An Introduction to Conjugate Gradient

Tyler Chen

This is the first piece from a series on the Conjugate Gradient algorithm. I have split up the content into the following pages:

- Introduction to Krylov subspaces
- Arnoldi and Lanczos methods
- Derivation of CG
- CG is Lanczos in disguise
- Error bounds for CG
- Finite precision CG
- Current Research

All of the pages have been compiled into a single pdf document to facilitate offline reading.

The following are some supplementary pages which are not directly related to Conjugate Gradient, but somewhat related:

- The Remez Algorithm

The intention of this website is not to provide a rigorous explanation of the topic, but rather, to provide some (hopefully useful) intuition about where this method comes from, how it works in theory and in practice, and what people are currently interested in learning about it. I do assume some linear algebra background (roughly at the level of a first undergrad course in linear algebra), but I try to add some refreshers along the way. My hope is that it can be a useful resources for undergraduates, engineers, tech workers, etc. who want to learn about some of the most recent developments in the study of Conjugate Gradient (i.e. communication avoiding methods work).

If you are a bit rusty on your linear algebra I suggest taking a look at the Khan Academy videos. For a more rigorous and much broader treatment of iterative methods, I suggest Anne Greenbaum's book on the topic. A popular introduction to Conjugate Gradient in exact arithmetic written by Jonathan Shewchuk can be found here. Finally, for a much more detailed overview of modern analysis of the Lanczos and Conjugate Gradient methods in exact arithmetic and finite precision, I suggest Gerard Meurant and Zdenek Strakos's report.

Motivation

Solving a linear system of equations $Ax = b$ is one of the most important tasks in modern science. A huge number of techniques and algorithms for dealing with more complex equations end up using linear approximations. As a result, applications such as weather forecasting, medical imaging, and training neural nets all require repeatedly solving linear systems to achieve the real world impact that we often take for granted. When A is symmetric and positive definite (if you don't remember what that means, don't worry, I have a refresher below), the Conjugate Gradient algorithm is a very popular choice for methods of solving $Ax = b$.

This popularity of CG is due to a couple factors. First, like most Krylov subspace methods, CG is *matrix free*. This means that you don't ever need to explicitly represent A as a matrix, you only need to be able to compute the product $v \mapsto Av$ for a given input vector v. For very large problems this means a big reduction in storage, and if A has some structure (eg, A comes from a DFT, difference/integral operator, is very sparse, etc.), it allows the algorithm to take advantage of fast matrix vector products. Second, CG only requires $\mathcal{O}(n)$ additional storage to run (as compared to $\mathcal{O}(n^2)$ that many other algorithms require). This can be very useful when the size of the system is very large as it reduces the communication costs of moving data in and out of memory/caches.

Measuring the accuracy of solutions

Perhaps the first question that should be asked about any numerical method is , *does it solve the intended problem?* In the case of solving linear systems, this means asking *does the output approximate the true solution?* If not, then there isn't much point using the method.

Let's quickly introduce the idea of the *error* and the *residual*. These quantities are both useful (in different ways) for measuring how close the approximate solution $\tilde{x}$ is to the true solution $x^* = A^{-1}b$.

The *error* is simply the difference between x and $\tilde{x}$. Taking the norm of this quantity gives us a scalar value which measures the distance between x and $\tilde{x}$. In some sense, this is perhaps the most natural way of measuring how close our approximate solution is to the true solution. In fact, when we say the sequence $x_0, x_1, x_2, \ldots$ converges to x_*, we mean that the scalar sequence,$\|x^* - x_0\|, \|x^* - x_1\|, \|x^* - x_2\|, \ldots$ converges to zero. Thus, solving $Ax = b$ could be written as minimizing $\|x - x^*\| = \|x - A^{-1}b\|$.

Of course, since we are trying to compute x^*, it doesn't make sense for an algorithm to explicitly depend on x^*. The *residual* of $\tilde{x}$ is defined as $b - A\tilde{x}$. Again, $\|b - Ax^*\| = 0$, and since x^* is the only point where this is true, minimizing $\|b - Ax\|$ gives the true solution. The advantage is that we can easily compute the residual $b - A\tilde{x}$ once we have our numerical solution $\tilde{x}$, while there is not necessarily a good way to compute the error $x^* - x$.

Krylov subspaces

From the previous section, we know that minimizing $\|b - Ax\|$ will give the solution x^*. Unfortunately, this problem is "just as hard" as solving $Ax = b$.

We would like to relax this problem in some way to make it "easier". One way to do this is to restrict the values that x can be. For instance, we can enforce that x comes from a smaller set of values which should make the problem of minimizing $\|b - Ax\|$ simpler (since there are less possibilities for x). For instance, if we say that $x = cy$ for some fixed vector y, then this is a scalar minimization problem. Of course, by restricting what values we choose for x it is quite likely that we will not longer be able to exactly solve $Ax = b$.

It then makes sense to try to balance the difficulty of the problems we have to solve at each step with the accuracy of the solutions they give. One way to do this is to start with an easy problem and get a very approximate solution, and then gradually increase the difficulty of the problem while refining the solution. If we do it in the right way, it's plausible that "increasing the difficulty" of the problem we are solving won't lead to extra work at each step, since we might be able to take advantage of having an approximate solution from a previous step.

Suppose we have a sequence of subspaces $V_0 \subset V_1 \subset V_2 \subset \cdots V_m$. Then we can construct a sequence of iterates, $x_0 \in V_0, x_1 \in V_1, \ldots$. If, at each step, we ensure that x_k minimizes $\|b - Ax\|$ over V_k, then the norm of the residuals will decrease (because $V_k \subset V_{k+1}$).

Ideally this sequences of subspaces would:

1. be easy to construct
2. be easy to optimize over (given the previous work done)
3. eventually contain the true solution

We now formally introduce Krylov subspaces, and show that they can satisfy these properties.

The k-th Krylov subspace generated by a square matrix A and a vector v is defined to be,

$$\mathcal{K}_k(A, v) = \text{span}\{v, Av, \ldots, A^{k-1}v\}$$

First, these subspaces are relatively easy to construct because we can get a basis by repeatedly applying A to v. In fact, we can fairly easily construct an orthonormal basis for these spaces with the Arnoldi/Lanczos algorithms.

Therefore, if we can find a quantity which can be optimized over each direction of an orthonormal basis independently, then optimizing over these expanding subspaces will be easy because we only need to optimize in a single new direction at each step.

We now show that $\mathcal{K}_k(A, b)$ will eventually contain our solution by the time $k = n$. While this result comes about naturally from our derivation of CG, I think it is useful to relate polynomials with Krylov subspace methods early on, as the two are intimately related.

Suppose A has characteristic polynomial,

$$p_A(t) = \det(tI - A) = c_0 + c_1 t + \cdots + c_{n-1} t^{n-1} + t^n$$

It turns out that $c_0 = (-1)^n \det(A)$ so that c_0 is nonzero if A is invertible.

The Cayley-Hamilton Theorem states that a matrix satisfies its own characteristic polynomial. This means,

$$0 = p_A(A) = c_0 I + c_1 A + \cdots c_{n+1} A^{n-1} + A^n$$

Moving the identity term to the left and dividing by $-c_0$ (which won't be zero since A is invertible) we can write,

$$A^{-1} = -(c_1/c_0)I - (c_2/c_0)A - \cdots - (1/c_0)A^{n-1}$$

This says that A^{-1} can be written as a polynomial in A! (I think the coolest facts from linear algebra.) In particular,

$$x^* = A^{-1}b = -(c_1/c_0)b - (c_2/c_0)Ab - \cdots - (1/c_0)A^{n-1}b$$

That is, the solution x^* to the system $Ax = b$ is a linear combination of $b, Ab, A^2b, \dots, A^{n-1}b$ (i.e. $x^* \in \mathcal{K}_n(A, b)$). This observation is the motivation behind Krylov subspace methods. I might be useful to think of Krylov subspace methods as building low degree polynomial approximations to $A^{-1}b$ using powers of A times b (in fact Krylov subspace methods can be used to approximate $f(A)b$ where f is any function).