6.S978 Graphs, Linear Algebra, and Optimization – Fall 2015 October 1, 2015

Lecture 6

Lecturer: Aleksander Mądry *Scribe: Christina Lee*

1 Recap of the Last Lecture

Last time we consider the problem of solving a linear system

$$Ax = b,$$

where A is symmetric and positive definite, i.e., a PSD, matrix. This means that its eigenvalues are all real and positive. We enumerated them in a non-decreasing order $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

We intended to apply iterative methods to this task and our measure of progress relied on two notions: the *right-hand side error* $r(x) := b - Ax$ and the *left-hand side error* $e(x) := x - x^*$.

We cast the task of linear system solving as the following convex optimization problem,

$$\min_x g(x), \tag{1}$$

where

$$g(x) := \frac{1}{2}x^T Ax - b^T x = \frac{1}{2}\|e(x)\|_A^2 - \frac{1}{2}(x^*)^T Ax^*, \tag{2}$$

and $\|\cdot\|_A$ is an A-norm defined as

$$\|y\|_A^2 := y^T Ay.$$

That is, the function $g(x)$ is just a "shifted" version of the A-norm $\|e(x)\|_A^2$ of the left-hand side error of our candidate solution x.

To solve this optimization problem, we applied the gradient descent method to it. The resulting algorithm is presented as Algorithm 1.

Algorithm 1 Gradient descent method when applied to the optimization problem (1).

```
$x_1 \leftarrow 0$
for $s = 1 \dots T-1$ do
    $x_{s+1} \leftarrow x_s - \eta \nabla g(x_s) = x_s + \eta \cdot r(x_s)$
end for
return $x_T$
```

Last time, we exploited the fact that g is strong λ_1-convexity and λ_n-smooth to obtain a very strong bound on the running time of that algorithm. Specifically, we proved the following general theorem about the convergence of gradient descent method when applied to strongly convex and smooth functions.

Theorem 1 *Let f be a L-smooth and strongly ℓ-convex function and let us set $\eta = \frac{2}{L+\ell}$ in gradient descent method, then*

$$|f(x_T) - f(x^*)| \leq \frac{L}{2}\exp\left(-\frac{4(T-1)}{\kappa+1}\right)\|x_1 - x^*\|_2^2,$$

where $\kappa := \frac{L}{\ell}$ is called the condition number *of f.*

Applying the above theorem to Algorithm 1 and noting the fact that $\ell = \lambda_1$ and $L = \lambda_n$ for g, we get that the number of iterations T_ε to obtain $\|x_T\|_A^2 \leq \varepsilon$ is at most

$$T_\varepsilon = O\left(\kappa \ln\left(\lambda_n \|x^*\|_2^2 \varepsilon^{-1}\right)\right). \tag{3}$$

2 Building Polynomials with Gradient Descent

Looking at the iteration bound (3), we see that the dependence on ε^{-1} is only logarithmic – which is what we want – and the key factor influencing the complexity of Algorithm 1 is the linear dependence on the condition number $\kappa = \frac{\lambda_n}{\lambda_1}$. It is natural to wonder then: can we improve this dependence?

In order to answer this question, we need to develop a different perspective on Algorithm 1. To this end, observe that

$$r_{s+1} = b - Ax_{s+1} = b - A(x_s + \eta r_s) = (I - \eta A)r_s = (I - \eta A)^s r_1 = (I - \eta A)^s b,$$

where r_s is a shorthand for $r(x_s)$. Consequently, we have that

$$x_T = x_{T-1} + \eta r_{T-1} = \sum_{i=1}^{T-1} \eta (I - \eta A)^i b = p_{T,\eta}(A) b,$$

where

$$p_{T,\eta}(x) := \sum_{i=1}^{T-1} \eta (1 - \eta x)^i \tag{4}$$

is a univariate polynomial of degree-$(T-1)$ that was applied to the matrix A.

In other words, one can view the output x_T of the Algorithm 1 as an application of degree-$(T-1)$ polynomial $p_{T,\eta}(x)$ to A and then multiplying it by the vector b.

Now, observe that the Algorithm 1 and its analysis is completely basis independent. So, we can choose a basis that is most convenient for us. The right choice here is to work in the eigenbasis of the matrix A, i.e., the orthonormal basis given by the eigenvectors of A. In this basis, all the objects we are analyzing become extremely simple, that is, diagonal. In particular, the matrix A becomes

$$A = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n \end{bmatrix},$$

and the polynomial $p_{T,\eta}(x)$ applied to A is simply

$$p_{T,\eta}(A) = \begin{bmatrix} p_{T,\eta}(\lambda_1) & 0 & \dots & 0 \\ 0 & p_{T,\eta}(\lambda_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & p_{T,\eta}(\lambda_n) \end{bmatrix}.$$

To understand what is the property of the polynomial $p_{T,\eta}(x)$ that we are looking for, recall that ideally we would like to compute $x^* = A^{-1}b$. So, our goal is to have the matrix $p_{T,\eta}(A)$ to become as close to possible to being A^{-1}, which in the eigenbasis of A is just

$$A^{-1} = \begin{bmatrix} \lambda_1^{-1} & 0 & \dots & 0 \\ 0 & \lambda_2^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n^{-1} \end{bmatrix},$$

Consequently, we see that there is an underlying optimization task at hand. Namely, we want to find a (univariate) polynomial $p(z)$ that, on one hand, approximates the (univariate) function $\frac{1}{z}$ well at each point corresponding to the eigenvalues of A, i.e., has

$$p(\lambda_i) \approx \lambda_i^{-1}, \tag{5}$$

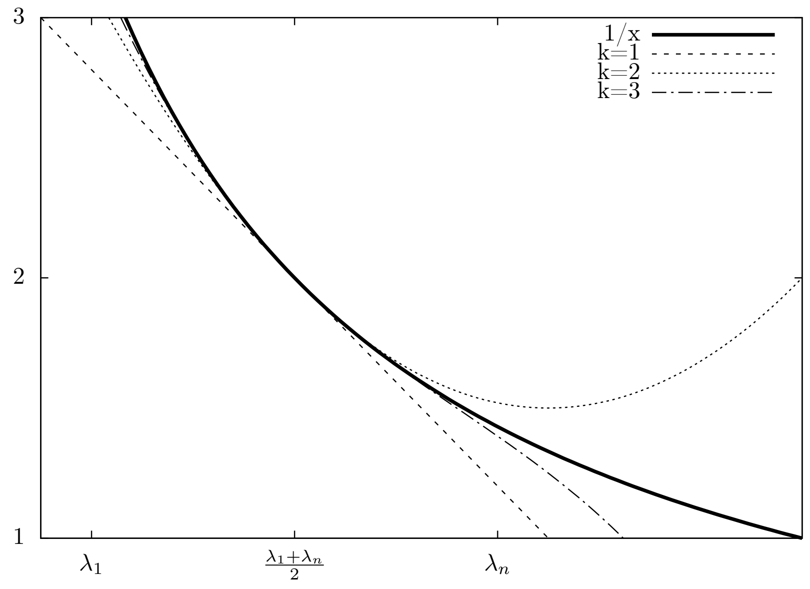

Figure 1: Approximation of the function $1/z$ around the point $z = \frac{\lambda_1+\lambda_n}{2}$ using Taylor approximation of orders $k = 1$, $k = 2$, and $k = 3$.

for all $i = 1, \ldots, n$, while, on the other hand, has its degree be at most $T - 1$. The latter constraint arises since the degree of this polynomial corresponds directly to the number of iterations Algorithm 1 takes. (Observe that if there was no upper bound on the degree of that polynomial then we could get an arbitrarily good approximation in (5).)

It turns out that the polynomial $p_{T,\eta}(z)$ (see (4)) that the Algorithm 1 uses is very special. Namely, it is actually the $(T-1)$-th order Taylor series approximation of the function $\frac{1}{z}$ around the point $\frac{1}{\eta} = \frac{\lambda_1+\lambda_n}{2}$. In other words, the way our gradient descent method algorithm approaches the approximation task (5) is by computing the $(T-1)$-th order Taylor approximation of the intended function $\frac{1}{z}$ around the point $\frac{\lambda_1+\lambda_n}{2}$, which is the middle of the interval $[\lambda_1, \lambda_n]$ in which all the points of interest lie. This is really remarkable that such a principled and sophisticated choice came up out of our very general gradient descent design scheme!

As we implicitly proved in the last lecture, the choice of polynomial $p_{T,\eta}$ leads to the bound (3), i.e., linear dependence of T on the condition number κ. Is there maybe a better choice of the polynomial?

In short, the answer is: yes. The intuition here is that Taylor polynomials are not really best suited for solving the approximation tasks as (5). They are invented with the goal of providing the best possible *local* approximation around the given point. In (5), however, we are interested in getting good approximation on a number of points that are spread out over the whole interval $[\lambda_1, \lambda_n]$. As a result, while the polynomial $p_{T,\eta}$ focuses on getting a very good fit around the middle point $\frac{\lambda_1+\lambda_n}{2}$, the corresponding much looser fit on the endpoints of that interval undermines the whole effort. See Figure 1 for an illustration.

Consequently, instead of the local approximation provided by Taylor series one should focus on Fourier-like series that were designed to provide such global type of approximation.

3 Conjugate Gradient Method: Beyond Taylor Approximation

Once we understood better the optimization question underlying our iterative linear system solving framework, let us try to abstract away the problem at hand and design a more direct algorithm for it.

To this end, let us note that in our framework each intermediate solution x_s, obtained after s

iterations, belongs to a so-called *Krylov subspace (of order s)* $\mathcal{K}_s$, defined as

$$\mathcal{K}_s := \mathrm{span}\left\{b, Ab, A^2b, \ldots A^{s-1}b\right\}.$$

Furthermore, each vector $x \in \mathcal{K}_s$ corresponds to multiplying some polynomial in the matrix A of degree at most $s-1$ by the vector b. So, in principle, if we knew what this polynomial is, it could be computed in s iterations in our framework.

Now, from this perspective, we can view the Algorithm 1 as an approach to choosing a sequence of specific points $x_s \in \mathcal{K}_s$ for $s = 1, \ldots, T$. As we discussed above, these choices correspond to taking successful higher-order Taylor series approximation of the function $1/x$ around the middle of the interval $[\lambda_1, \lambda_n]$. They, in turn, give us a solution to the approximation problem (5) that results in the iterations bound (3), which has a linear dependence on the condition number κ.

However, once we know what our real goal is: to choose a point x_T in the Krylov subspace $\mathcal{K}_T$ of order T that aims to solve the problem (5), there might be a better approach to achieving it. In particular, instead of picking an explicitly constructed point x_T in hopes it will result in good performance, we can make our algorithm solve the underlying optimization directly.

This idea leads to an algorithm called *conjugate gradient method* that is presented as Algorithm 2. In oder words, this algorithm simply chooses x_T to be minimizer of our proxy objective function g (see

Algorithm 2 Conjugate gradient method.

```
Compute
    $$x_T := \operatorname*{argmin}_{x \in \mathcal{K}_T} g(x). \qquad (6)$$
return $x_T$.
```

(2)) for our target error measure $\|e(x)\|_A^2$.

3.1 Efficient Implementation of the Conjugate Gradient Method

The first point we need to address is the implementation of the Algorithm 2. After all, our description of conjugate gradient method does not really explain how to find the point x_T efficiently. Also, the optimization problem (6) that defines the point x_T does not seem to be too different to our original optimization problem (1). Why should it be easier to solve?

It turns out that despite this seeming similarity to the problem (1), solving the optimization problem (6) can indeed be done efficiently. The key reason for that is the fact that this optimization problem exhibits a very convenient structure when one works in an appropriate basis. Specifically, a basis $v_1, \ldots, v_T$ that is A-orthogonal, i.e., a one in which each v_i and v_j with $i \neq j$ are orthogonal with respect to the *A-inner product* $\cdot_A$ defined as

$$x \cdot_A y := x^T A y.$$

In such A-orthonormal basis, our objective function g becomes separable, breaking down in T independent and simple to solve problems. Also, by applying Gram-Schmidt orthogonalization procedure in a careful manner, one can compute such an A-orthogonal basis $v_1, \ldots, v_T$ in only $O(T)$ (as opposed to $O(T^2)$) matrix-vertex multiplications of A. Working out the details of this implementation are a part of the Problem set 1. We will show there that indeed one can compute x_T in the conjugate gradient method can be implemented using only $O(T)$ matrix-vector multiplications of A overall.

3.2 Analyzing the Performance of the Conjugate Gradient Method

Once we discussed the implementation of the conjugate gradient descent, we can turn our attention to analyzing its performance. Specifically, once we know that computing the point x_T requires only $O(T)$ iterations of our framework, we would like to know how large T has to be to obtain the desired quality of the solution.

To this end, let us reformulate the underlying optimization question (5) into a slightly more convenient to work with form. More precisely, let us note that if our solution x_T is represented as $p_T(A)b$, for some degree-$(T-1)$ univariate polynomial $p(z)$, then we have that

$$e(x_T) = x_T - x^* = p_T(A)b - x^* = p_T(A)Ax^* - x^* = q_T(A)(-x^*), \tag{7}$$

where $q_T(z)$ is also a degree-T polynomial defined as

$$q_T(z) := 1 - zp_T(z). \tag{8}$$

So, there is a one-to-one correspondence between polynomials $p_T(z)$ and polynomials $q_T(z)$. Namely, any degree-$(T-1)$ polynomial $p_T(z)$ defines a degree-T polynomial $q_T(z)$ via (8). On the other hand, any degree-T polynomial $q_T(z)$ with $q_T(0) = 1$ defines a degree-$(T-1)$ polynomial $p_T(z) := \frac{q_T(z)-1}{z}$. (Note that the requirement that $q_T(0) = 1$ implies that the polynomial $q_T(z) - 1$ is divisible by z.)

Therefore, instead of thinking of the polynomials $p_T(z)$ and the corresponding problem (5) of approximating the $1/z$ function on the eigenvalue points, one can think of trying to design polynomials $q_T(z)$ that make the initial error $-x^*$ in (7) vanish as quickly as possibly. More formally, the latter task boils down to finding for a given desired error parameter ε a polynomial $q(z)$ that has as small degree as possible while satisfying

$$\begin{aligned} q(0) &= 1 \\ |q(\lambda_i)|^2 &\leq \varepsilon, \end{aligned} \tag{9}$$

for all $i = 1, \ldots, n$. Once we find such a polynomial, we will know that by (7)

$$\|e(x)\|_A^2 = \|q_T(A)(-x^*)\|_A^2 \leq \|q_T(A)\|_2^2 \|x^*\|_A^2 = \max_i |q_T(\lambda_i)|^2 \|x^*\|_A^2 \leq \varepsilon \|x^*\|_A^2,$$

where we also used the fact that the eigenvectors of the matrix $q_T(A)$ are $q_T(\lambda_1), \ldots, q_T(\lambda_n)$. This is exactly the error guarantee that we would like to get in our algorithm. It is also worth noting that the above error guarantee bound works for *any* M-norm $\|\cdot\|_M^2$, i.e., we have

$$\|e(x)\|_M^2 \leq \varepsilon \|x^*\|_M^2,$$

for any PSD matrix M, even if it is unrelated to A.

In the light of the above, from now on, we just need to focus exclusively on the question underlying the optimization problem (9). That is, to bound the minimum degree T_ε that a polynomial $q(z)$ has to have in order to satisfy the conditions (9).

Observe that this question is *no longer* algorithmic! It is now just a question from approximation theory, a well-studied branch of mathematics. In particular, we would be completely satisfied by a purely existential statements, i.e., the polynomial $q(z)$ of the minimal degree just has to exist, we do not need to be able to efficiently compute it. (Although, one can show that the constructions we will use below can be performed efficiently.)

Now, turning to that question, if we were able to have the polynomial $q(z)$ have degree at most n then we can just resort to interpolation to get a perfect fit at all the eigenvalues. That is, we can consider a degree-n polynomial $q_{int}(z)$ defined as

$$q_{int}(z) := \prod_{i=1}^{n} \left(1 - \frac{z}{\lambda_i}\right).$$

Clearly, $q_{int}(0) = 1$ and $q_{int}(\lambda_i) = 0$, for all i. So, $T_\varepsilon \leq n$ or, in other words, once we allow the conjugate gradient method run for n iterations, the computed point x_T will be exactly x^*.

The above statement should not be too surprising, as one could expect that the span of the Krylov subspace $\mathcal{K}_n$ is the whole of $\mathbb{R}^n$. Our key interest, however, is in obtaining bounds on T_ε that are much smaller than n. To get a hold on this regime, we need to resort to certain fundamental results in

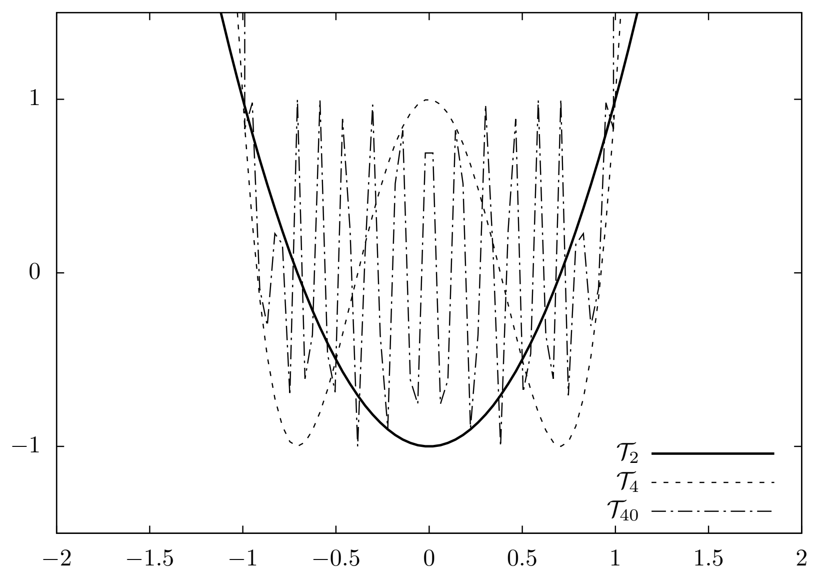

Figure 2: Chebyshev polynomials $\mathcal{T}_2$, $\mathcal{T}_4$, and $\mathcal{T}_{40}$.

approximation theory. Specifically, the object of our interest will be a family of extremal polynomials $\{\mathcal{T}_k(z)\}_k$ called *Chebyshev polynomials (of the first kind)*, defined as

$$\mathcal{T}_k(z) := \frac{1}{2}\left(\left(z+\sqrt{z^2-1}\right)^k + \left(z-\sqrt{z^2-1}\right)^k\right).$$

Note that, despite appearances, each $\mathcal{T}_k(z)$ is indeed a polynomial. (Try working this out for k being 1 and 2.) This polynomials are extremal in a number of ways. The way we will care about here most is that

$$|\mathcal{T}_k(z)| \leq 1, \tag{10}$$

when $z \in [-1,1]$ (they actually oscillate between -1 and 1 in that interval) and the value of $|\mathcal{T}_k(z)|$ has the sharpest increases outside of that interval among all the polynomials of degree k.

Let us consider now, for a given $k \geq 1$, a degree-k polynomial $q_k^*(z)$ defined as

$$q_k^*(z) := \frac{\mathcal{T}_k\left(\frac{\lambda_n+\lambda_1-2z}{\lambda_n-\lambda_1}\right)}{\mathcal{T}_k\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)}. \tag{11}$$

Observe that due to our normalization by $\mathcal{T}_k\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)$, we have that

$$q_k^*(0) = 1,$$

as desired. Furthermore, for any $z \in [\lambda_1, \lambda_n]$ we have by (10) that

$$\begin{aligned} |q_k^*(z)|^2 &\leq \mathcal{T}_k\left(\frac{\lambda_n+\lambda_1}{\lambda_n-\lambda_1}\right)^{-2} \qquad (12)\\ &= \mathcal{T}_k\left(\frac{\kappa+1}{\kappa-1}\right)^{-2} \\ &= 4\left(\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right)^{-2} \\ &\leq 4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2k} \leq 4\cdot\exp\left(-\frac{4k}{\sqrt{\kappa}+1}\right), \end{aligned}$$

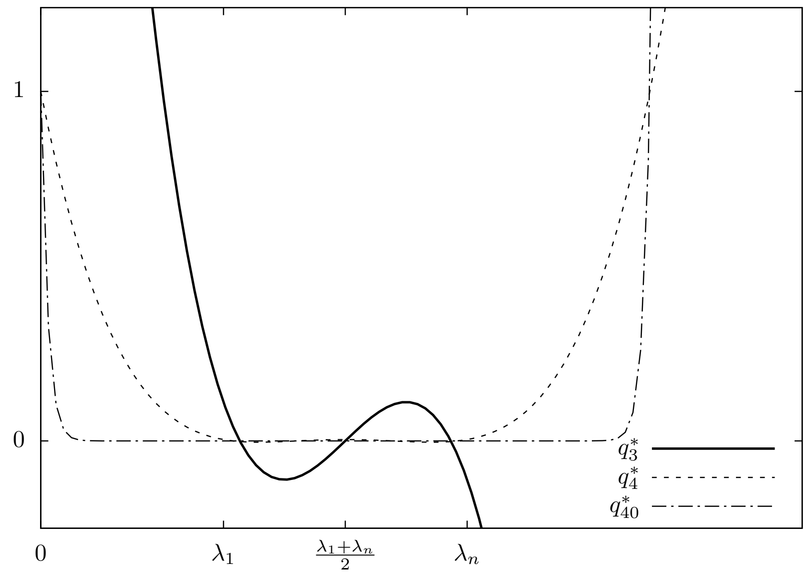

Figure 3: Polynomials q_3^*, q_4^*, and q_{40}^* over the interval $[\lambda_1, \lambda_n]$.

where $\kappa = \frac{\lambda_n}{\lambda_1}$ is the condition number and we used the fact that, by (10),

$$\left|\mathcal{T}_k\left(\frac{\lambda_n + \lambda_1 - 2z}{\lambda_n - \lambda_1}\right)\right| \leq 1,$$

for any $z \in [\lambda_1, \lambda_n]$.

As a result, by using this polynomial $q_k^*(z)$ to solve our optimization problem (9), we obtain the following bound on the performance of the conjugate gradient method.

Theorem 2 *After T iterations of the conjugate gradient method (Algorithm 2), we have that*

$$\|e(x_T)\|_A^2 \leq 4 \cdot \exp\left(-\frac{4k}{\sqrt{\kappa}+1}\right) \|x^*\|_A^2.$$

In other words, we obtain that number of iterations T_ε needed to get $\|e(x_T)\|_A^2 \leq \varepsilon$ is at most

$$T_\varepsilon \leq O\left(\sqrt{\kappa} \ln \frac{\|x^*\|_A}{\varepsilon}\right), \tag{13}$$

which improves our dependence on κ from linear (in (3)) to square root one.

Finally, it is important to note that the upper bound (12) on the value of our polynomial $q_k^*(z)$ holds not only for all the eigenvalue points λ_i, but actually it is valid for all z in the whole interval $[\lambda_1, \lambda_n]$. So, $q_k^*(z)$ provides a more robust solution than what is required by the problem (9).

4 Spectrum of the Laplacian Matrix

Recall that the reason why we were interested in solving linear systems was the need to compute ℓ_2-projection $\prod_{\mathcal{F}_{s,t}}$ onto the space of s-t flows. As we have shown in Lecture 4, this projection corresponds to electrical flow computations and these, in turn, boil down to solving a *Laplacian systems*, i.e., a linear system of the form

$$L\varphi = \sigma, \tag{14}$$

where L is the Laplacian matrix of the underlying graph, φ is the vector of vertex potentials we want to find, and σ is the vector of demands that we want our electrical flow to obey.

As we now have in hand algorithms for solving linear system, we need to examine how they can be applied in the context of Laplacian and what would be the resulting running times. This, in turn, requires us to analyze certain numerical properties of the Laplacian, such as its eigenvalue spectrum.

To this end, let us consider an undirected graph $G = (V, E, r)$ with $n = |V|$ vertices, $m = |E|$ edges, and edge resistances r_e assigned to each edge $e \in E$. Also, let us impose an arbitrary orientation on the edges of G. In Lecture 4, we defined the Laplacian of G as

$$L = B^T R^{-1} B,$$

where B is an $n \times m$ *edge-vertex incidence matrix* given by

$$B_{v,e} := \begin{cases} -1 & v \text{ is the head of } e \\ 1 & v \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases},$$

and R is an $m \times m$ diagonal *resistances matrix* defined as

$$R_{e,e'} := \begin{cases} r_e & \text{if } e = e' \\ 0 & \text{otherwise} \end{cases}.$$

However, there is a different, equivalent definition of the Laplacian. Namely, we have that

$$L = D - A,$$

where D is a diagonal $n \times n$ *(weighted) degree matrix* D defined as

$$D_{uv} := \begin{cases} \sum_{e \in N(u)} r_e^{-1} & \text{if u=v} \\ 0 & \text{otherwise,} \end{cases}$$

and A is an $n \times n$ *adjacency matrix* given by

$$A_{uv} := \begin{cases} r_e^{-1} & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, one can express the entries of the Laplacian explicitly as

$$L_{uv} := \begin{cases} \sum_{e \in N(u)} r_e^{-1} & \text{if } u = v \\ -r_e^{-1} & \text{if } e = (u, v) \in E \\ 0 & \text{otherwise,} \end{cases}$$

where $N(u)$ is the set of edges incident to the vertex u.

Interestingly, one can also decompose the Laplacian of G into a sum of elementary Laplacians

$$L = \sum_{e \in E} r_e^{-1} L^e = \sum_{e \in E} r_e^{-1} \chi_e \chi_e^T,$$

where each L^e is just a Laplacian of a simple graph on the vertex set V that contains only a single (unweighted) edge $e = (u, v)$ or, alternatively, an outer product of characteristic vectors χ_e with a -1 at u-th coordinate, 1 at v-th coordinate, and zeros at all the other coordinates. (So, each L^e is an $n \times n$ matrix with exactly four non-zero entries.)

Now, observe that L is a symmetric matrix. So, it has n real eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. Furthermore, one can prove the following theorem about them.

Theorem 3 *Let L be a Laplacian of a graph G and $\lambda_1 \leq \ldots \leq \lambda_n$ be its eigenvalues.*

(a) $\lambda_1 = 0$ and an all-ones vector $\vec{1}$ is an eigenvector corresponding to this eigenvalue;

(b) $\lambda_2 > 0$ *iff* G *is connected;*

(c) $\lambda_n \leq 2d_{\max}$*, where* $d_{\max}$ *is the maximum (weighted) vertex degree, i.e., the largest entry of the degree matrix* D*.*

(We will leave proving this theorem as a problem on the upcoming problem set.)

One of important implication of the above theorem is that L is *not* positive definite. That is, L is *positive semi-definite*, i.e., $L \succeq 0$, but $L \not\succ 0$. Even more importantly, as $\lambda_1 = 0$, the matrix L is also *not* invertible. This is very undesirable for us, as our algorithms for linear system solving assumed the underlying matrix is positive definite and, in particular, that it is invertible.

Fortunately, this turns out to be not too much of a problem. Observe that, as long as G is connected (which is always the case for us), the eigenspace of L corresponding to the eigenvalue 0 is just the one dimensional subspace of all the constant vectors, i.e., $\text{span}(\vec{1})$. So, the linear system (14) can be solved whenever the demand vector σ is orthogonal to that subspace. Formally, instead of considering the inverse L^{-1} of the Laplacian, which does not exist, one can consider the next best thing: the *Moore-Penrose pseudoinverse* L^{+} defined as

$$L^{+} := \sum_{i>1} \lambda_i^{-1} v_i v_i^T,$$

where v_i is the eigenvector corresponding to eigenvalue λ_i in the orthonormal eigenbasis of L. In other words, L^{+} behaves as an inverse of the Laplacian for every vector that is orthogonal to the eigenspaces corresponding to eigenvalue 0.

Consequently, the linear system (14) has a solution, given by $L^{+}\sigma$ as long as $\sigma \perp \vec{1}$. Observe, however, that the latter condition boils down to insisting that the demands of the demand vector σ sum up to zero. This must be always the case anyway if we want to compute a flow satisfying such demands. Therefore, the requirement $\sigma \perp \vec{1}$ is just a manifestation of the obvious inability to find an electrical flow (or any flow, for that matter) whose demands do not balance out.

Furthermore, one can check that once this $\sigma \perp \vec{1}$ condition is satisfied, our algorithms end up computing a correct solution with the condition number κ becoming the *pseudo-condition number*

$$\kappa^{+} := \frac{\lambda_n}{\lambda_2}.$$

Finally, it is worth pointing out that in this setting the solution that we end up computing will not be unique. That is, for any solution φ^* to the Laplacian system 14, the solution $\varphi^* + \alpha\vec{1}$, for any $\alpha \in \mathbb{R}$, will also be a solution. This is just a manifestation of the fact that Ohm's law is invariant under shifts of vertex potentials.