Natural Algorithm **July 27, 2016**

Linear System - Overview

Notes: Chi-Ning Chou

In which we have an overview about the modern algorithms for linear system.

Linear system problem, given A and $\mathbf{b}$ asking for $\mathbf{x}^* := \min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}||$, is ubiquitous in almost every applications in computer science. How to solve linear system faster is one of the major problem in algorithm design. In linear algebra class, students have learned how to use Gaussian elimination to solve general linear system, which takes $O(n^3)$ time where n is the size of the matrix. This is obviously non-optimal since there are only $O(n^2)$ elements in the problem.

Nevertheless, computer scientists came up with a bunch of *iterative method* for linear system. These algorithms keep updating current approximated solution and guarantee that the next approximation could be better. It turns out that although these iterative methods only produce approximated solution instead of exact solution, the computational time could be very fast. Furthermore, as real-life applications seldom encounter general linear system, once we focus on specific class of linear system, the computational time could be almost linear [KMP12] in the number of nonzero entries in the matrix.

1 Algorithms for Linear System

In this series of posts, we are going to focus on the iterative algorithms for linear system. These algorithms will produce approximation to the least square solution $\mathbf{x}^* = A^+\mathbf{b}$ of linear system $A\mathbf{x} = \mathbf{b}$, where A^+ is the pseudoinverse of A. Different algorithms proved the error bound w.r.t. different norms and focus on different classes of matrices. The following is a list for some well-known algorithms and their results:

Method	Time	Error norm	Matrix
Randome Kaczmarz	$O(\tilde{\kappa}(A)\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_2$	Arbitrary
Steepest gradient descent	$O(t_A \cdot \kappa(A)\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_A$	Symmetric + PD
Conjugate gradient descent	$O(t_A \cdot \sqrt{\kappa(A)}\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_A$	Symmetric + PD
Conjugate gradient descent (Preconditioned by tree)	$O(m^{3/2}\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_A$	Symmetric + PD
Conjugate gradient descent (Preconditioned by tree)	$O(m^{4/3}\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_A$	Laplacian
KMP SDD solver	$\tilde{O}(m\log n\log\frac{1}{\epsilon})$	$\|\|\cdot\|\|_A$	SDD

Table 1: Current common linear system solvers. $\tilde{\kappa}(A)$ is the average condition number of A, $\kappa(A)$ is the condition number of A, t_A is the computation time of A timing a vector, m is the number of nonzero entries.

Here, I will introduce the high level idea of each algorithm and leave the details in later posts.

1.1 Random Kaczmarz method

In Kaczmarz method, one think of each row in the linear system as a subspace and project current solution to a chosen subspace at each iteration. Deterministic Kaczmarz method can be very slow while the random Kaczmarz method has a nice convergence rate with high probability.

Theorem 1 *After running $O(\tilde{\kappa}(A)\log\frac{1}{\epsilon})$ iterations, with probability 0.9, we have*

$$||\mathbf{x}_T - \mathbf{x}^*||^2 \leq \epsilon^2 ||\mathbf{x}^*||^2$$

, where $\tilde{\kappa}(A)$ is the average conditional number defined as

$$\tilde{\kappa}(A) := ||A||_F ||A^+|| \tag{1}$$

, where $||\cdot||_F$ is the Frobenius norm and A^+ is the pseudo-inverse of A.

1.2 Steepest gradient descent method

Here, we consider symmetric and positive definite matrix A. In gradient-based method for linear system, one think of solving the problem by minimizing the objective function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and use the gradient of $f(\cdot)$, *i.e.,* $\partial f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, to update the current solution. If we chose the step size to minimize the objective function, we call it *steepest* gradient descent. Denote the execution time of multiplying matrix A with a vector as t_A, we have the following convergence theorem for steepest gradient descent method.

Theorem 2 *In time $O(t_A \cdot \kappa(A)\log\frac{1}{\epsilon})$, we have*

$$||\mathbf{x} - A^+\mathbf{b}||_A \leq \epsilon ||A^+\mathbf{b}||_A$$

, where $\kappa(A)$ is the conditional number of A defined as $\kappa(A) := \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$ and $||\cdot||_A$ is the matrix norm defined as $||\mathbf{v}||_A := \sqrt{\mathbf{v}^T A\mathbf{v}}$.

1.3 Conjugate gradient descent

Conjugate gradient descent method provides a square-root saving of the conditional number with a simple observation. In gradient descent, one update the current solution with gradient vector $\mathbf{r}_t = \mathbf{b} - A\mathbf{x}_{t-1}$. As a result, the solution at the t-th iteration must lie in the t-th *Krylov subspace* defined as $\mathcal{K}_t := \text{span}\{\mathbf{b}, A\mathbf{b}, \dots, A^{t-1}\mathbf{b}\}$. Thus, the philosophy of conjugate gradient descent is straightforward: *at the t-th iteration, minimize the objective function in $\mathbf{x}_0 + \mathcal{K}_t$.*

Theorem 3 *In time $O(t_A \cdot \sqrt{\kappa(A)}\log\frac{1}{\epsilon})$, we have*

$$||\mathbf{x} - A^+\mathbf{b}||_A \leq \epsilon ||A^+\mathbf{b}||_A$$

Remark 4 *The algorithm of conjugate gradient descent does two things in each iteration:*

1. *Find the t-th A-orthogonal vector $\mathbf{p}_t$ in $\mathcal{K}_t$ by Gram-Schmidt orthogonalization.*

2. *Update $\mathbf{x}_{t-1}$ with $\mathbf{p}_t$.*

Remark 5 *To analyze the conjugate gradient descent, one think of the minimization in Krylov subspace as finding a polynomial in A to minimize the objective function:*

$$\frac{1}{2}||\mathbf{x}_t - \mathbf{x}^*||_A^2 = \min_{q\in\mathcal{Q}_t} ||q(A)(\mathbf{x}_0 - \mathbf{x}^*)||_A^2 \tag{2}$$

, where $\mathcal{Q}_t$ is the set of all polynomials of degree t and evaluate 1 at 0. Then, one upper bound the r.h.s. with polynomial in the eigenvalues of A and find a good enough polynomial to upper bound the error. The final upper bound is proved with Chebyshev polynomial.

1.4 Other advanced method

The above algorithms are classical iterative method for linear system, which focus on a general class of matrices, *i.e.,* the symmetric and positive definite matrix. The convergence rate is logarithmic in the inverse of the tolerance and has certain dependence on the condition number. Soon, people found that it's not easy to improve conjugate gradient method until they came up with the brilliant idea of *preconditioning*. Once the matrix has nice structure property, it will be simple to find a good *preconditioner* and make the algorithm faster. Using this idea, a line of works [ST08], [ST11], [ST14] open the studies toward almost linear time algorithm for linear system with SDD matrix. In next section, we are going to see how do preconditioning work and the high level intuition about these almost linear-time algorithms.

2 Preconditioning

The idea of preconditioning is actually quite simple and naive. Given a linear system $A\mathbf{x} = \mathbf{b}$, we find a *preconditioner* P and turn to solve $PA\mathbf{x} = P\mathbf{b}$. As long as P is chosen properly, the approximated solution will behave well. However, there are some issues to concern:

- How fast can we solve $PA\mathbf{x} = P\mathbf{b}$?
- How fast can we find P?
- How good the solution of $PA\mathbf{x} = P\mathbf{b}$ is?

2.1 How fast can we solve $PA\mathbf{x} = P\mathbf{b}$?

Let's first consider some extreme cases. If we take $P = I$, then the preconditioned system is exactly the same as the original one. That is, it is as hard as the original one, we can not gain any computational benefit. On the other hand, if we take $P = A^+$, the pseudoinverse of A, then the linear system becomes $(A^+A)\mathbf{x} = A^+\mathbf{b}$, which is trivial. However, in this case, the preconditioner A^+ will be computationally hard to compute.

2.2 How fast can we find P?

As what we discussed in the previous paragraph, a nice preconditioner in most of the time is computationally hard to find. As a result, sometimes we need to utilize the **structure** of the matrix to find a good preconditioner. In later posts, we will see how to use the property in a graph to help us finding a good preconditioner.

2.3 How good the solution of $PA\mathbf{x} = P\mathbf{b}$ is?

One can see that the null space of preconditioner P should be contained in the null space of A. Otherwise, the range space of PA will become smaller and might result in poorer approximation.

Preconditioning can help us reduce the time spending on solving the linear system. However, additional computational time has come up due to the search for good preconditioner. As a result, people started to design algorithms which can efficiently find a good enough preconditioner. In next section, we are going to see some physical metaphors of linear system which turn out to provide intuitions for us to find good preconditioner.

3 Physical Metaphors

Laplacian linear system is closely related to several applications in other fields such as electrical circuit, rubber band system, and social network. In this section, we discuss the high level intuitions behind these reductions and give mathematical connection.

3.1 Rubber band system

Consider a rubber band system with graph G defining its connectivity. Assume the plasticity coefficient of each rubber band is the same, *i.e.,* the force is proportional to the distance between two connected vertices. Let $\mathbf{x} = x_1, \dots, x_n$ denote the position of each vertex, the total energy of the rubber band system is

$$E(\mathbf{x}) = \frac{1}{2}\sum_{i\sim j}(x_i - x_j)^2 = \frac{1}{2}\mathbf{x}^T L\mathbf{x} \tag{3}$$

From the physics world, we know that as long as we fix the position of two vertices in the rubber band system, the whole system will evolve to a configuration that minimizes its energy. Concretely, if we fix

$$x_1 = a,\ x_2 = b \tag{4}$$

The rest of the vertices will follow:

$$\frac{\partial E(\mathbf{x})}{x_i} = \sum_{j:i\sim j} 2(x_i - x_j) = 0,\ \forall i = 3, 4, \dots, n \tag{5}$$

Furthermore, it is equivalent to solve the following linear system:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ & & & & \\ & & \text{L'} & & \\ & & & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a \\ b \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

, where L' is the submatrix of L consisting the third row to the last row.

Intuition (Laplacian linear system in rubber band system)

The modified Laplacian linear system in rubber band system is solving the stable configuration with two fixed vertices.

3.2 Electrical circuit

Consider a electrical circuit defined by graph G with unit resistance on each edge. Set the voltage of two endpoints as $v_1 = 1$ and $v_2 = 0$. The whole electrical circuit will evolve to a configuration that is stable, where on each edge, the Ohm's law is obeyed and on each vertex, the conservation law is satisfied. Denote the voltage of vertex i as v_i, we have

$$\begin{aligned} &\text{(Ohm's law)} & i_{ab} = v_a - v_b & \quad , \ \forall a,b \in V \\ &\text{(Conservation law)} & \sum_{b:a\sim b} i_{ab} = 0 & \quad , \ \forall a \in V\backslash\{1,2\} \end{aligned}$$

Now, from physics, the energy dissipation of the electrical circuit is

$$\mathcal{E}(\mathbf{i}) = \frac{1}{2}\sum_{a\sim b} i_{ab}^2 = \frac{1}{2}\sum_{a\sim b}(v_a - v_b)^2 = \mathbf{v}^T L\mathbf{v} \tag{6}$$

Intuition (Laplacian linear system in electrical circuit)

Using Laplacian linear system to compute effective resistance when given unit current.

References

[KMP12] Ioannis Koutis, Gary L Miller, and Richard Peng. A fast solver for a class of linear systems. *Communications of the ACM*, 55(10):99–107, 2012.

[ST08] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *arXiv preprint arXiv:0809.3232*, 2008.

[ST11] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[ST14] Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.