Spectral Graph Theory Lecture 24

Preconditioning Laplacians

Daniel A. Spielman November 28, 2018

A *preconditioner* for a positive semidefinite matrix $\boldsymbol{A}$ is a positive semidefinite matrix $\boldsymbol{B}$ such that it is easy to solve systems of linear equations in $\boldsymbol{B}$ and the condition number of $\boldsymbol{B}^{-1}\boldsymbol{A}$ is small. A good preconditioner allows one to quickly solve systems of equations in $\boldsymbol{A}$.

In this lecture, we will measure the quality of preconditioners in terms of the ratio

$$\kappa(\boldsymbol{A}, \boldsymbol{B}) \overset{\text{def}}{=} \beta/\alpha,$$

where α is the largest number and β is the smallest such that

$$\alpha\boldsymbol{B} \preccurlyeq \boldsymbol{A} \preccurlyeq \beta\boldsymbol{B}.$$

Lemma 24.0.1. *Let α and β be as defined above. Then, α and β are the smallest and largest eigenvalues of $\boldsymbol{B}^{-1}\boldsymbol{A}$, excluding possible zero eigenvalues corresponding to a common nullspace of $\boldsymbol{A}$ and $\boldsymbol{B}$.*

We need to exclude the common nullspace when $\boldsymbol{A}$ and $\boldsymbol{B}$ are the Laplacian matrices of connected graphs. If these matrices have different nullspaces $\alpha = 0$ or $\beta = \infty$ and the condition number β/α is infinite.

Proof of Lemma 24.0.1. We just prove the statement for β, in the case where neither matrix is singular. We have

$$\begin{aligned}\lambda_{max}(\boldsymbol{B}^{-1}\boldsymbol{A}) &= \lambda_{max}(\boldsymbol{B}^{-1/2}\boldsymbol{A}\boldsymbol{B}^{-1/2}) \\ &= \max_{\boldsymbol{x}} \frac{\boldsymbol{x}^T\boldsymbol{B}^{-1/2}\boldsymbol{A}\boldsymbol{B}^{-1/2}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} \\ &= \max_{\boldsymbol{y}} \frac{\boldsymbol{y}^T\boldsymbol{A}\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{B}\boldsymbol{y}}, \qquad \text{settting } \boldsymbol{y} = B^{-1/2}\boldsymbol{x},\end{aligned}$$

which equals β. □

Recall that the eigenvalues of $\boldsymbol{B}^{-1}\boldsymbol{A}$ are the same as those of $\boldsymbol{B}^{-1/2}\boldsymbol{A}\boldsymbol{B}^{-1/2}$ and $\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}$.

24.1 Approximate Solutions

Recall the $\boldsymbol{A}$-norm:

$$\|\boldsymbol{x}\|_{\boldsymbol{A}} = \sqrt{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}} = \left\|\boldsymbol{A}^{1/2}\boldsymbol{x}\right\|.$$

We say that $\widetilde{\boldsymbol{x}}$ is an ϵ-approximate solution to the problem $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ if

$$\|\widetilde{\boldsymbol{x}} - \boldsymbol{x}\|_{\boldsymbol{A}} \leq \epsilon \|\boldsymbol{x}\|_{\boldsymbol{A}}.$$

24.2 Iterative Refinement

We will now see how to use a *very* good preconditioner to solve a system of equations. Let's consider a preconditioner $\boldsymbol{B}$ that satisfies

$$(1-\epsilon)\boldsymbol{B} \preccurlyeq \boldsymbol{A} \preccurlyeq (1+\epsilon)\boldsymbol{B}.$$

So, all of the eigenvalues of

$$\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2} - \boldsymbol{I}$$

have absolute value at most ϵ.

The vector $\boldsymbol{B}^{-1}\boldsymbol{b}$ is a good approximation of $\boldsymbol{x}$ in the $\boldsymbol{A}$-norm. We have

$$\begin{aligned}
\left\|\boldsymbol{B}^{-1}\boldsymbol{b} - \boldsymbol{x}\right\|_{\boldsymbol{A}} &= \left\|\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{b} - \boldsymbol{A}^{1/2}\boldsymbol{x}\right\| \\
&= \left\|\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^{1/2}\boldsymbol{x}\right\| \\
&= \left\|\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}(\boldsymbol{A}^{1/2}\boldsymbol{x}) - \boldsymbol{A}^{1/2}\boldsymbol{x}\right\| \\
&\leq \left\|\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2} - \boldsymbol{I}\right\| \left\|\boldsymbol{A}^{1/2}\boldsymbol{x}\right\| \\
&\leq \epsilon \left\|\boldsymbol{A}^{1/2}\boldsymbol{x}\right\| \\
&= \epsilon \left\|\boldsymbol{x}\right\|_{\boldsymbol{A}}.
\end{aligned}$$

Remark: This result crucially depends upon the use of the $\boldsymbol{A}$-norm. It fails under the Euclidean norm.

If we want a better solution, we can just compute the residual and solve the problem in the residual. That is, we set

$$\boldsymbol{x}_1 = \boldsymbol{B}^{-1}\boldsymbol{b},$$

and compute

$$\boldsymbol{r}_1 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_1 = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_1).$$

We then use one solve in $\boldsymbol{B}$ to compute a vector $\boldsymbol{x}_2$ such that

$$\left\|(\boldsymbol{x} - \boldsymbol{x}_1) - \boldsymbol{x}_2\right\|_{\boldsymbol{A}} \leq \epsilon \left\|\boldsymbol{x} - \boldsymbol{x}_1\right\|_{\boldsymbol{A}} \leq \epsilon^2 \left\|\boldsymbol{x}\right\|_{\boldsymbol{A}}.$$

So, $\boldsymbol{x}_1 + \boldsymbol{x}_2$, our new estimate of $\boldsymbol{x}$, differs from $\boldsymbol{x}$ by at most an ϵ^2 factor. Continuing in this way, we can find an ϵ^k approximation of $\boldsymbol{x}$ after solving k linear systems in $\boldsymbol{B}$. This procedure is called *iterative refinement.*

24.3 Iterative Methods in the Matrix Norm

The iterative methods we studied last class can also be shown to produce good approximate solutions in the matrix norm. Given a matrix $\boldsymbol{A}$, these produce ϵ-approximation solutions after t

iterations if there is a polynomial q of degree t for which $q(0) = 1$ and $|q(\lambda_i)| \leq \epsilon$ for all eigenvalues of $\boldsymbol{A}$. To see this, recall that we can define $p(x)$ so that $q(x) = 1 - xp(x)$, and set

$$\widetilde{\boldsymbol{x}} = p(\boldsymbol{A})\boldsymbol{b},$$

to get

$$\left\| \widetilde{\boldsymbol{x}} - \boldsymbol{x} \right\|_{\boldsymbol{A}} = \left\| p(\boldsymbol{A})\boldsymbol{b} - \boldsymbol{x} \right\|_{\boldsymbol{A}} = \left\| p(\boldsymbol{A})\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x} \right\|_{\boldsymbol{A}}.$$

As $\boldsymbol{I}$, $\boldsymbol{A}$, $p(\boldsymbol{A})$ and $\boldsymbol{A}^{1/2}$ all commute, this equals

$$\begin{aligned} \left\| \boldsymbol{A}^{1/2} p(\boldsymbol{A})\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^{1/2}\boldsymbol{x} \right\| &= \left\| p(\boldsymbol{A})\boldsymbol{A}\boldsymbol{A}^{1/2}\boldsymbol{x} - \boldsymbol{A}^{1/2}\boldsymbol{x} \right\| \\ &\leq \left\| p(\boldsymbol{A})\boldsymbol{A} - \boldsymbol{I} \right\| \left\| \boldsymbol{A}^{1/2}\boldsymbol{x} \right\| \\ &\leq \epsilon \left\| \boldsymbol{x} \right\|_{\boldsymbol{A}}. \end{aligned}$$

24.4 Preconditioned Iterative Methods

Preconditioned iterative methods can be viewed as the extension of Iterative Refinement by algorithms like Chebyshev iteration and the Preconditioned Conjugate Gradient. These usually work with condition numbers much larger than 2.

In each iteration of a preconditioned method we will solve a system of equations in $\boldsymbol{B}$, multiply a vector by $\boldsymbol{A}$, and perform a constant number of other vector operations. For this to be worthwhile, the cost of solving equations in $\boldsymbol{B}$ has to be low.

We begin by seeing how the analysis with polynomials translates. Let λ_i be the ith eigenvalue of $\boldsymbol{B}^{-1}\boldsymbol{A}$. If $q_t(x) = 1 - xp_t(x)$ is a polynomial such that $|q_t(\lambda_i)| \leq \epsilon$ for all i, then

$$\boldsymbol{x}_t \overset{\text{def}}{=} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{b}$$

will be an ϵ-approximate solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$:

$$\begin{aligned} \left\| \boldsymbol{x} - \boldsymbol{x}_t \right\|_{\boldsymbol{A}} &= \left\| \boldsymbol{A}^{1/2}\boldsymbol{x} - \boldsymbol{A}^{1/2}\boldsymbol{x}_t \right\| \\ &= \left\| \boldsymbol{A}^{1/2}\boldsymbol{x} - \boldsymbol{A}^{1/2} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{b} \right\| \\ &= \left\| \boldsymbol{A}^{1/2}\boldsymbol{x} - \boldsymbol{A}^{1/2} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} \right\| \\ &= \left\| \boldsymbol{A}^{1/2}\boldsymbol{x} - \boldsymbol{A}^{1/2} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}(\boldsymbol{A}^{1/2}\boldsymbol{x}) \right\| \\ &\leq \left\| \boldsymbol{I} - \boldsymbol{A}^{1/2} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2} \right\| \left\| (\boldsymbol{A}^{1/2}\boldsymbol{x}) \right\|. \end{aligned}$$

We now prod this matrix into a more useful form:

$$\boldsymbol{I} - \boldsymbol{A}^{1/2} p_t(\boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2} = \boldsymbol{I} - p_t(\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2})\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2} = q_t(\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}).$$

So, we find

$$\left\| \boldsymbol{x} - \boldsymbol{x}_t \right\|_{\boldsymbol{A}} \leq \left\| q_t(\boldsymbol{A}^{1/2}\boldsymbol{B}^{-1}\boldsymbol{A}^{1/2}) \right\| \left\| (\boldsymbol{A}^{1/2}\boldsymbol{x}) \right\| \leq \epsilon \left\| \boldsymbol{x} \right\|_{\boldsymbol{A}}.$$

The Preconditioned Conjugate Gradient (PCG) is a magical algorithm that after t steps (each of which involves solving a system in $\boldsymbol{B}$, multiplying a vector by $\boldsymbol{A}$, and performing a constant number of vector operations) produces the vector $\boldsymbol{x}_t$ that *minimizes*

$$\|\boldsymbol{x}_t - \boldsymbol{x}\|_{\boldsymbol{A}}$$

over all vectors $\boldsymbol{x}_t$ that can be written in the form $p_t(\boldsymbol{b})$ for a polynomial of degree at most t. That is, the algorithm finds the best possible solution among all iterative methods of the form we have described. We first bound the quality of PCG by saying that it is at least as good as Preconditioned Chebyshev, but it has the advantage of not needing to know α and β. We will then find an improved analysis.

24.5 Preconditioning by Trees

Vaidya [Vai90] had the remarkable idea of preconditioning the Laplacian matrix of a graph by the Laplacian matrix of a subgraph. If H is a subgraph of G, then

$$\boldsymbol{L}_H \preccurlyeq \boldsymbol{L}_G,$$

so all eigenvalues of $\boldsymbol{L}_H^{-1}\boldsymbol{L}_G$ are at least 1. Thus, we only need to find a subgraph H such that $\boldsymbol{L}_H$ is easy to invert and such that the largest eigenvalue of $\boldsymbol{L}_H^{-1}\boldsymbol{L}_G$ is not too big.

It is relatively easy to show that linear equations in the Laplacian matrices of trees can be solved exactly in linear time. One can either do this by finding an LU-factorization with a linear number of non-zeros, or by viewing the process of solving the linear equation as a dynamic program that passes up once from the leaves of the tree to a root, and then back down.

We will now show that a special type of tree, called a *low-stretch spanning tree* provides a very good preconditioner. To begin, let T be a spanning tree of G. Write

$$\boldsymbol{L}_G = \sum_{(u,v)\in E} w_{u,v}\boldsymbol{L}_{u,v} = \sum_{(u,v)\in E} w_{u,v}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T.$$

We will actually consider the trace of $\boldsymbol{L}_T^{-1}\boldsymbol{L}_G$. As the trace is linear, we have

$$\begin{aligned}\operatorname{Tr}\left(\boldsymbol{L}_T^{-1}\boldsymbol{L}_G\right) &= \sum_{(u,v)\in E} w_{u,v}\operatorname{Tr}\left(\boldsymbol{L}_T^{-1}\boldsymbol{L}_{u,v}\right)\\ &= \sum_{(u,v)\in E} w_{u,v}\operatorname{Tr}\left(\boldsymbol{L}_T^{-1}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T\right)\\ &= \sum_{(u,v)\in E} w_{u,v}\operatorname{Tr}\left((\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T\boldsymbol{L}_T^{-1}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)\right)\\ &= \sum_{(u,v)\in E} w_{u,v}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T\boldsymbol{L}_T^{-1}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v).\end{aligned}$$

To evaluate this last term, we need to know the value of $(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T\boldsymbol{L}_T^{-1}(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)$. You already know something about it: it is the effective resistance in T between u and v. In a tree, this equals the distance in T between u and v, when we view the length of an edge as the reciprocal of its weight. This is because it is the resistance of a path of resistors in series. Let $T(u, v)$ denote the path in T from u to v, and let $w_1, \ldots, w_k$ denote the weights of the edges on this path. As we view the weight of an edge as the reciprocal of its length,

$$(\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T \boldsymbol{L}_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v) = \sum_{i=1}^{k} \frac{1}{w_i}. \tag{24.1}$$

Even better, the term (24.1) is something that has been well-studied. It was defined by Alon, Karp, Peleg and West [AKPW95] to be the *stretch* of the unweighted edge (u, v) with respect to the tree T. Moreover, the *stretch* of the edge (u, v) with weight $w_{u,v}$ with respect to the tree T is defined to be exactly

$$w_{u,v} \sum_{i=1}^{k} \frac{1}{w_i},$$

where again $w_1, \ldots, w_k$ are the weights on the edges of the unique path in T from u to v. A sequence of works, begining with [AKPW95], has shown that every graph G has a spanning tree in which the sum of the stretches of the edges is low. The best result so far is due to [AN12], who prove the following theorem.

Theorem 24.5.1. *Every weighted graph G has a spanning tree subgraph T such that the sum of the stretches of all edges of G with respect to T is at most*

$$O(m \log n \log \log n),$$

where m is the number of edges G. Moreover, one can compute this tree in time $O(m \log n \log \log n)$.

Thus, if we choose a low-stretch spanning tree T, we will ensure that

$$\operatorname{Tr}\left(\boldsymbol{L}_T^{-1} \boldsymbol{L}_G\right) = \sum_{(u,v) \in E} w_{u,v} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v)^T \boldsymbol{L}_T^{-1} (\boldsymbol{\chi}_u - \boldsymbol{\chi}_v) \leq O(m \log n \log \log n).$$

In particular, this tells us that $\lambda_{max}(L_T^{-1} \boldsymbol{L}_G)$ is at most $O(m \log n \log \log n)$, and so the Preconditioned Conjugate Gradient will require at most $O(m^{1/2} \log n)$ iterations, each of which requires one multiplication by $\boldsymbol{L}_G$ and one linear solve in $\boldsymbol{L}_T$. This gives an algorithm that runs in time $O(m^{3/2} \log n \log 1/\epsilon)$, which is much lower than the $O(n^3)$ of Gaussian elimination when m, the number of edges in G, is small.

This result is due to Boman and Hendrickson [BH01].

24.6 Improving the Bound on the Running Time

We can show that the Preconditioned Conjugate Gradient will actually run in closer to $O(m^{1/3})$ iterations. Since the trace is the sum of the eigenvalues, we know that for every $\beta > 0$, $\boldsymbol{L}_T^{-1} \boldsymbol{L}_G$ has at most

$$\operatorname{Tr}\left(\boldsymbol{L}_T^{-1} \boldsymbol{L}_G\right) / \beta$$

eigenvalues that are larger than β.

To exploit this fact, we use the following lemma. It basically says that we can ignore the largest eigenvalues of $\boldsymbol{B}^{-1}\boldsymbol{A}$ if we are willing to spend one iteration for each.

Lemma 24.6.1. *Let $\lambda_1, \dots, \lambda_n$ be positive numbers such that all of them are at least α and at most k of them are more than β. Then, for every $t \geq k$, there exists a polynomial $p(X)$ of degree t such that $p(0) = 1$ and*

$$|p(\lambda_i)| \leq 2\left(1 + \frac{2}{\sqrt{\beta/\alpha}}\right)^{-(t-k)},$$

for all λ_i.

Proof. Let $r(X)$ be the polynomial we constructed using Chebyshev polynomials of degree $t - k$ for which

$$|r(X)| \leq 2\left(1 + \frac{2}{\sqrt{\beta/\alpha}}\right)^{-(t-k)},$$

for all X between α and β. Now, set

$$p(X) = r(X) \prod_{i:\lambda_i > \beta} (1 - X/\lambda_i).$$

This new polynomial is zero at every λ_i greater than β, and for X between α and β

$$|p(X)| = |r(X)| \prod_{i:\lambda_i > \beta} |(1 - X/\lambda_i)| \leq |r(X)|,$$

as we always have $X < \lambda_i$ in the product. □

Applying this lemma to the analysis of the Preconditioned Conjugate Gradient, with $\beta = \mathrm{Tr}\left(\boldsymbol{L}_T^{-1}\boldsymbol{L}_G\right)^{2/3}$ and $k = \mathrm{Tr}\left(\boldsymbol{L}_T^{-1}\boldsymbol{L}_G\right)^{1/3}$, we find that the algorithm produces ϵ-approximate solutions within

$$O(\mathrm{Tr}\left(\boldsymbol{L}_T^{-1}\boldsymbol{L}_G\right)^{1/3} \ln(1/\epsilon)) = O(m^{1/3} \log n \ln 1/\epsilon)$$

iterations.

This result is due to Spielman and Woo [SW09].

24.7 Further Improvements

We now have three families of algorithms for solving systems of equations in Laplaican matrices in nearly-linear time.

- By subgraph preconditioners. These basically work by adding back edges to the low-stretch trees. The resulting systems can no longer be solved directly in linear time. Instead, we use Gaussian elimination to eliminate the degree 1 and 2 vertices to reduce to a smaller system, and then solve that system recursively. The first nearly linear time algorithm of this form ran in time $O(m \log^c n \log 1/\epsilon)$, for some constant c [ST09]. An approach of this form was first made practical (and much simpler) by Koutis, Miller, and Peng [KMP11]. The asymptotically fastest method also works this way. It runs in time $O(m \log^{1/2} m \log^c \log n \log 1/\epsilon)$, [CKM$^+$14] (Cohen, Kyng, Miller, Pachocki, Peng, Rao, Xu).
- By sparsification (see my notes from Lecture 19 from 2015). These algorithms work rather differently, and do not exploit low-stretch spanning trees. They appear in the papers [PS14, KLP$^+$16].
- Accelerating Gaussian elimination by random sampling, by Kyng and Sachdeva [KS16]. This is the most elegant of the algorithms. While the running time of the algorithms, $O(m \log^2 n \log 1/\epsilon)$ is not the asymptotically best, the algorithm is so simple that it is the best in practice. An optimized implementation appears in the package `Laplacian.jl`.

There are other algorithms that are often fast in practice, but for which we have no theoretical analysis. I suggest the Algebraic Multigrid of Livne and Brandt, and the Combinatorial Multigrid of Yiannis Koutis.

24.8 Questions

I conjecture that it is possible to construct spanning trees of even lower stretch. Does every graph have a spanning tree of average stretch $2 \log_2 n$? I do not see any reason this should not be true. I also believe that this should be achievable by a practical algorithm. The best code that I know for computing low-stretch spanning trees, and which I implemented in `Laplacians.jl`, is a heuristic based on the algorithm of Alon, Karp, Peleg and West. However, I do not know an analysis of their algorithm that gives stretch better than $O(m 2^{\sqrt{\log n}})$. The theoretically better low-stretch trees of Abraham and Neiman are obtained by improving constructions of [EEST08, ABN08]. However, they seem too complicated to be practical.

The eigenvalues of $\boldsymbol{L}_H^{-1} \boldsymbol{L}_G$ are called generalized eigenvalues. The relation between generalized eigenvalues and stretch is the first result of which I am aware that establishes a combinatorial interpretation of generalized eigenvalues. Can you find any others?

References

[ABN08] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 781–790, Oct. 2008.

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, February 1995.

[AN12] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC '12)*, pages 395–406, 2012.

[BH01] Erik Boman and B. Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[CKM$^+$14] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly mlog1/2n time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 343–352, New York, NY, USA, 2014. ACM.

[EEST08] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 32(2):608–628, 2008.

[KLP$^+$16] Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 842–850. ACM, 2016.

[KMP11] I. Koutis, G.L. Miller, and R. Peng. A nearly-mlogn time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 52nd Annual IEEE Symposium on*, pages 590–598, 2011.

[KS16] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 573–582. IEEE, 2016.

[PS14] Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 333–342, 2014.

[ST09] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2009. Available at `http://www.arxiv.org/abs/cs.NA/0607105`.

[SW09] Daniel A. Spielman and Jaeoh Woo. A note on preconditioning by low-stretch spanning trees. *CoRR*, abs/0903.2816, 2009. Available at `http://arxiv.org/abs/0903.2816`.

[Vai90] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.