Chapter 5

Solving systems of linear equations

Let $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$ be given. We want to solve $Ax = b$. What is the fastest method you know?

We usually learn how to solve systems of linear equations via Gaussian elimination, or perhaps inverting A by first using the SVD, or some other kind of matrix factorization technique such as the Cholesky decomposition. These methods are called *direct* methods: they produce an exact solution (up to numerical precision) in a bounded number of iterations. Unfortunately they are also rather slow; for instance computing the inverse takes $O(n^3)$ time, and even just writing it down will usually require $O(n^2)$ space, even if the original matrix A has a very efficient representation — for instance, it is *sparse*, i.e. it contains only few non-zero entries (think of the adjacency matrix of a graph with low degree).

What we really want is an algorithm whose running time is related to the input size. If A is sparse, it can be specified succinctly by listing only the non-zero entries; the running time of any good algorithm should scale accordingly. In these lectures we will see a couple of algorithms that take advantage of sparsity. The first class of algorithms are called "iterative methods". These are simple and efficient in practice, but suffer from a poor dependence of the running time on the condition number of the matrix. A second class of algorithms is restricted to the case of so-called SDD systems (we will define these laters), but can be much more efficient (at least in theory), running in quasi-linear time with no dependence on the condition number.

5.1 Iterative methods

Let's first look into *iterative methods*. These kinds of algorithms only involve a sequence of matrix-vector multiplications by A (or related matrices). Any such multiplication can be done in time $O(m)$ where m is the number of nonzero entries of A, and hence the sparser A is the faster the algorithm will run. A drawback of these methods is that they never return the exact solution; instead the algorithm will slowly construct a better and better approximation to the solution x, and the running time will depend on the quality of approximation desired.

Later on we will return to direct methods, and explore how an approximate form of the

Cholesky decomposition can be computed very efficiently as well.

5.1.1 First-order Richardson iteration

For any $\alpha \in \mathbb{R}$,

$$\begin{aligned} Ax = b &\iff \alpha Ax = \alpha b \\ &\iff x = (\mathbb{I} - \alpha A)x + \alpha b. \end{aligned}$$

What's the point? We've written x as the fixed point of a simple linear equation. This suggests the following iterative process: set x^0 to anything, say $x^0 = 0$, and then iterate

$$x^t = (\mathbb{I} - \alpha A)x^{t-1} + \alpha b.$$

Note that if this process converges, $x^t \to_{t\to\infty} x^*$ for some $x^* \in \mathbb{R}^n$, then necessarily $Ax^* = b$. We are going to show that there is convergence as long as $\mathbb{I} - \alpha A$ has largest singular value strictly less than 1, and convergence time will depend on how much smaller than 1 it is.

For simplicity assume A is symmetric positive definite. This assumption is not as restrictive as it sounds, and in particular is no more restrictive than assuming that A is invertible. Indeed, given a system $Ax = b$, we can always multiply both sides by A^T and obtain $A^TAx = A^Tb$, where the matrix A^TA is now positive-definite (assuming A is invertible). Note that A^TA may not be sparse (even if A was), but we can still compute A^TAv in $O(nnz(A))$ operations for any vector v (we will also need the list of non-zero entries in every column for this).

Let $0 < \lambda_1 \leq \cdots \leq \lambda_n$ be the singular values of A. Then the singular values of $\mathbb{I} - \alpha A$ are $1 - \alpha\lambda_i$. Its largest singular value is $\max_i |1 - \alpha\lambda_i|$, and this is minimized by taking $\alpha = 2/(\lambda_1 + \lambda_n)$, in which case we get $1 - 2\lambda_1/(\lambda_1 + \lambda_n)$ for the norm. (Note that we may not know $\lambda_1 + \lambda_n$ a priori. A good guess will do though, as any choice of α such that $\alpha < 2/(\lambda_1 + \lambda_n)$ will work.)

Now let's show convergence. If x^* is such that $Ax^* = b$, we can write for any $t \geq 0$

$$\begin{aligned} x^* - x^t &= ((\mathbb{I} - \alpha A)x^* + \alpha b) - ((\mathbb{I} - \alpha A)x^{t-1} + \alpha b) \\ &= (\mathbb{I} - \alpha A)(x^* - x^{t-1}), \end{aligned}$$

so

$$\begin{aligned} \left\|x^* - x^t\right\| &= \left\|(\mathbb{I} - \alpha A)^t(x^* - x^0)\right\| \\ &\leq \left\|\mathbb{I} - \alpha A\right\|^t \left\|x^*\right\| \\ &\leq e^{-\frac{2\lambda_1}{\lambda_1+\lambda_n}t}\left\|x^*\right\|. \end{aligned}$$

Therefore the number of iterations required to get relative error $\|x^* - x^t\|/\|x^*\| \leq \varepsilon$ is at most $(\lambda_1 + \lambda_n)/(2\lambda_1)\ln(1/\varepsilon)$. The important term here is λ_n/λ_1, which is called the *condition number* of A — the smaller the better. It is usually denoted κ and is a measure of how "skewed" the linear transformation implemented by A is.

5.1.2 Interpretation using gradient descent

Here is a completely different way to arrive at exactly the same algorithm. Consider the function

$$f(x) = \frac{1}{2}\big\|Ax - b\big\|^2 = \frac{1}{2}x^T A^T Ax - b^T Ax + \frac{1}{2}\|b\|^2.$$

Our goal is to find an x^* which minimizes f. For convenience, let $b' = A^T b$ and $K = A^T A$, so that $f(x) = x^T Kx/2 - (b')^T x + c$, where $c = \|b\|^2/2$. Let's solve the minimization problem via the gradient method. Fix a step size η and:

- Set x^0 to be an arbitrary vector;
- Iteratively update $x^{t+1} = x^t - \eta \nabla f(x^t) = x^t - \eta \cdot (Kx^t - b')$.

Then it is easy to check by induction that $x^t - x^* = (\mathbb{I} - \eta K)^t (x_0 - x^*)$. The choice of an optimal step size η amounts to selecting η so that $\|\mathbb{I} - \eta K\|$ is minimized. The best choice is $\eta = 2/(\mu_1 + \mu_n)$, where μ_1 and μ_n are the smallest and largest eigenvalues of K respectively. This gives precisely the same algorithm — with $K = A^T A$ instead of A, and $b' = A^T b$ instead of b — as in the previous section!

5.1.3 Speeding up via polynomials

Now here is a really nice trick that gives a quadratic speed-up for the previous algorithm. The vector x^t constructed at the t-th step of the algorithm can be written as

$$x^t = \sum_{i=0}^{t} (\mathbb{I} - \alpha A)^i (\alpha b) = p_t(A) b,$$

where $p_t(A) = \sum_{i=0}^{t} \alpha(\mathbb{I} - \alpha A)^i$. Now if we take $t \to \infty$, what do you recognize?

$$\sum_{i=0}^{\infty} \alpha(\mathbb{I} - \alpha A)^i = \alpha(\mathbb{I} - (\mathbb{I} - \alpha A))^{-1} = \alpha(\alpha A)^{-1} = A^{-1},$$

provided the series converges, which is the case as long as $\|\mathbb{I} - \alpha A\| < 1$. This should be no surprise, since we are solving for $x = A^{-1}b$: what the previous algorithm is doing is simply taking the Taylor expansion of the inverse!

Now, the obvious question is, can we do better, where here better means obtaining an approximation of similar quality but using a polynomial with a lower degree —- this way we'd have to perform fewer iterations, and thus get a faster algorithm.

What do we need exactly? Given a matrix A, we want p_t such that $\|p_t(A)A - \mathbb{I}\| \leq \varepsilon$, i.e. $|p_t(\lambda_i)\lambda_i - 1| \leq \varepsilon$ for $i = 1, \ldots, n$, where λ_i are the eigenvalues of A (assume again that A is symmetric positive definite). If we have this, then

$$\begin{aligned} \|p_t(A)b - x^*\| &= \|p_t(A)Ax^* - x^*\| \\ &= \|(p_t(A)A - \mathbb{I})x^*\| \\ &\leq \|p_t(A)A - \mathbb{I}\|\|x^*\| \\ &\leq \varepsilon\|x^*\|, \end{aligned}$$

as desired. The following theorem gives us what we need.

Theorem 5.1. *For every $t \geq 1$ and $0 < \lambda_1 \leq \lambda_n$, let $\kappa = \lambda_n/\lambda_1$. There is a polynomial $q_t(x)$ of degree t such that $q_t(0) = 1$ and*

$$|q_t(x)| \leq 2\left(1 + \frac{2}{\sqrt{\kappa}}\right)^{-t} \leq 2e^{-\frac{t}{2\sqrt{\kappa}}}, \qquad \forall x \in [\lambda_1, \lambda_n].$$

To see why this is just what we need, note that $q_t(0) = 1$ implies $q_t(x) = 1 - xp_t(x)$ for some p_t of degree $t-1$. Using this p_t, we'll get an approximation with error ε as long as t is at least $\ln(2/\varepsilon)\sqrt{\kappa}/2$: as promised, a quadratic improvement over the previous algorithm.

The proof of Theorem 5.1 uses a construction based on *Chebyshev polynomials*. If you've never seen those they are worth a look — you'll prove the theorem in your homework.

5.1.4 Preconditioning

The two methods for solving $Ax = b$ we've seen have a linear dependence on the sparsity of the matrix A, but a different scaling with respect to the condition number $\kappa = \lambda_{max}\lambda_{min}$: linear in κ for first-order Richardson, $\sqrt{\kappa}$ for the improvement using Chebyshev polynomials. So a natural question is, given an arbitrary matrix A, could we somehow efficiently preprocess A to make its condition number smaller? This would lead to a corresponding improvement in running time for each of the three methods.

This is the idea behind preconditioning. Consider any symmetric invertible $n \times n$ matrix B. Then $Ax = b \iff AB^{-1}(Bx) = b$. Let $y = Bx$, solving $Ax = b$ can be done by solving for y in $AB^{-1}y = b$, and then for x in $Bx = y$. (You might worry that AB^{-1} is not symmetric. We could simply multiply by BA on the left, and solve $BA^2B^{-1}y = BAb$ instead.) Can we find B such that both tasks are easier than the original one? We'd like $Bx = y$ to be easy, and $\kappa(AB^{-1})$ to be much smaller than $\kappa(A)$.

We're going to see a very nice idea on how to do this for the special case where A is a Laplacian matrix. (Recall that Laplacian matrices are not invertible, so when we write inverse we really mean the pseudo-inverse, i.e. the inverse on the orthogonal complement of the nullspace — the $\mathbf{1}$ vector. Sometimes we use the notation L^+ instead of L^{-1} to emphasize this.)

The idea is the following. Suppose G is a given graph, and H a subgraph of G. Then I claim that $L_H \leq L_G$ in the semidefinite order. An easy way to see this is to remember the interpretation of the Laplacian as a quadratic form:

$$x^T L_G x = \sum_{(a,b)\in E(G)} w_{a,b}(x_b - x_a)^2 \geq \sum_{(a,b)\in E(H)} w_{a,b}(x_b - x_a)^2 = x^T L_H x.$$

In particular, we get that $\lambda_{min}(L_G L_H^+) \geq 1$ (the matrix is not symmetric, but we can still talk about its eigenvalues; they are the same as those of $L_H^{+/2} L_G L_H^{+/2}$), so the question is whether we can find H such that, (i) $L_H x = y$ is easy to solve, and (ii) $L_H^+ L_G$ has small norm. The idea to guarantee (i) is to use H that is a *spanning tree* for G. (We need the tree to be spanning so that L_H is invertible on the same space as L_G is.)

Exercise 1. Give a linear time-algorithm to solve $L_H x = y$ (exactly) when H is a tree.

What kind of trees will be such that $\|L_H^+ L_G\|$ is small? We can get a naive bound by computing the trace:

$$\begin{aligned}\mathrm{Tr}\big(L_H^+ L_G\big) &= \sum_{(a,b)\in E(G)} w_{a,b}\mathrm{Tr}\big(L_H^+(e_b-e_a)(e_b-e_a)^T\big)\\ &= \sum_{(a,b)\in E(G)} w_{a,b}(e_b-e_a)^T L_H^+(e_b-e_a).\end{aligned}$$

Now, for fixed (a,b) we've seen this quantity before: it is the effective resistance between a and b, in the tree H. But it's easy to see that, since H is a tree, the effective resistance is just the sum of the resistances along the unique path in H from a to b. Since effective resistances are the inverses of weights, we are led to define the *stretch* of an edge $(a,b) \in E(G)$ with respect to the tree H as

$$s_{a,b} = w_{a,b}\sum_{i=1}^{k}\frac{1}{w_i},$$

where the sum is over the path in H from a to b. So what we are looking for is a *low-stretch spanning tree*...and such things can be computed efficiently!

Theorem 5.2 (Abraham and Neiman '12)**.** *Every weighted graph G has a spanning tree subgraph H such that the sum of the stretches of all edges of G with respect to H is at most $O(m\log n\log\log n)$, where m is the number of edges in G. Moreover, this tree can be computed in time $O(m\log n\log\log n)$.*

The upshot is that using such a tree, we can reduce to the case where $\kappa(L_H^+ L_G) = O(m\log n\log\log n)$. In Claim 11.7 we saw that for an arbitrary (unweighted) graph G the smallest eigenvalue of the (normalized) Laplacian had to be $\Omega(n^{-3})$, and that this bound could be achieved. So a priori the condition number of an arbitrary Laplacian could be as large as $O(n^3)$: here we get an improvement even for the case of dense graphs.

5.2 Laplacian systems

We already saw that the spacial case of Laplacian systems allowed for improvements — using preconditioning to reduce the condition number. Note the interesting interplay between algebraic techniques (after all, we are after solving a system of linear equations) and combinatorial arguments (graph, trees, etc.) that allowed us to make progress.

So let's focus further on Laplacian systems, and give a completely different, quasi-linear time algorithm for solving Laplacian systems. Before this, let's argue that Laplacian systems are not that restricted. In particular, they are no more restrictive than symmetric diagonally dominant (SDD) systems. A real $n\times n$ matrix A is called SDD if the diagonal coefficients of A are all non-negative and satisfy $A_{ii} \geq \sum_{j\neq i}|A_{ij}|$ for all $i \in \{1,\ldots,n\}$.

Consider an SDD matrix A. Then A can be written as $A = P+N$, where P is a Laplacian and N is SDD with non-negative off-diagonal entries. To see this, simply place all negative diagonal entries of A in P, and the positive ones in N; split the diagonal entries so that P is a Laplacian (all row sums are zero), and N is diagonally dominant.

Next we deal with the positive diagonal entries of N. For this the idea is to double the size of the matrix, and consider $A' \in \mathbb{R}^{2n\times 2n}$ such that $A' = P + N'$ where P' is block diagonal with its upper left and lower right blocks both equal to P, and N' is such that $N'_{i,i} = N'_{n+i,n+i} = N_{i,i}$ and $N'_{i,n+j} = N'_{n+j,i} = -P_{i,j}$ for all $i \neq j$, all other entries being zero. Then you can check that P' is a Laplacian, N' is SDD with non-positive off-diagonal entries, and $Ax = b$ if and only if $A'(x, -x) = b' = (b, -b)$.

Finally we need to account for the fact that some row sums may be positive, instead of zero as required of a Laplacian. The idea for this is to introduce an additional vertex, $(2n+1)$, and fake edges. Whenever $A'_{i,i} > \sum_{j\neq i} |A'_{i,j}|$ we add an edge between $(2n+1)$ and i with weight $A'_{i,i} - \sum_{j\neq i} |A'_{i,j}|$. This gives a matrix A'' that is a Laplacian. Moreover, setting $b''_{2n+1} = 0$, the equation $A''x = b''$ has at least one solution $x'' = (x, -x, 0)$; assuming both b and x are orthogonal to the kernel $(1, \ldots, 1)^T$ of A'' the solution is also unique.

Therefore for the remainder of these lectures we let $L = D - A$ be the (un-normalized) Laplacian associated with a weighted, undirected, connected graph G. L has a single 0 eigenvalue with associated eigenvector $(1, \ldots, 1)^T$. We will denote its pseudo-inverse by L^+. Given $b \in \mathbb{R}^n$, our goal is to solve $Lx = b$.

5.2.1 Sparsification

Before we proceed, it will be useful to discuss the topic of sparsification. This can be understood as another "preprocessing" step that reduces the number of edges in the graph while maintaining important combinatorial properties of the graph.

Generally speaking, suppose we are given a graph $G = (V, E)$, and we want to solve some kind of cut or partitioning problem. The running time will typically depend on the number of edges of the graph, which could be $O(n^2)$. The same situation arises in the algorithm we saw last time: the running time for solving $Lx = b$ was brought down to $\tilde{O}(m)$, where m is the sparsity of L. Could we do even better, by "simplifying" the graph via a pre-processing step in order to efficiently produce a graph G' on the same vertex set but with a smaller number of edges and such that G' has roughly the same "cut structure" as G? This is the idea behind sparsification.

Edge sampling

A natural idea is to keep each edge with a certain probability p. Let's do a "back-of-the-envelope" calculation. If a cut $(S, \overline{S})$ involves c edges, and c' is the number of edges across that same partition in G', then on expectation $\mathbf{E}[c'] = pc$. Moreover, by the Chernoff bound,

$$\mathbf{Pr}\left(|pc - c'| \geq \varepsilon pc\right) \leq e^{-\varepsilon^2 pc/2}.$$

If we choose $p = \Omega(d \log n/(\varepsilon^2 c))$ for some d, then the probability drops to n^{-d}. It is possible to show (it is a theorem of Karger) that if the smallest cut in G has size c then the number of cuts of size αc is at most $n^{2\alpha}$. So if we are interested in preserving the size of all small cuts, say cuts of size at most Cn for some large C, we can choose d above large enough as a function of c and perform a union bound. This lets us take $p \approx \log n/(\varepsilon^2 n)$ and so we sparsify a dense graph using only $O(n \log n/\varepsilon^2)$ edges.

Unfortunately this only lets us preserve the size of the few cuts that are relatively small — for slightly larger cuts we'll have no guarantees. It also doesn't work if there are small cuts, as then the concentration is too weak. What is needed is a method to sample edges involved in fewer cuts with higher probability. (Think of the dumbbell graph: we don't want to miss that middle "bridge" edge!)

There is a way to achieve this which involves adding a weight w_e to each edge, such that $w_e \leq c$ where c is the size of the smallest cut in which the edge is involved. We sample edges with probability p/w_e and we assign them a weight of w_e. The expectation for the size of any cut is correct, as before. It is possible to show that this method does better — it provides guarantees that are more uniform across all cuts. But we're going to see how to do even better.

Laplacian approximations

We will show the following theorem:

Theorem 5.3. *For any $\varepsilon > 0$ and graph Laplacian L there exists a Laplacian $\hat{L}$ such that $(1-\varepsilon)L \leq \hat{L} \leq (1+\varepsilon)L$, a condition that we will abbreviate as $L \approx_\varepsilon \hat{L}$. Moreover, $\hat{L}$ has $O(n \log n/\varepsilon^2)$ nonzero entries.*

Not only do we want existence, as in the theorem, but we also want $\hat{L}$ to be efficiently constructible from L. You will see from the proof that this is the case, provided we have an efficient way to compute "effective resistances". This point I will gloss over. In fact it itself reduces to solving a system of linear equations...but an easier one!

Remark 5.4. Recall the interpretation of the Laplacian as a quadratic form, $x^T L x = \sum_{(i,j)\in E}(x_i - x_j)^2$. If $\hat{L} \approx_\varepsilon L$ then for any cut $(S, \overline{S})$, by setting $x_i = 1$ for $i \in S$ and $x_i = 0$ and using

$$(1-\varepsilon)x^T L x \leq x^T \hat{L} x \leq (1+\varepsilon)x^T L x$$

we see that the size of the same cut in $\hat{G}$ will be a close approximation, up to a multiplicative $(1 \pm \varepsilon)$, to its size in G. So the notion of "spectral sparsifier" implicit in Theorem 5.3 is at least as strong as that of "combinatorial sparsifier" (which only requires approximately preserving the size of all cuts); in the homework you will show that it can be strictly stronger (i.e. not every combinatorial sparsifier is a spectral sparsifier).

Let's see that this kind of approximation is also the right kind for our broader goal of solving a Laplacian system. I claim that, for these purposes, it is enough to have $L \approx_{1/2} \hat{L}$. Indeed, once we have such an approximation we can compute an approximate solution x

iteratively by setting $x^{(0)} = 0$ and $x^{(t+1)} = x^{(t)} - \frac{1}{2}\hat{L}^+(Lx^{(t)} - b)$ (note that (**??**) forces $\hat{L}$ and L to have exactly the same nullspace).

Claim 5.5. *We have* $\|x^{(t)} - L^+b\|_L \leq (2/3)^{t-1}\|L^+b\|_L$, *where* $\|u\|_L = u^TLu$.

The weird norm in the claim is just to deal with the nullspace of L being nonzero. As long as b is orthogonal to $(1, \ldots, 1)^T$, we have $LL^+b = b$ and $\|L^+b\|_L = \|b\|$, $\|x^{(t)} - L^+b\|_L = \|Lx^{(t)} - b\|$.

Proof. We prove it by induction on t. The bound is clearly true for $t = 0$. Suppose it true for some t. Then

$$\begin{aligned} Lx^{(t+1)} - b &= (Lx^{(t)} - b) - \frac{1}{2}L\hat{L}^+(Lx^{(t)} - b) \\ &= \left(\mathbb{I} - \frac{1}{2}L\hat{L}^+\right)(Lx^{(t)} - b). \end{aligned}$$

From (**??**) we see that $\Pi/3 \leq L\hat{L}^+/2 \leq \Pi$, where $\Pi = L^+L = \hat{L}^+\hat{L}$; therefore $\|\Pi - L\hat{L}^+/2\| \leq 2/3$. □

We now introduce an additional piece of useful notation. For a symmetric matrix S we write $\overline{S} = L^{+/2}SL^{+/2}$, where L is always the same fixed Laplacian. With this notation, the condition $L \approx_\varepsilon \hat{L}$ is equivalent to

$$\left\|\overline{\hat{L}} - \Pi\right\| \leq \varepsilon, \tag{5.1}$$

which is the condition that we'll focus on from now on.

Sampling via effective resistances

Let's prove the theorem. Our strategy is to use some form of importance sampling, with a well-chosen set of weights. For any edge (a, b), let $r_{a,b} = (e_a - e_b)^TL^+(e_a - e_b)$, where L^+ is the pseudo-inverse of the Laplacian matrix (the inverse of L on its range) and e_a the indicator vector with a 1 in the a-th coordinate and 0 elsewhere. This quantity is called the *effective resistance* of edge (a, b). The reason is as follows: there is a nice interpretation of the Laplacian in terms of currents and voltages. Let v be a vector of voltages at each vertex in the graph. Think of each edge as having a resistance of 1. Then Ohm's law $v(b) - v(a) = r_{ab}(i(b) - i(a))$ lets us compute the currents. Conservation of current at each vertex implies that depending on our choice of voltages, to realize it we will need to inject or extract current at any given vertex. Let i_e be the corresponding vector of *exterior currents*. Then I claim that $i_e = Lv$. This is easy to see from the definition of the Laplacian: for any vertex a, $(Lv)_a = v(a) - \sum_{b\sim a}\frac{1}{d_b}v(v)$ is the default in voltages at a based on what we want to impose, via v, and what Ohm's law gives us.

Inverting this equation, we see that $v = L^+i_e$. Now suppose we force one unit of current from a to b, $i_e = v_b - v_a$. Again by Ohm's law, the effective resistance between a and b will be the difference in potential that is induced by this current, so $r_{ab}^{eff} = (e_b - e_a)^TL^+(e_b - e_a)$.

Back to our sampling procedure. Let's set $p_{a,b} = \min(1, Cr_{a,b} \log n/\varepsilon^2)$, for some constant C that will be defined later. Edge (a, b) is sampled with probability $p_{a,b}$, and it is given a weight $1/p_{a,b}$. This way we get the expectation right: if we let $\hat{L}$ be the random matrix associated to the sampled graph then on expectation $\mathbf{E}[\hat{L}] = L$. Our goal is to show that with high probability $\hat{G}$ has $O(n \log n/\varepsilon^2)$ edges and is an ε-approximation to G, in the sense that $L \approx_\varepsilon \hat{L}$.

Number of edges. Let's start by counting the expected number of edges in $\hat{G}$. We have

$$\begin{aligned}\sum_{(a,b)\in E} r_{a,b} &= \sum_{(a,b)\in E} (e_b - e_a)^T L^+ (e_b - e_a) \\ &= \sum_{(a,b)\in E} \mathrm{Tr}\big(L^+ (e_b - e_a)(e_b - e_a)^T\big) \\ &= \mathrm{Tr}\big(L^+ L\big) \\ &= n - 1,\end{aligned}$$

the dimension of the range of L. The choice of weights $r_{a,b}$ we made is precisely so that this equation works out. Once we scale by $C \log n/\varepsilon^2$ to obtain $p_{a,b}$, we see that the expected number of edges in $\hat{G}$ is $O(n \log n/\varepsilon^2)$, which is quasi-linear in n. Moreover, using that each edge is sampled independently a simple Chernoff bound shows that the probability we sample more than a constant times this number of edges is exponentially small.

Approximation quality. The main remaining question is whether we can show our sampling technique achieves $L \approx_\varepsilon \hat{L}$. Let $X_{a,b}$ be the random matrix defined as $\frac{1}{p_{a,b}} L^{+/2} L_{a,b} L^{+/2}$ with probability $p_{a,b}$, and 0 otherwise, where $L_{a,b} = (e_b - e_a)(e_b - e_a)^T$. If we ignore the "sandwiching" by $L^{+/2}$, $X_{a,b}$ is the Laplacian matrix corresponding to the random graph obtained when we flip the coin that lets us decide whether to include edge (a, b) in $\hat{G}$ or not. We have

$$\mathbf{E}\Big[\sum_{a,b} X_{a,b}\Big] \;=\; L^{+/2}\Big(\sum_{a,b} L_{a,b}\Big) L^{+/2} \;=\; L^{+/2} L L^{+/2} = \Pi,$$

the projector on the range of L. Let $X = \sum_{a,b} X_{a,b}$. We will prove that with high probability $\Pi \approx_\varepsilon X$, i.e. $e^{-\varepsilon}\Pi \leq X \leq e^{\varepsilon}\Pi$. Note that if we have this we can simply multiply on both sides by $L^{1/2}$ and deduce the conclusion we really want, $e^{-\varepsilon} L \leq \hat{L} \leq e^{\varepsilon} L$.

This is starting to look very much like a Chernoff bound...except we are dealing with matrices! As in the Chernoff (or, rather, Hoeffding) bound, the quality of concentration we get will very much depend on the maximal size of the matrices $X_{a,b}$, this time measured by

their operator norm. So let's make sure this is never too big:

$$\begin{aligned}\|X_{a,b}\| &\le \frac{1}{p_{a,b}}\left\|L^{+/2}L_{a,b}L^{+/2}\right\| \\ &= \frac{1}{p_{a,b}}\mathrm{Tr}\big(L^{+}(e_a-e_b)(e_a-e_b)^T\big) \\ &= \frac{r_{a,b}}{p_{a,b}} \\ &= \frac{\varepsilon^2}{C\log n},\end{aligned} \tag{5.2}$$

where we used that $L^{+/2}L_{a,b}L^{+/2}$ is positive semidefinite of rank 1 to equate its norm with its trace. Note how the norm is independent of (a,b). This uniformity is a very good sign that we'll be able to obtain good concentration, and another main advantage of sampling via effective resistances.

Matrix Chernoff bounds. We're almost done — we're only missing the appropriate *matrix Chernoff bound*! Luckily for us, such a thing exists. Just as we saw a matrix Freedman's inequality, here is a matrix analogue of Hoeffding's inequality:

Theorem 5.6. *Let $X_1,\dots,X_m$ be independent random positive semidefinite n-dimensional matrices (not necessarily identically distributed) and $R>0$ such that $\|X_i\|\le R$ for all i. Let $X=\sum X_i$, and μ_{min},μ_{max} the smallest and largest eigenvalues of $\mathbf{E}[X]$ respectively. Then for any $0<\varepsilon<1$,*

$$\mathbf{Pr}\left(\lambda_{min}\Big(\sum_{i=1}^m X_i\Big)\le(1-\varepsilon)\mu_{min}\right)\le n e^{-\frac{\varepsilon^2\mu_{min}}{2R}},$$

$$\mathbf{Pr}\left(\lambda_{max}\Big(\sum_{i=1}^m X_i\Big)\ge(1+\varepsilon)\mu_{max}\right)\le n e^{-\frac{\varepsilon^2\mu_{max}}{3R}}.$$

Note the main difference in the above statement, with respect to the standard Hoeffding bound, is the dimensional factor n on the right-hand side. This is a bit annoying (it would have been nice to have a dimension-free bound!), but in general it is unavoidable.

Exercise 2. Give an example of $X_1,\dots,X_m$ independent and identically distributed random n-dimensional matrices such that $\|X_i\|\le 1$ for all $i\in\{1,\dots,m\}$ and $X=\sum_i X_i$ is such that

$$\mathbf{Pr}\left(\lambda_{max}(X)\ge(1+\varepsilon)\lambda_{max}(\mathbf{E}[X])\right)\ge n e^{-\varepsilon^2\mu_{max}/C},$$

for some $0<\varepsilon<1$ (possibly depending on n) and a constant $C>0$.

Let's apply the theorem to conclude our analysis. In our case, by (5.2) we can set $R=\varepsilon^2/(C\log n)$. Also for us $\mathbf{E}\,X=\Pi$, which has both its smallest and largest eigenvalues equal to 1 (there is the 0 eigenvalue, but it doesn't matter, as we can do all the analysis in

the subspace orthogonal to the associated $\mathbf{1}$ eigenvector; note $\mathbf{1}^T X_{a,b}\mathbf{1} = 0$ for all $(a,b) \in E$). The matrix Chernoff bound gives us

$$\mathbf{Pr}\left(\lambda_{min}(X) \leq (1-\varepsilon)\right) \leq ne^{-\frac{\varepsilon^2 C \log n}{2\varepsilon^2}} = n^{1-C/2},$$

and

$$\mathbf{Pr}\left(\lambda_{max}(X) \geq (1+\varepsilon)\right) \leq ne^{-\frac{\varepsilon^2 C \log n}{3\varepsilon^2}} = n^{1-C/3}.$$

If we choose say $C = 4$, then both probabilities are small, and we get the desired approximation $(1-\varepsilon)\Pi \leq X \leq (1+\varepsilon)\Pi$ with high probability.

5.3 Gaussian elimination

Back to our original problem: we want to solve $Lx = b$! The most standard technique is to proceed by Gaussian elimination. The idea for this is to write the Cholesky factorization of L,

$$L = PTDT^T P, \tag{5.3}$$

where T is lower triangular, D diagonal, and P is a permutation matrix that corresponds to choosing a particular order for the elimination. It is then straightforward to solve for $x = P(T^T)^{-1}D^{-1}T^{-1}Pb$, since each of the inverses can be computed very easily. The total number of operations required is $O(n + nnz(T))$, where $nnz(T)$ denotes the number of non-zero entries of T.

To find a decomposition of the form (5.3) we proceed iteratively, as described in the following procedure:

Gaussian-Elimination(G, L):

(1) Set $L^{(0)} = L$;

(2) Fix an ordering $(v_1, \ldots, v_n)$ of the vertices of G;

(3) For $i = 1, \ldots, n$ do

($\star$) Set $L^{(i)} = L^{(i-1)} - \frac{L^{(i-1)}(:,v_i)L^{(i-1)}(:,v_i)^T}{L^{(i-1)}(v_i,v_i)}$;

(4) Return $D = diag(1/L^{(i-1)}(v_i, v_i))$, $T = (L^{(0)}(:, v_1)|\cdots|L^{(n-1)}(:, v_n))$, and $P = \sum_i e_i^T e_{v_i}$.

Figure 5.1: Gaussian elimination

The procedure has n iterations, and each iteration takes $O(n^2)$ time to update the matrix $L^{(i+1)}$. Therefore Gaussian elimination runs in $O(n^3)$ time.

We can give a combinatorial interpretation of this algorithm. Given a Laplacian L and a vertex v, let $(L)_v = \sum_{e \in E:\, v \in e} w(e) b_e b_e^T$, where $b_e = v - u$ if $e = (u, v)$. $(L)_v$ is the Laplacian associated to the subgraph containing only those edges incident on b. Then we can write the update above as

$$L^{(i)} = L^{(i-1)} - (L^{(i-1)})_{v_i} + (L^{(i-1)})_{v_i} - \frac{L^{(i-1)}(:, v_i) L^{(i-1)}(:, v_i)^T}{L^{(i-1)}(v_i, v_i)},$$

where v_i is the i-th vertex in the chosen elimination order. We would like to argue that each $L^{(i-1)}$ can be interpreted as the Laplacian of a graph $G^{(i-1)}$ on the vertex set $V^{(i-1)} = \{v_i, \ldots, v_n\}$. Clearly for $i = 1$ this is true. Suppose it true for some i. Observe that $L^{(i-1)} - (L^{(i-1)})_{v_i}$ is a Laplacian: it is the Laplacian of the subgraph of $G^{(i-1)}$ obtained by removing vertex v_i and all edges incident on it. The following exercise shows that $(L^{(i-1)})_{v_i} - \frac{L^{(i-1)}(:, v_i) L^{(i-1)}(:, v_i)^T}{L^{(i-1)}(v_i, v_i)}$ is also a Laplacian.

Exercise 3. Show that

$$C_i = (L^{(i-1)})_{v_i} - \frac{L^{(i-1)}(:, v_i) L^{(i-1)}(:, v_i)^T}{L^{(i-1)}(v_i, v_i)} = \sum_{v_j, v_k \sim v_i} \frac{w^{(i-1)}(v_i, v_j) w^{(i-1)}(v_i, v_k)}{w^{(i-1)}(v_i)} b_{(v_j, v_k)} b_{(v_j, v_k)}^T, \tag{5.4}$$

where $w^{(i-1)}$ denote edge weights in $G^{(i-1)}$.

Another way to formulate the formula derived in the exercise is that to go from $G^{(i-1)}$ to $G^{(i)}$ we remove the subgraph incident on v_i, and instead replace it by a clique on the neighbors of v_i in $G^{(i-1)}$, where each pair of neighbors is linked by an edge having weight $\frac{w^{(i-1)}(v_i, v_j) w^{(i-1)}(v_i, v_k)}{w^{(i)}(v_i)}$.

Note that this procedure has one very unfortunate effect. Suppose that the original graph G is sparse, e.g. it is a constant-degree graph. Then the Laplacian L is also sparse: it only has $n + dn$ nonzero entries. Then computing the updates $L^{(i+1)}$ should only require $O(d^2)$ operations, leading to a linear-time algorithm (for constant d). Unfortunately, the update does *not* preserve sparsity: as we just discussed, each update consists in deleting one vertex and adding a (weighted) clique between all its neighbors — so the graph is to quickly going to become denser and denser.

Our goal is to develop an algorithm which avoids such "densification". We will present a very recent quasi-linear time algorithm for this problem due to Kyng and Sachdeva [**?**].

We think of G as a sparse graph, though this need not strictly be the case. It will also be convenient to allow G to have multi-edges, i.e. two vertices can be linked multiple times, with different weights. In terms of the Laplacian, which is the way the grap is represented throughout the algorithm, it means there can be multiple terms proportional to the Laplacian $L_{(u,v)}$ for a single edge. We define L^+ to be the pseudo-inverse of L, and $\Pi = LL^+ = L^+L$ the projector on the range of L.

5.3.1 Sampling cliques

Based on the intuition we got from analyzing the Gaussian elimination procedure as a vertex elimination procedure where a star on a vertex is replaced by a clique on its neighbors, our main idea is to introduce a randomized algorithm which, instead of introducing a complete clique on all neighbors of the eliminated vertex v_i, only considers a sub-clique, with Laplacian $Y_i = \sum_e Y_{i,e}$, obtained by sub-sampling a small fraction of the edges.

Before proceeding, we need to set a bit of notation. Let $b_{u,v}$ denote the Laplacian associated with a single (unweighted) edge $e = (u, v)$. The algorithm will maintain explicit representations of the Laplacian it manipulates through a multi-edge decomposition as $L = \sum_{e=(u,v)} w(e) b_{u,v} b_{u,v}^T$, where $w(e)$ is the weight of an edge e. We allow multi-edges, meaning we can have $e \neq e'$ with the same endpoints (u, v) but different (or even the same) weights $w(e)$, $w(e')$. Given a sub-graph, also represented via its Laplacian S, and a vertex v we let $(S)_v$ denote the sub-Laplacian of S containing all edges of S incident on v. We write $w_S(v)$ for the sum of weights of all edges in S that are incident on v, and $d_S(v)$ for the degree of v in S (the number of edges of S incident on v, counted with multiplicity).

Consider the following procedure:

CLIQUE-SAMPLE(S,v): S a Laplacian specified by an explicit decomposition as a sum of weighted multi-edge Laplacian; v a vertex.

(1) For j from 1 to $d_S(v)$ do

- Sample $e = (u, v)$ incident on v with probability $w(e)/w_S(v)$;
- Sample $e' = (u', v)$ incident on v with probability $1/d_S(v)$;
- If $u \neq u'$ set $Y_j = \frac{w(e)w(e')}{w(e)+w(e')} b_{(u,u')} b_{(u,u')}^T$; else set $Y_j = 0$.

(2) Return $Y = \sum_j Y_j$.

Figure 5.2: Sampling cliques

The following claim shows that the clique sampling procedure works, on expectation.

Claim 5.7. *Let S be a Laplacian input to* CLIQUE-SAMPLE *and v a vertex. Then the output Y of* CLIQUE-SAMPLE*(S,v) satisfies the following properties:*

(1) $Y = \sum_j Y_j$ where each Y_j is either 0 or the Laplacian of a multi-edge whose endpoints are neighbors of v in S;

(2) $\mathbf{E}[Y] = C(S, v)$, where $C(S, v) = (S)_v - \frac{S(:,v)S(:,v)^T}{S(v,v)}$;

Proof. The first property is clear. For the second property, we can write

$$\begin{aligned}\mathbf{E}\left[\sum_j Y_j\right] &= d_S(v) \sum_{e=(u,v)} \sum_{e'=(u',v),\, u' \neq u} \frac{w(e)}{w_S(v)} \frac{1}{d_S(v)} \frac{w(e)w(e')}{w(e)+w(e')} b_{(u,u')} b_{(u,u')}^T \\ &= C(S,v),\end{aligned}$$

where to get the second equality, observe that each pair of edges (e, e') appears exactly twice in the sum, with different weights; then use Exercise 1 from the previous lecture. □

Claim 5.7 is already good enough to argue that "in expectation" the following modified Gaussian elimination algorithm works:

MODIFIED-GAUSSIAN-ELIMINATION(G, L): G a weighted graph and L the Laplacian of G, specified as a sum of weighted edge Laplacians (with repetition in case G has multi-edges).

(1) Set $\hat{S}^{(0)} = L$;

(2) Fix an ordering $(v_1, \ldots, v_n)$ of the vertices of G;

(3) For $i = 1, \ldots, n$ do

($\star'$) Let $\hat{C}_i = \text{CLIQUE-SAMPLE}(\hat{S}^{(i-1)}, v_i)$;
Set $\hat{S}^{(i)} = \hat{S}^{(i-1)} - (\hat{S}^{(i-1)})_{v_i} + \hat{C}_i$.

(4) Return $\hat{D} = \text{diag}(1/\hat{S}^{(i-1)}(v_i, v_i))$, $\hat{T} = (\hat{S}^{(0)}(:, v_1) | \cdots | \hat{S}^{(n-1)}(:, v_n))$, and $\hat{P} = \sum_i e_{v_i}^T e_i$.

Figure 5.3: Modified Gaussian elimination

Due to property (2) in Claim 5.7 we see that in expectation, the rule ($\star'$) performs exactly the same update as ($\star$): if $(\hat{D}, \hat{T})$ is the output of the algorithm, and $\hat{P}$ the chosen ordering of vertices, then

$$\mathbf{E}\left[\hat{P}\hat{T}\hat{D}\hat{T}^T\hat{P}^T\right] = L, \tag{5.5}$$

as desired.

We can also bound the running time of the algorithm. The subroutine CLIQUE-SAMPLE$(\hat{S}, v)$ can be executed in time $O(\deg_S(v))$ (For this one must show how to implement the neighbor-sampling steps in constant time on average — this is possible by performing a pre-processing step to store all neighbors of v in an appropriate data structure). By definition, procedure ($\star'$) preserves the total number of multi-edges in the graph. Since we choose the next vertex to eliminate at random at each step, its expected degree is the average degree in the initial graph, scaled by n/i at the i-th iteration. Summing over all $i = 1, \ldots, n$ we obtain a running time of $O(m \log n)$.

5.3.2 A Martingale

Of course, correctness in expectation is, by itself, useless. What we really need, as we saw in (5.1), is a bound on the norm of $\overline{\hat{L}} - \Pi$. To get started, let's find a way to express $\overline{\hat{L}} - \Pi$ as a sum of random variables that correspond to the clique samples made by the algorithm. Towards this, introduce a sequence of random variables

$$\hat{L}^{(k)} = \hat{S}^{(k)} + \sum_{i=1}^{k} \frac{\hat{S}^{(i-1)}(:, v_i)\hat{S}^{(i-1)}(:, v_i)^T}{\hat{S}^{(i-1)}(v_i, v_i)}, \tag{5.6}$$

for $k = 0, \ldots, n$. Intuitively, $\hat{L}^{(k)}$ represents the Laplacian of an "intermediate graph" at the k-th iteration of the algorithm: for vertices $v_1, \ldots, v_k$, $\hat{L}^{(k)}$ is like the final row-reduced Laplacian, whereas for vertices $v_{k+1}, \ldots, v_n$, $\hat{L}^{(k)}$ is the graph that results from the original graph, with all the additional cliques added in. In particular, $\hat{L}^{(0)} = L$ and $\hat{L}^{(n)} = \hat{L}$. Using the definition,

$$\begin{aligned}
\hat{L}^{(k)} - \hat{L}^{(k-1)} &= \hat{S}^{(k)} - \hat{S}^{(k-1)} + \frac{\hat{S}^{(k-1)}(:, v_k)\hat{S}^{(k-1)}(:, v_k)^T}{\hat{S}^{(k-1)}(v_k, v_k)} \\
&= \hat{C}_k - (\hat{S}^{(k-1)})_{v_k} + \frac{\hat{S}^{(k-1)}(:, v_k)\hat{S}^{(k-1)}(:, v_k)^T}{\hat{S}^{(k-1)}(v_k, v_k)} \\
&= \hat{C}_k - \mathbf{E}[\hat{C}_k | \mathcal{F}_{k-1}],
\end{aligned}$$

where the last equality follows from (2) in Claim 5.7, and we introduced $\mathcal{F}_{k-1}$ to denote all the random choices made by the algorithm up to the start of the k-th iteration. Now $\hat{C}_k$ itself decomposes as a sum $\hat{C}_k = \sum_e Y_{k,e}$, and so using a telescoping sum we can write

$$\begin{aligned}
\hat{L} - L &= \hat{L}^{(n)} - \hat{L}^{(0)} \\
&= \sum_{k=1}^{n} \left(\hat{L}^{(k)} - \hat{L}^{(k-1)} \right) \\
&= \sum_{k=1}^{n} \sum_{e=1}^{d_k} \left(Y_{k,e} - \mathbf{E}[Y_{k,e} | \mathcal{F}_{k-1}] \right),
\end{aligned}$$

where $d_k = \deg_{\hat{S}^{(k-1)}}(v_k)$. Multiplying left and right by $L^{+/2}$, we have succeeded in expressing the difference between the graph Laplacian L and the estimate $\hat{L}$ returned by our algorithm as a sum of mean-zero random variables. Let

$$Z = L^{+/2}(\hat{L} - L)L^{+/2} = \sum_{k=1}^{n} \sum_{e=1}^{d_k} \overline{X_{k,e}}, \tag{5.7}$$

where $\overline{X_{k,e}} = \overline{Y_{k,e}} - \mathbf{E}[\overline{Y_{k,e}} | \mathcal{F}_{k-1}]$. Our goal (5.1) is then to bound the probability that Z does not satisfy the inequalities $-\varepsilon\Pi \leq Z \leq \varepsilon\Pi$, for some $\varepsilon \leq 1/2$; since Z has the same kernel as L (which is the same as $\hat{L}$), this is equivalent to $-\varepsilon\mathbb{I} \leq Z \leq \varepsilon\mathbb{I}$.

5.3.3 Matrix concentration inequalities

The decomposition (5.7) expresses Z as a sum of mean-zero matrix random variables. These random variables are not independent, but they form a martingale difference sequence (MDS): for each $1 \leq k \leq n$ and $1 \leq e \leq d_k$ we gave $\mathbf{E}[\overline{X_{k,e}}|\mathcal{F}_{k-1}] = 0$ (note that here it is sufficient to condition on all choices up to the $(k-1)$-st iteration, as all the edge samples e made by Clique-Sample are taken independently from each other). Since we have an MDS, we are in a good shape to apply the following concentration theorem, which is a matrix-valued variant of Freedman's inequality due to Joel Tropp:

Theorem 5.8. *Let $\{X_i = Y_i - Y_{i-1}\}_{i=1,\ldots,m}$ be a martingale difference sequence, where Y_i is a symmetric $n \times n$ matrix. Suppose that $\|X_i\| \leq R$ for all $i \in \{1,\ldots,m\}$. Let $W_k = \sum_{i=1}^{k} \mathbf{E}[X_i^2|Y_{i-1}]$. Then for all $t \geq 0$ and $\sigma^2 > 0$,*

$$\mathbf{Pr}\left(\exists k : \|Y_k\| \geq t \text{ and } \|W_k\| \leq \sigma^2\right) \leq n e^{-\frac{t^2/2}{\sigma^2 + Rt/3}}.$$

Note that if the X_i are i.i.d. then W_k is just a number. In this case we can set $\sigma^2 = \sum_i X_i^2$ in Theorem 5.8, and recover (a matrix variant of) Bernstein's inequality. In terms of the quality of the bounds, the only difference between the matrix Freedman's and the scalar Freedman's inequality is the dimension dependence, which shows up via the prefactor of n in front of the exponential.

5.3.4 Variance analysis

To apply Theorem 5.8 we need to control two things: first, the operator norm of each edge sample taken by Clique-Sample, and second, the "variance" $\|W_k\|$.

To bound the norm of the Y_i (note that the relevant quantity for the application of Theorem 5.8 is really the norm of $\overline{Y_i}$) we need to make some assumption on the weights that are considered by the algorithm. Assume for simplicity that the initial Laplacian L is the Laplacian of an unweighted graph, so that all weights $w(e) = 1$.

Let $\rho > 0$ be a parameter to be decided on later. Say that a Laplacian S is ρ-bounded with respect to L if $\|w(e)\overline{b_{(u,v)}}\| \leq 1/\rho$ for every edge $e = (u,v)$ in S. We can assume that the initial Laplacian L is ρ-bounded with respect to itself: this means that all edge weights should be at most $1/\rho$, which can be achieved by splitting each "regular" edge into ρ multi-edges of weight $1/\rho$ each. The following claim shows that the Clique-Sample procedure does not increase this measure of weight throughout the algorithm:

Claim 5.9. *Let S be a Laplacian input to Clique-Sample, L a Laplacian, and assume that S is $1/\rho$-bounded with respect to L. Then the output Y of Clique-Sample(S,v) is $1/\rho$-bounded with respect to L, i.e. $\|\overline{Y}_j\| \leq 1/\rho$ for every j.*

Proof. To prove the claim we use the following:

Exercise 4. Show the following "triangle inequality": for any three distinct vertices v, u, u',

$$\left\|\overline{b_{(u,u')}b_{(u,u')}^T}\right\| \leq \left\|\overline{b_{(v,u)}b_{(v,u)}^T}\right\| + \left\|\overline{b_{(v,u')}b_{(v,u')}^T}\right\|. \tag{5.8}$$

[Hint: quadratic forms...]

Using (5.8),

$$\begin{aligned}\frac{w(e)w(e')}{w(e)+w(e')}\|\overline{b_{(u,u')}b_{(u,u')}^T}\| &\leq \frac{w(e)w(e')}{w(e)+w(e')}\Big(\|\overline{b_{(v,u)}b_{(v,u)}^T}\| + \|\overline{b_{(v,u')}b_{(v,u')}^T}\|\Big)\\ &\leq \frac{w(e)w(e')}{w(e)+w(e')}\Big(\frac{1}{\rho w(e)}+\frac{1}{\rho w(e')}\Big)\\ &= \frac{1}{\rho},\end{aligned}$$

where in the second line we used the assumption that S itself is ρ-bounded with respect to L. □

Claim 5.9 readily gives us a good choice for R in our application of theorem 5.8: we can take $R = 1/\rho$. Now it remains to study the variance term $W_k = \sum_{i=1}^{k}\sum_{e=1}^{d_i}\mathbf{E}[\overline{X_{i,e}}^2|\mathcal{F}_{i-1}]$ (we can always assume that all edges for a given run of Clique-Sample are taken in together). Fix an index k, and observe that

$$\mathbf{E}\Big[\sum_{e=1}^{d_k}\overline{X_{k,e}}^2\,\Big|\,\mathcal{F}_{k-1}\Big] \leq \frac{1}{\rho}\,\overline{C(\hat{S}^{(k-1)},v_k)},$$

where we used $\overline{X_{k,e}}^2 \leq \frac{1}{\rho}\overline{X_{k,e}}$, which follows from Claim 5.9, and the definition of $C(\cdot,\cdot)$. Note that $\overline{C(\hat{S}^{(k-1)},v_k)}$ is itself a random variable, which depends on all choices made by the algorithms at iterations prior to the k-th. What we do know from the definition is that $C(\hat{S}^{(k-1)},v_k) \leq (\hat{S}^{(k-1)})_{v_k}$, where v_k is chosen uniformly at random among $(n-k)$ possible vertices. Let's make an assumption that the Laplacian $\hat{S}^{(k-1)}$ considered throughout the procedure remain "reasonably bounded" compared to the original Laplacian L, i.e.

$$\hat{S}^{(k-1)} \leq D\,L \tag{5.9}$$

for some constant D. This is not unreasonable, as from the definition of $\hat{L}^{(k-1)}$ we see that $\hat{S}^{(k-1)} \leq \hat{L}^{(k-1)}$, so we can hope that the approximations $\hat{L}^{(k-1)}$ we defined do not deviate too much from L.

Assuming (5.9), since $\hat{S}^{(k-1)} = \frac{1}{2}\sum_{i=k}^{n}(\hat{S}^{(k-1)})_{v_i}$ (the 1/2 is due to the fact that each edge gets counted twice, for each of its endpoints) we get that on average over the choice of vertex v_k (all other prior choices of the algorithm remaining fixed),

$$\mathop{\mathbf{E}}_{\pi}\Big[\overline{C(\hat{S}^{(k-1)},v_k)}\Big] \leq \frac{D}{2(n-k)}\mathbb{I}. \tag{5.10}$$

There is one remaining subtlety that prevents us from using this bound directly for the value of σ^2 in Theorem 5.8. The reason is that the choice of vertex v_k must be made when we make the call to Clique-Sample. So we can consider v_k to be uniformly random at the

time when we sample the first edge $Y_{k,e}$. For subsequent edges however, v_k has been fixed, so that the above averaging argument for bounding $\overline{C(\hat{S}^{(k-1)}, v_k)}$ no longer applies.

This difficulty can be overcome by introducing a second martingale, with random variables the $C(\hat{S}^{(k-1)}, v_k)$ themselves. This martingale does not have expectation zero, but we can bound its terms by another application of Freedman's inequality, yielding a good bound on $C(\hat{S}^{(k-1)}, v_k)$ that we can apply not only at the time of the first clique sample, but for subsequent ones as well.

Ignoring the difficulty, by summing (5.10) over $k = 1, \dots, n$, we get that by choosing $\sigma^2 = O(\log n/\rho)$ in Theorem 5.8 the bound $\|W_k\| \leq \sigma^2$ holds always (in fact, it'll only hold with high probability, due to the above-mentioned issue). With $t = \varepsilon$ and $R = 1/\rho$ the bound from the theorem becomes $ne^{-O(\varepsilon\rho/\log n)}$. With a constant value of ε, say $\varepsilon = 1/10$, it is enough to choose $\rho = O(\log^2 n)$ to make the bound inverse polynomial. Overall, this gives a running time $O(m\rho \log n) = O(m \log^3 n)$ for the algorithm: quasilinear in the input size, as desired.

Remark 5.10. A second issue we glossed over is the assumption (5.9). In fact it is possible to reduce the proof to an even stronger condition, viz.

$$(1 - \varepsilon)L \leq \hat{L}^{(k-1)} \leq (1 + \varepsilon)L \tag{5.11}$$

for each $k = 1, \dots, n$. The idea here is to proceed in two steps. Define an event E_k that corresponds to (5.11), and condition the martingale on E_k being true. Then the above proof carries through, and we can bound the truncated martingale. Finally, it is possible to show that the truncated martingale has *more* chances of not satisfying the conclusion that $-\varepsilon\mathbb{I} \leq Z \leq \varepsilon\mathbb{I}$, therefore the bound on the truncated martingale applies to the untruncated one as well, with no loss. We refer to [**?**] for more on this.