Iterative Solvers for Linear Systems and Preconditioning

Yihan Gao

April 1, 2015

Table of Contents

Iterative Methods for Linear System Solving

- Given linear system $\mathbf{Ax} = \mathbf{b}$, we could solve them via direct methods such as Gaussian elminiation. But such algorithms can be very slow, especially when $\mathbf{A}$ is sparse.
- Iterative algorithms solve linear equations while only performing multiplications by $\mathbf{A}$ and a few other vector operations. They do not find exact solutions, but they get closer to the solution with each iteration.
- Throughout this presentation we will assume that $\mathbf{A}$ is positive definite or positive semidefinite.

First-Order Richardson Iteration

Richardson's iteration is an iterative process that has the solution to $\mathbf{Ax} = \mathbf{b}$ as a fixed point. Note that if $\mathbf{Ax} = \mathbf{b}$, then for any α,

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

The last step can be viewed as an iterative update. It converges if $I - \alpha\mathbf{A}$ has norm less than 1, the convergence rate depends on how much the norm is less than 1.

Convergence Rate of Richardson Iteration

Let $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of A, then the eigenvalues of $I - \alpha\mathbf{A}$ are $1 - \alpha\lambda_i$, and the norm of $I - \alpha\mathbf{A}$ is:

$$\max(|1 - \alpha\lambda_1|, |1 - \alpha\lambda_n|)$$

It is minimized by

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

And the norm is

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

Convergence Rate of Richardson Iteration

The update formula of Richardson Iteration

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

Rearranging the terms, we see that:

$$\mathbf{x} - \mathbf{x}^{(t)} = (I - \alpha \mathbf{A})(\mathbf{x} - \mathbf{x}^{(t-1)})$$

Thus we can get ϵ-approximation of $\mathbf{x}$ by running for about

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

iterations.

Polynomial approximation of the inverse

Another way of viewing Richardson's iteration is that it provides us with a polynomial in $\mathbf{A}$ that approximates $\mathbf{A}^{-1}$. In particular, it can be expressed as:

$$\mathbf{x}^{(t)} = p^t(\mathbf{A})\mathbf{b}$$

where p^t is a polynomial of degree t.
Our goal is

$$||p^t(\mathbf{A})\mathbf{b} - \mathbf{x}|| = ||p^t(\mathbf{A})\mathbf{A}\mathbf{x} - \mathbf{x}|| \leq \epsilon||\mathbf{x}||$$

In general a polynomial p^t computes a solution to precision ϵ if

$$||\mathbf{A}p^t(\mathbf{A}) - I|| \leq \epsilon$$

Better Polynomials

Thus, if we can find better polynomials p^t that approximates $\mathbf{A}^{-1}$, we can find better iterative methods. Note that the eigenvalues of $\mathbf{A}p^t(\mathbf{A}) - I$ are $\lambda_i p^t(\lambda_i) - 1$, therefore it suffices to find a polynomial p^t such that

$$|\lambda_i p^t(\lambda_i) - 1| \leq \epsilon$$

Define $q(x) = 1 - xp^t(x)$, then it suffices to find a polynomial q of degree $t + 1$ such that

$$q(0) = 1$$

$$|q(x)| \leq \epsilon, \text{ for } x \in [\lambda_1, \lambda_n]$$

Chebyshev Polynomials

We can construct such q using the Chebyshev polynomials. The tth Chebyshev polynomial, written as T_t, can be defined as the polynomial such that

$$\cos(tx) = T_t(\cos(x))$$

Following the definition, it is easy to see that $-1 \leq T_t(x) \leq 1$ for all $x \in [-1, 1]$. For values outside $[-1, 1]$, we have the following property:

$$T_t(1+\gamma) \geq (1+\sqrt{2\gamma})^t/2, \text{ for } \gamma > 0$$

It means that $T_t(x)$ grows very quickly on $x \in [1, \infty]$.

Chebyshev Polynomials

We perform a linear map on T_t so that $[-1,1]$ is mapped to $[\lambda_{min}, \lambda_{max}]$. More specifically, we define

$$l(x) = \frac{\lambda_{max} + \lambda_{min} - 2x}{\lambda_{max} - \lambda_{min}}$$

so that

$$l(x) \in [-1,1], \text{ for } x \in [\lambda_{min}, \lambda_{max}]$$

Let

$$q(x) = \frac{T_t(l(x))}{T_t(l(0))}$$

It is obvious that $q(0) = 1$.

Chebyshev Polynomials

$$q(x) = \frac{T_t(l(x))}{T_t(l(0))}$$

Note that for $x \in [\lambda_{min}, \lambda_{max}]$, $l(x) \in [-1, 1]$, $T_t(l(x)) \in [-1, 1]$.
As for $T_t(l(0))$, notice that

$$l(0) = 1 + \frac{2\lambda_{min}}{\lambda_{max} - \lambda_{min}}$$

Thus, using the property of Chebyshev Polynomials, we get

$$q(x) \leq 2(1 + 2\sqrt{\lambda_{min}/\lambda_{max}})^{-t}, \text{ for } x \in [\lambda_{min}, \lambda_{max}]$$

Conjugate Gradient

Sometimes, it is useful to measure error in the matrix norm, which is defined by:

$$||x||_A = \sqrt{x^T Ax}$$

Conjugate Gradient Algorithm is one such algorithm that minimizes the matrix norm of residual. It begins with vector **b**, and after t iterations it produces a vector that is in the span of

$$\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^t\mathbf{b}\}$$

The Conjugate Gradient will find the vector $\mathbf{x}_t$ in this subspace that minimizes the error in the A-norm. In other words, Conjugate Gradient method is the optimal iterative method in terms of A-norm.

Error in A-norm

$$||x_t - x||_A^2 = x_t^T A x_t - 2x^T A x_t + x^T A x = x_t^T A x_t - 2b^T x_t + x^T A x$$

Let $p_0, \ldots, p_t$ be a basis of this subspace, and let

$$x_t = \sum_{i=0}^{t} c_i p_i$$

Then we have,

$$\begin{aligned} x_t^T A x_t - 2b^T x_t &= (\sum_{i=0}^{t} c_i p_i)^T A (\sum_{i=0}^{t} c_i p_i) - 2b^T (\sum_{i=0}^{t} c_i p_i) \\ &= \sum_{i=0}^{t} c_i^2 p_i^T A p_i - 2 \sum_{i=0}^{t} c_i b^T p_i + \sum_{i \neq j} c_i c_j p_i^T A p_j \end{aligned}$$

Conjugate Gradient Algorithm

$$||x_t - x||_A^2 = \sum_{i=0}^{t} c_i^2 p_i^T A p_i - 2 \sum_{i=0}^{t} c_i b^T p_i + \sum_{i \neq j} c_i c_j p_i^T A p_j + \text{const}$$

To simplify the optimization, the Conjugate Gradient will choose p_i such that $p_i^T A p_j = 0$ for all $i \neq j$. In that case, the objective function becomes

$$\sum_{i=0}^{t} (c_i^2 p_i^T A p_i - 2 c_i b^T p_i)$$

This is minimized by setting derivatives to zero, which gives

$$c_i = (b^T p_i)/(p_i^T A p_i)$$

Computing the basis vectors

p_i can be computed methods similar to Gram-Schmidt Process

$$p_{t+1} = Ap_t - \sum_{i=0}^{t} p_i \frac{(Ap_t)^T Ap_i}{p_i^T Ap_i}$$

Actually the last summation only has two non-zero terms since Ap_i is in the span of $p_0, \ldots, p_{i+1}$ and is orthogonal to Ap_t when $i < t - 1$. Therefore each iteration of Conjugate Gradient takes $O(1)$ matrix multiplication and $O(1)$ vector operations.

Preconditioning

Preconditioning is an approach to solving linear equations in a matrix A by finding a matrix B that approximates A, but easier to solve. Remember that B is an ϵ-approximation of A if

$$(1-\epsilon)A \preceq B \preceq (1+\epsilon)A$$

We show that if **A** is an ϵ-approximation of **B**, then $\mathbf{B}^{-1}\mathbf{b}$ is not far from x in A-norm.

$$\begin{aligned} ||B^{-1}b - x||_A &= ||A^{1/2}B^{-1}b - A^{1/2}x|| \\ &= ||A^{1/2}B^{-1}(Ax) - A^{1/2}x|| \\ &\leq ||A^{1/2}B^{-1}A^{1/2} - I||||A^{1/2}x|| \\ &= ||A^{1/2}B^{-1}A^{1/2} - I||||x||_A \end{aligned}$$

Preconditioning

$$||B^{-1}b - x||_A \leq ||A^{1/2}B^{-1}A^{1/2} - I||||x||_A$$

Note that $A^{1/2}B^{-1}A^{1/2}$ is similar to $B^{-1/2}AB^{-1/2}$, therefore

$$\begin{aligned}\lambda_{max}(A^{1/2}B^{-1}A^{1/2}) &= \lambda_{max}(B^{-1/2}AB^{-1/2}) \\ &= \max_x \frac{x^T B^{-1/2}AB^{-1/2}x}{x^T x} \\ &= \max_{y=B^{-1/2}x} \frac{y^T A y}{y^T B y} \\ &\leq 1 + \epsilon\end{aligned}$$

Similarly, $\lambda_{min}(A^{1/2}B^{-1}A^{1/2}) \geq 1 - \epsilon$, therefore,

$$||A^{1/2}B^{-1}A^{1/2} - I|| \leq \epsilon$$

Preconditioned Iterative Methods

Preconditioning can be applied together with iterative methods, if B can be easily inverted, then we can trasnform the equations by:

$$B^{-1}Ax = B^{-1}b$$

Then we can use iterative methods on matrix $B^{-1}A$, but whenever we need to compute $B^{-1}Ax$ for some vector x, we first compute Ax, then use a solver of B to compute $B^{-1}Ax$.

Preconditioning by Trees

If A is the Laplacian matrix of a graph G, then it is possible to precondition A by the Laplacian matrix of a subgraph H. For any subgraph H of G

$$L_H \preceq L_G$$

Suppose we can find subgraph H that are easy to invert and the largest eigenvalue of $L_H^{-1} L_G$ is not too big, then we can use L_H as preconditioner. In particular, if H is a spanning tree of G, then L_H is easily invertible.

Low-stretch spanning tree

Write L_G as sum of edge Laplacians:

$$L_G = \sum_{(u,v)\in E} w_{u,v} L_{u,v} = \sum_{(u,v)\in E} w_{u,v} (\chi_u - \chi_v)(\chi_u - \chi_v)^T$$

Let's consider the trace of $L_H^{-1} L_G$, we have

$$\begin{aligned}\mathbf{Tr}(L_T^{-1} L_G) &= \sum_{(u,v)\in E} w_{u,v} \mathbf{Tr}(L_T^{-1} (\chi_u - \chi_v)(\chi_u - \chi_v)^T) \\ &= \sum_{(u,v)\in E} w_{u,v} (\chi_u - \chi_v)^T L_T^{-1} (\chi_u - \chi_v)\end{aligned}$$

Note that, $(\chi_u - \chi_v)^T L_T^{-1} (\chi_u - \chi_v)$ is the effective resistance between u and v in T.

Low-stretch spanning tree

Since T is a spanning tree, the effective resistance between u and v is equal to the distance in T. Let $w_1, \ldots, w_k$ be the weights of edges on the path between u and v, then

$$(\chi_u - \chi_v)^T L_T^{-1} (\chi_u - \chi_v) = \sum_{i=1}^{k} \frac{1}{w_i}$$

The term

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

is defined to be the *stretch* of edge (u, v) with respect to the tree T.

Preconditioning by Trees

- There are efficient algorithms that can find low-stretch spanning tree. In particular, we can find spanning tree with sum of stretchs at most $O(m \log n \log \log n)$ in time $O(m \log n \log \log n)$.
- Therefore, the Preconditioned Conjugate Gradient will require at most $O(m^{1/2} \log n)$ iterations, each iteration requires one multiplication by L_G and one linear solve in L_T.
- In fact, it is possible to get algorithms that solve linear systems in Laplacians in time $O(m \log n \log \log n \log \epsilon)$ by combining low-stretch spanning trees and high-quality graph sparsifiers.

References

- Lecture Notes from Daniel A Spielman:
  www.cs.yale.com/homes/spielman/561/lec17-12.pdf
  www.cs.yale.com/homes/spielman/561/lec18-12.pdf
  www.cs.yale.com/homes/spielman/561/lec19-12.pdf
- 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.