigen), lists eigenvalues in increasing order (if they real).

In [12]:
```
function spec_locs(g)
    L = Array(laplacian_matrix(g)) # laplacian L of g
    F = eigen(L) # spectral decomposition
    v = F.vectors # eigenvectors
    return v[:,2], v[:,3] # returns 2nd and 3rd eigenvectors
end
```

Out[12]:
```
spec_locs (generic function with 1 method)
```

In [13]:
```
gplot(g, layout=spec_locs)
```

Out[13]:



Interestingly, the outcome is very similar to the origina, more natural drawing. We will come back later to try to explain this, after we have developed further understanding of the spectral properties of the Laplacian matrix.

4.2.2 Laplacian and connectedness

As we said before, the Laplacian matrix contains information about the connectedness of G. We elaborate on a first connection here. Recall that, for any graph G, the Laplacian eigenvalue $\mu_1 = 0$.

Lemma (Laplacian and Connectedness): If G is connected, then the Laplacian eigenvalue $\mu_2 > 0$.

Proof: Let $G = (V, E)$ with $n = |V|$ and let $L = \sum_{i=1}^{n} \mu_i \mathbf{y}_i \mathbf{y}_i^T$ be a spectral decomposition of its Laplacian L with $0 = \mu_1 \leq \cdots \leq \mu_n$. Suppose by way of contradiction that $\mu_2 = 0$. Any eigenvector $\mathbf{y} = (y_1, \ldots, y_n)^T$ with 0 eigenvalue satisfies $L\mathbf{y} = \mathbf{0}$ by definition. By the Laplacian Quadratic Form Lemma then

$$0 = \mathbf{y}^T L\mathbf{y} = \sum_{e=\{i,j\} \in E} (y_i - y_j)^2.$$

In order for this to hold, it must be that any two adjacent vertices i and j have $y_i = y_j$. Furthermore, because G is connected, between any two of its vertices u and v - adjacent or not - there is a path $u = w_0 \sim \cdots \sim w_k = v$ along which the y_w's must be the same. Thus $\mathbf{y}$ is a constant vector. But that is a contradiction since the eigenvectors $\mathbf{y}_1, \ldots, \mathbf{y}_n$ are in fact linearly independent, so that $\mathbf{y}_1$ and $\mathbf{y}_2$ cannot both be a constant vector. □

We state the following more general result without proof.

Lemma (Number of Connected Components): If μ_{k+1} is the smallest nonzero Laplacian eigenvalue of G, then G has k connected components.

We will be interested in more quantitative results of this type. Before proceeding, we start with a simple observation. By our proof of the Spectral Theorem, the largest eigenvalue μ_n of the matrix Laplacian L is the solution to the optimization problem

$$\mu_n = \max\{\langle \mathbf{x}, L\mathbf{x} \rangle : \|\mathbf{x}\| = 1\}.$$

Such extremal characterization is useful in order to bound the eigenvalue μ_n, since any choice of $\mathbf{x}$ with $\|\mathbf{x}\| = 1$ gives a lower bound through the quantity $\langle \mathbf{x}, L\mathbf{x} \rangle$. That perspective will be key to our application to graph partitioning.

For now, we give a simple consequence.

Lemma (Laplacian and Degree): Let $G = (V, E)$ be a graph with maximum degree $\bar{\delta}$. Let μ_n be the largest eigenvalue of its matrix Laplacian L. Then

$$\mu_n \geq \bar{\delta} + 1.$$

Proof idea: As explained before the statement of the lemma, it suffices to find a good test unit vector $\mathbf{x}$ to plug into $\langle \mathbf{x}, L\mathbf{x} \rangle$. A clever choice does the trick.

Proof: Let $u \in V$ be a vertex with degree $\bar{\delta}$. Let $\mathbf{z}$ be the vector with entries

$$z_i = \begin{cases} \bar{\delta} & \text{if } i = u \\ -1 & \text{if } \{i,u\} \in E \\ 0 & \text{o.w.} \end{cases}$$

and let $\mathbf{x}$ be the unit vector $\mathbf{z}/\|\mathbf{z}\|$. By definition of the degree of u, $\|\mathbf{z}\|^2 = \bar{\delta}^2 + \bar{\delta}(-1)^2 = \bar{\delta}(\bar{\delta}+1)$. Using the *Laplacian Quadratic Form Lemma*,

$$\langle \mathbf{z}, L\mathbf{z}\rangle = \sum_{e=\{i,j\}\in E} (z_i - z_j)^2 \geq \sum_{i:\{i,u\}\in E} (z_i - z_u)^2 = \sum_{i:\{i,u\}\in E} (-1-\bar{\delta})^2 = \bar{\delta}(\bar{\delta}+1)^2$$

where we restricted the sum to those edges incident with u and used the fact that all terms in the sum are nonnegative. Finally

$$\langle \mathbf{x}, L\mathbf{x}\rangle = \left\langle \frac{\mathbf{z}}{\|\mathbf{z}\|}, L\frac{\mathbf{z}}{\|\mathbf{z}\|} \right\rangle = \frac{1}{\|\mathbf{z}\|^2}\langle \mathbf{z}, L\mathbf{z}\rangle = \frac{\bar{\delta}(\bar{\delta}+1)^2}{\bar{\delta}(\bar{\delta}+1)} = \bar{\delta}+1$$

so that

$$\mu_n = \max\{\langle \mathbf{x}', L\mathbf{x}'\rangle : \|\mathbf{x}'\| = 1\} \geq \langle \mathbf{x}, L\mathbf{x}\rangle = \bar{\delta}+1$$

as claimed. □

NUMERICAL CORNER We construct a graph with two connected components and check the results above. Rather than using LightGraphs.jl (https://github.com/JuliaGraphs/LightGraphs.jl), we work directly with the adjacency matrix.

```
In [14]: A = [0 1 1 0 0; 1 0 1 0 0; 1 1 0 0 0; 0 0 0 0 1; 0 0 0 1 0]
```

```
Out[14]: 5×5 Array{Int64,2}:
          0  1  1  0  0
          1  0  1  0  0
          1  1  0  0  0
          0  0  0  0  1
          0  0  0  1  0
```

Note the block structure.

The degrees can be obtained by summing the rows of the adjacency matrix.

```
In [15]: degrees = reshape(sum(A,dims=1),:)
```

```
Out[15]: 5-element Array{Int64,1}:
          2
          2
          2
          1
          1
```

In [16]:
```
D = Diagonal(degrees)
```

Out[16]:
```
5×5 Diagonal{Int64,Array{Int64,1}}:
 2  ·  ·  ·  ·
 ·  2  ·  ·  ·
 ·  ·  2  ·  ·
 ·  ·  ·  1  ·
 ·  ·  ·  ·  1
```

In [17]:
```
L = D - A
```

Out[17]:
```
5×5 Array{Int64,2}:
  2  -1  -1   0   0
 -1   2  -1   0   0
 -1  -1   2   0   0
  0   0   0   1  -1
  0   0   0  -1   1
```

In [18]:
```
F = eigen(L)
F.values
```

Out[18]:
```
5-element Array{Float64,1}:
 1.1102230246251565e-15
 1.3322676295501878e-15
 2.0
 3.0
 3.0
```

Observe that (up to numerical error) there are two 0 eigenvalues and that the largest eigenvalue is greater or equal than the maximum degree plus one.

To compute the Laplacian matrix, one can also use the function `laplacian_matrix` in LightGraphs.jl (https://github.com/JuliaGraphs/LightGraphs.jl). For example, the Laplacian of the Petersen graph is the following:

In [19]:
```
g = smallgraph(:petersen)
L = Array(laplacian_matrix(g))
```

Out[19]:
```
10×10 Array{Int64,2}:
  3  -1   0   0  -1  -1   0   0   0   0
 -1   3  -1   0   0   0  -1   0   0   0
  0  -1   3  -1   0   0   0  -1   0   0
  0   0  -1   3  -1   0   0   0  -1   0
 -1   0   0  -1   3   0   0   0   0  -1
 -1   0   0   0   0   3   0  -1  -1   0
  0  -1   0   0   0   0   3   0  -1  -1
  0   0  -1   0   0  -1   0   3   0  -1
  0   0   0  -1   0  -1  -1   0   3   0
  0   0   0   0  -1   0  -1  -1   0   3
```

In [20]:
```
F = eigen(L)
F.values
```

Out[20]:
```
10-element Array{Float64,1}:
 2.6645352591003757e-15
 1.9999999999999982
 1.9999999999999982
 1.9999999999999984
 2.0
 2.0000000000000027
 4.999999999999998
 5.000000000000001
 5.000000000000002
 5.0000000000000036
```