Spectral Graph Theory Lecture 15

Iterative solvers for linear equations

Daniel A. Spielman October 21, 2009

15.1 Overview

In this and the next lecture, I will discuss iterative algorithms for solving linear equations in Laplacian matrices and symmetric diagonally dominant matrices. I will begin with an introduction to iterative linear solvers in general, and then finish by explaining how this is relevant to Laplacians.

15.2 Why iterative methods?

One is first taught to solve linear systems like

$$A\boldsymbol{x} = \boldsymbol{b}$$

by *direct methods* such as Gaussian elimination, computing the inverse of A, or the LU factorization. However, all of these algorithms can be very slow. This is especially true when A is sparse. Just writing down the inverse takes $O(n^2)$ space, and computing the inverse takes $O(n^3)$ time if we do it naively. This might be OK if A is dense. But, it is very wastefull if A only has $O(n)$ non-zero entries.

In general, we prefer algorithms whose running time is proportional to the number of non-zero entries in the matrix A, and which do not require much more space than that used to store A.

Iterative algorithms solve linear equations *while only performing multiplictions* by A, and performing a few vector operations. Unlike the *direct methods* which are based on elimination, the iterative algorithms do not get exact solutions. Rather, they get closer and closer to the solution the longer they work. The advantage of these methods is that they need to store very little, and are often much faster than the direct methods. When A is symmetric, the running times of these methods are determined by the eigenvalues of A.

15.3 First-Order Richardson Iteration

To get started, we will examine a simple, but sub-optimal, iterative method, Richardson's iteration. The idea of the method is to find an iterative process that has the solution to $A\boldsymbol{x} = \boldsymbol{b}$ as a fixed

point, and which converges. We observe that if $A\boldsymbol{x} = \boldsymbol{b}$, then for any α,

$$\begin{aligned} \alpha A\boldsymbol{x} &= \alpha\boldsymbol{b}, \quad \Longrightarrow \\ \boldsymbol{x} + (\alpha A - I)\boldsymbol{x} &= \alpha\boldsymbol{b}, \quad \Longrightarrow \\ \boldsymbol{x} &= (I - \alpha A)\boldsymbol{x} + \alpha\boldsymbol{b}. \end{aligned}$$

This leads us to the following iterative process:

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

where we will take $\boldsymbol{x}^0 = \boldsymbol{0}$. We will show that this converges if

$$I - \alpha A$$

has norm less than 1, and that the convergence rate depends on how much the norm is less than 1. As we are assuming A is symmetric, $I - \alpha A$ is symmetric as well, and so its norm is the maximum absolute value of its eigenvalues. Let $\lambda_1 \leq \lambda_2 \ldots \leq \lambda_n$ be the eigenvalues of A. Then, the eigenvalues of $I - \alpha A$ are

$$1 - \alpha\lambda_i,$$

and the norm of $I - \alpha A$ is

$$\max_i |1 - \alpha\lambda_i| = \left|\max\left(1 - \alpha\lambda_1, 1 - \alpha\lambda_n\right)\right|.$$

This is minimized by taking

$$\alpha = \frac{2}{\lambda_n + \lambda_1},$$

in which case the smallest and largest eigenvalues of $I - \alpha A$ become

$$\pm\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1},$$

and the norm of $I - \alpha A$ becomes

$$1 - \frac{2\lambda_1}{\lambda_n + \lambda_1}.$$

To show that $\boldsymbol{x}^t$ converges to the solution, $\boldsymbol{x}$, consider $\boldsymbol{x} - \boldsymbol{x}^t$. We have

$$\begin{aligned} \boldsymbol{x} - \boldsymbol{x}^t &= \left((I - \alpha A)\boldsymbol{x} + \alpha\boldsymbol{b}\right) - \left((I - \alpha A)\boldsymbol{x}^{t-1} + \alpha\boldsymbol{b}\right) \\ &= (I - \alpha A)(\boldsymbol{x} - \boldsymbol{x}^{t-1}). \end{aligned}$$

So,

$$\boldsymbol{x} - \boldsymbol{x}^t = (I - \alpha A)^t\boldsymbol{x},$$

and

$$\begin{aligned} \left\|\boldsymbol{x} - \boldsymbol{x}^t\right\| = \left\|(I - \alpha A)^t\boldsymbol{x}\right\| &\leq \left\|(I - \alpha A)^t\right\| \|\boldsymbol{x}\| \\ &= \|(I - \alpha A)\|^t \|\boldsymbol{x}\| \\ &\leq \left(1 - \frac{2\lambda_1}{\lambda_n + \lambda_1}\right)^t \|\boldsymbol{x}\|. \\ &\leq e^{-2\lambda_1 t/(\lambda_n + \lambda_1)} \|\boldsymbol{x}\|. \end{aligned}$$

So, if we want to get a solution $\boldsymbol{x}^t$ with

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}^t\|}{\|\boldsymbol{x}\|} \leq \epsilon,$$

it suffices to run for

$$\frac{\lambda_n + \lambda_1}{2\lambda_1} \ln(1/\epsilon) = \left(\frac{\lambda_n}{2\lambda_1} + \frac{1}{2}\right) \ln(1/\epsilon).$$

iterations. The term

$$\frac{\lambda_n}{\lambda_1}$$

is called the *condition number*[1] of the matrix A, when A is symmetric. It is often written $\kappa(A)$, and the running time of iterative algorithms is often stated in terms of this quantity. We see that if the condition number is small, then this algorithm quickly provides an approximate solution.

15.4 A polynomial approximation of the inverse

I am now going to give another interpretation of Richardson's iteration. It provides us with a polynomial in A that approximates A^{-1}. In particular, the tth iterate, $\boldsymbol{x}^t$ can be expressed in the form

$$p^t(A)\boldsymbol{b},$$

where p^t is a polynomial of degree $t-1$. To see this note that

$$\boldsymbol{x}^t = \boldsymbol{x} - (I - \alpha A)^t \boldsymbol{x} = (I - (I - \alpha A)^t)\boldsymbol{x} = (I - (I - \alpha A)^t)A^{-1}\boldsymbol{b},$$

so

$$p^t(A) = (I - (I - \alpha A)^t)A^{-1}.$$

This really is a polynomial in A. The term A^{-1} disappears, as the difference

$$(I - (I - \alpha A)^t)$$

is a sum of powers of A of degree at least 1 (the I terms cancel).

We will view $p^t(A)$ as a good approximation of A^{-1} if

$$\|Ap^t(A) - I\|$$

is small. As A, A^{-1} and I all commute, we have

$$Ap^t(A) - I = p^t(A)A - I = (I - (I - \alpha A)^t) - I = (I - \alpha A)^t.$$

Thus, the norm of this matrix is at most

$$\left(1 - \frac{2\lambda_1}{\lambda_n + \lambda_1}\right)^t.$$

[1]For general matrices, the condition number is defined to be the ratio of the largest to smallest singular value.

15.5 Faster iterations

It is natural to ask if we can find a faster iterative method, say by exploiting the values of more iterates. To do so, we again use an equation in $\boldsymbol{x}$ to define an interation of which $\boldsymbol{x}$ is a fixed point. For the rest of this lecture, define

$$M \stackrel{\text{def}}{=} I - \alpha A,$$

and remember that

$$\boldsymbol{x} = M\boldsymbol{x} + \alpha\boldsymbol{b}.$$

From this we derive

$$\begin{aligned} \omega\boldsymbol{x} &= \omega M\boldsymbol{x} + \omega\alpha\boldsymbol{b}, \quad \Longrightarrow \\ \boldsymbol{x} &= \omega M\boldsymbol{x} + (1-\omega)\boldsymbol{x} + \omega\alpha\boldsymbol{b}. \end{aligned}$$

This leads us to consider the iteration

$$\boldsymbol{x}^{t+1} = \omega M\boldsymbol{x}^t + (1-\omega)\boldsymbol{x}^{t-1} + \omega\alpha\boldsymbol{b}. \tag{15.1}$$

We have chosen this iteration so that $\boldsymbol{x}$ is a fixed point. Now, let's see how quickly it converges. As before, we find that

$$\boldsymbol{x} - \boldsymbol{x}^{t+1} = \omega M(\boldsymbol{x} - \boldsymbol{x}^t) + (1-\omega)(\boldsymbol{x} - \boldsymbol{x}^t).$$

So, let's set $\boldsymbol{y}^t$ to be the tth error vector

$$\boldsymbol{y}^t = \boldsymbol{x} - \boldsymbol{x}^t,$$

and observe that these satisfy

$$\boldsymbol{y}^{t+1} = \omega M\boldsymbol{y}^t + (1-\omega)\boldsymbol{y}^{t-1}.$$

To express this rule as multiplication by a matrix, we need to consider $\boldsymbol{y}^{t+1}$ and $\boldsymbol{y}^t$ together as one vector. We obtain

$$\begin{pmatrix} \boldsymbol{y}^{t+1} \\ \boldsymbol{y}^t \end{pmatrix} = \begin{bmatrix} \omega M & (1-\omega)I \\ I & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{y}^t \\ \boldsymbol{y}^{t-1} \end{pmatrix}.$$

Define

$$S \stackrel{\text{def}}{=} \begin{bmatrix} \omega M & (1-\omega)I \\ I & 0 \end{bmatrix},$$

and let's find the value of ω that minimizes the absolute values of the eigenvalues of S.

If $\begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix}$ is an eigenvector of S of eigenvalue λ, then

$$\begin{bmatrix} \omega M & (1-\omega)I \\ I & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} = \begin{pmatrix} \omega M\boldsymbol{u} + (1-\omega)\boldsymbol{v} \\ \boldsymbol{u} \end{pmatrix} = \begin{pmatrix} \lambda\boldsymbol{u} \\ \lambda\boldsymbol{v} \end{pmatrix},$$

so it must be the case that $\boldsymbol{u} = \lambda\boldsymbol{v}$. From the top row, we obtain

$$\lambda^2\boldsymbol{v} = \omega\lambda M\boldsymbol{v} + (1-\omega)\boldsymbol{v}.$$

So, if $\boldsymbol{v}$ is an eigenvector of M with eigenvalue μ_i, λ will be an eigenvalue of S provided that

$$\lambda^2 = \omega\lambda\mu_i + (1-\omega) \iff \lambda^2 - \omega\mu_i\lambda + (\omega - 1) = 0.$$

Generically each eigenvalue of M leads to two eigenvalues of S, and by counting we see that this provides all the eigenvalues of S. Solving this quadratic equation, we find eigenvalues

$$\lambda^+ = \frac{\omega\mu_i + \sqrt{\omega^2\mu_i^2 - 4(\omega-1)}}{2} \quad \text{and} \quad \lambda^- = \frac{\omega\mu_i - \sqrt{\omega^2\mu_i^2 - 4(\omega-1)}}{2}.$$

Assume that all eigenvalues of M lie strictly between $-\mu$ and μ. We then find that a good choice for ω is

$$\omega = \frac{2}{1+\sqrt{1-\mu^2}}.$$

The motivation for this choice is that it causes

$$\omega^2\mu^2 = 4(\omega - 1),$$

so for all μ_i between $-\mu$ and μ

$$\omega^2\mu_i^2 \leq 4(\omega - 1),$$

and the square of the absolute value of the resulting complex eigenvalues λ is

$$|\lambda|^2 = \frac{\omega^2\mu_i^2 + 4(\omega-1) - \omega^2\mu_i^2}{4} = (\omega - 1).$$

So, all of the eigenvalues of S are complex, and have absolute value

$$\sqrt{(\omega-1)} = \omega\mu/2.$$

To see that this is an improvement, consider the case in which μ is near 1, say $\mu = 1-\epsilon$. Then, to first order,

$$\frac{\omega\mu}{2} = \frac{\mu}{1+\sqrt{1-\mu^2}} = \frac{1-\epsilon}{1+\sqrt{1-(1-\epsilon)^2}} \sim \frac{1-\epsilon}{1+\sqrt{2\epsilon}} \sim 1 - \epsilon - \sqrt{2\epsilon}.$$

This is much further from 1 than $1-\epsilon$, and suggests that our algorithm should take around $\sqrt{\kappa(A)}$ iterations to converge, instead of $\kappa(A)$. However, the matrix S was not symmetric, so this argument is not quite precise.

To make it precise, we need to expand our initial vector in the eigenbasis of S. Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be a basis of eigenvectors of M, and for each let $\boldsymbol{v}_i^+$ and $\boldsymbol{v}_i^-$ be the two induced eigenvectors of S:

$$\boldsymbol{v}_i^+ = \begin{pmatrix} \lambda^+\boldsymbol{v}_i \\ \boldsymbol{v}_i \end{pmatrix} \quad \text{and} \quad \boldsymbol{v}_i^- = \begin{pmatrix} \lambda^-\boldsymbol{v}_i \\ \boldsymbol{v}_i \end{pmatrix}.$$

We will initialize the algorithm by setting

$$\boldsymbol{x}^0 = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{x}^1 = \boldsymbol{b}.$$

15.6 A polynomial from the second-order method

To make this precise, we will see that this method also applies a polynomial in A to b, and that it is a very good approximation of A^{-1}. Our analysis will be facilitated by a judicious choice of $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$. We set

$$\boldsymbol{x}^0 = \mathbf{0} \quad \text{and} \quad \boldsymbol{x}^1 = \boldsymbol{b}.$$

We then find that

$$\boldsymbol{y}^i = \begin{bmatrix} I & 0 \end{bmatrix} S^{i-1} \begin{bmatrix} M \\ I \end{bmatrix} \boldsymbol{y}^0.$$

So, there is a polynomial p^i of degree i such that $\boldsymbol{y}^i = p^i(A)\boldsymbol{x}$. To bound the eigenvalues of $p^i(A)$. With some computation, one can prove that the eigenvalues of $p^i(A)$ are real and have absolute value at most

$$\frac{\mu\omega}{2}(1 + i\sqrt{1-\mu^2}).$$

I thought I had a way of making this computation obvious, but I was wrong. Sorry about that.

15.7 The best polynomials

To evaluate how close $p(A)$ is to A^{-1}, we measured the eigenvalues of

$$I - Ap(A).$$

If we want to find the polynomial in A of degree $i-1$ that comes closest to A^{-1}, we can instead set

$$q(x) = 1 - xp(x),$$

and find the polynomial q that minimizes

$$|q(\lambda_i)|$$

for λ_i and eigenvalue of A. Of course, we need the constant $q(x)$ to be 1, so we restrict $q(0) = 1$. It is possible to find the optimal polynomial q under the assumption that the eigenvalues of A lie between λ_1 and λ_n. If $\lambda_1 > 0$, then the optimal polynomial of degree i may be constructed using Chebyshev polynomials, and guarantees that

$$\left|q^i(\lambda)\right| \leq 2\left(1 - \frac{2}{\sqrt{\lambda_n/\lambda_1} + 1}\right)^i,$$

for all $\lambda \in [\lambda_1, \lambda_n]$. So, to get error ϵ, the best polynomial will require degree around

$$\ln(1/\epsilon)\left(\sqrt{\lambda_n/\lambda_1} + 1\right)/2.$$

This is quadratically better than we got from the Richardson iteration.

15.8 The Conjugate Gradient

For positive-definite systems, there is an algorithm that dominates all of these: the Conjugate Gradient. I do not have time to explain how it works, but I can tell you what it does. In the ith iteration, it finds the optimal approximation $\boldsymbol{x}^i$ to $\boldsymbol{x}$ in the span of

$$\left\{\boldsymbol{b}, A\boldsymbol{b}, A^2\boldsymbol{b}, \ldots, A^{i-1}\boldsymbol{b}\right\}.$$

It does this while performing an amount of work comparable to that done by the second-order method: one matrix multiplication and a couple of vector operations per iteration.

However, the Conjugate Gradient defines optimality in a slightly counter-intuitive way. It minimizes

$$\left\|\boldsymbol{x}^i - \boldsymbol{x}\right\|^A \stackrel{\text{def}}{=} \sqrt{(\boldsymbol{x}^i - xx)^T A(\boldsymbol{x}^i - xx)}.$$

However, for many applications this is actually a better way to measure error than the ordinary Euclidean norm!

By using Chebyshev polynomials, we can show that the Conjugate Gradient method will always find a vector $\boldsymbol{x}^i$ in the ith iteration that satisfies

$$\left\|\boldsymbol{x} - \boldsymbol{x}^i\right\|_A \leq 2\left(1 - \frac{2}{\sqrt{\lambda_n/\lambda_1} + 1}\right)^i \|\boldsymbol{x}\|_A.$$

15.9 Solving equations in Laplacians

It may seem strange to solve linear equations in Laplacian matrices, since they are degenerate. It becomes less strange once you observe that we know what their nullspace is, and can restrict ourselves to working orthogonal to this nullspace. In this case, we only care about the non-zero eigenvalues. The problem of solving linear equations in Laplacians arises in many applications. I hope I have time to mention some in class.

In the meantime, let's look at what happens when we try to solve an equation in the Laplacian of a path graph. If we are going to use an iterative method, it is clear that we will need at least $\Theta(n)$ iterations. After all, it takes $n/2$ iterations before the value of $\boldsymbol{x}(1)$ has any dependence on $\boldsymbol{b}(n/2)$. On the other hand, the unweighted path graph has $\lambda_n/\lambda_2 = O(n^2)$, so an iterative method can converge in $O(n)$ iterations. This tells us that, up to constant factors, the dependence of the convergence rate on λ_n/λ_2 cannot be improved.

On the other hand, we see that if λ_2 is large, then we can solve equations in the Laplacian very quickly.

In the next lecture, I will show you a technique for avoiding a dependence on λ_n/λ_2 entirely. By approximating Laplacians by Laplacians of simpler graphs, we will see how to solve any Laplacian system in time $O(m^{4/3})$.

15.10 Research Project

The reason that I taught this lecture this way is that I have been trying to find a more intuitive explanation for the quadratic speedup of the Chebyshev method over Richardon's. I don't like merely appealing to the properties of Chebyshev polynomials. In fact, I would really prefer a geometric explanation. Please help me find one!