Introduction to Conjugate Gradient

Tyler Chen

Introduction

Solving a linear system of equations $Ax = b$ is one of the most important tasks in modern science. Applications such as weather forecasting, medical imaging, and training neural nets all require repeatedly solving linear systems.

Loosely speaking, methods for linear systems can be separated into two categories: direct methods and iterative methods. Direct methods such as Gaussian elimination manipulate the entries of the matrix A in order to compute the solution $x = A^{-1}b$. On the other hand, iterative methods generate a sequence $x_0, x_1, x_2, \ldots$ of approximations to the true solution $x^* = A^{-1}b$, where hopefully each iterate is a better approximation to the true solution.

At first glance, it may seem that direct methods are better. After all, after a known number of steps you get the exact solution. In many cases this is true, especially when the matrix A is dense and relatively small. The main drawback to direct methods is that they are not able to easily take advantage of sparsity. That means that even if A has some nice structure and a lot of entries are zero, direct methods will take the same amount of time and storage to compute the solution as if A were dense. This is where iterative methods come in. Often times iterative methods require only that the product $x \mapsto Ax$ be able to be computed. If A is sparse the product can be done cheaply, and if A has some known structure, a you might not even need to construct A. Such methods are aptly called "matrix free".

The rest of this piece gives an introduction to Conjugate Gradient, a commonly used iterative method for solving $Ax = b$ when A is symmetric positive definite. My intention is not to provide a rigorous explanation of the topic, but rather to provide some (hopefully useful) intuition about where these methods come from and why they are useful in practice. I assume some linear algebra background (roughly at the level of a first undergrad course in linear algebra).

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. Finally, for a relatively recent overview of modern analysis of the Lanczos and Conjugate Gradient methods I suggest Gerard Meurant and Zdenek Strakos's report.

MAYBE THIS SHOULD BE SPLIT UP INTO SMALLER PIECES

1. Introduction to Linear Systems/Krylov subspaces
2. Derivation of CG
3. Error bounds for CG in exact arithmetic
   - minimax
   - chebyshev
4. Remez Algorithm
5. Finite precision CG
6. Extend T algorithm

Measuring the accuracy of solutions

Perhaps the first question that should be asked about an iterative method is, "Does the sequence of approximate solutions $x_0, x_1, x_2, \ldots$ converges to the true solution? If this sequence doesn't converge to the true solution (or something close to the true solution), then it won't be very useful in solving $Ax = b$.

Let's quickly introduce the idea of the *error* and the *residual* of an approximate solution x_k. These are both useful measures of how close x_k is to $x^* = A^{-1}b$. The *error* is simply the difference between x and x_k. Taking the norm of this quantity gives us a scalar value which measures the distance between x and x_k. 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 (where $\|\cdot\|$ is the norm associated with some metric space). Thus, solving $Ax = b$ could be written as minimizing $\|x - x^*\| = \|x - A^{-1}b\|$ for some norm $\|\cdot\|$.

Of course, since we are trying to compute x^*, we don't want our algorithm to depend on x^*. The *residual* of x_k is defined as $b - Ax_k$. Again $b - Ax^* = 0$ so minimizing $\|b - Ax\|$ will give the true solution. The advantage here is of course that we can easily compute $b - Ax_k$ once we have x_k.

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 $Ax = b$. However, if we restrict x to come from a smaller set of values, then the problem become simpler. 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 might not be possible to exactly solve $Ax = b$.

We would like to somehow balance how easy the problems we have to solve with how accurate the solutions they are. One way to do this is to start with an easy problem and get a coarse problem, and then gradually increase the difficulty of the problem while refining the solution.

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 make sure 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 iterate)
3. eventually contain the true solution

We now formally introduce Krylov subspaces, and show that they 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 them by repeatedly applying A to v. In fact, we can fairly easily construct an orthonormal basis for these spaces (discussed below).

Therefore, if we have a quantity which can be optimized over each direction of an orthonomal 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 later from our description of some algorithms, I think it is useful to immediately relate polynomials with Krylov subspace methods 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 $-c_0$ 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! 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 means that the solution to $Ax = b$ is a linear combination of $b, Ab, A^2b, \ldots, A^{n-1}b$. This observation is the motivation behind Krylov subspace methods. Thus, Krylov subspace methods can be viewed 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).

Finally, we note that $x = A^{-1}b \in \mathcal{K}_n(A, b)$.

Derivation of CG

We will now derive the CG method by minimizing the A-norm of the error over successive Krylov subspaces, $\mathcal{K}_k(A, b)$.

Before we get into the details, let's define some notation and review a few key concepts from linear algebra which we will rely on when deriving the CG algorithm.

- Any inner product $\langle \cdot, \cdot \rangle$ induces a norm $\| \cdot \|$ defined by $\|x\|^2 = \langle x, x \rangle$.
- For the rest of this piece we will denote the standard (Euclidian) inner product by $\langle \cdot, \cdot \rangle$ and the induced norm by $\| \cdot \|$ or $\| \cdot \|_2$.
- A positive definite matrix A naturally induces the inner product $\langle \cdot, \cdot \rangle_A$ defined by $\langle x, y \rangle_A = \langle x, Ay \rangle = \langle Ax, y \rangle$. The associated norm, called the A-nor will be denoted by $\langle \cdot, \cdot \rangle_A$ and is defined by $\|x\|_A^2 = \langle x, x \rangle_A = \langle x, Ax \rangle = \|A^{1/2}x\|$.
- The point in a subspace V nearest to a point x is the projection of x onto V (where projection is done with the inner product and distance is measured with the induced norm.) Given an orthonormal basis for V, this amounts to summing the projection of x onto each of the basis vectors.

Now that we have that out of the way, let's begin our derivation. As stated above, we will minimize the A-norm of the error over sucessive Krylov subspaces generated by A and b. That is to say x_k will be the point so that,

$$\|x_k - x^*\|_A = \min_{x \in \mathcal{K}_k(A,b)} \|x - x^*\|_A, \qquad x^* = A^{-1}b$$

Since we are minimizing with respect to the A-norm, it will be useful to have an A-orthonormal basis for $\mathcal{K}_k(A, b)$. That is, a basis which is orthonormal in the A-inner product. For now, let's just say we have such a basis, $\{p_0, p_1, \ldots, p_{k-1}\}$. Since $x_k \in \mathcal{K}_k(A, b)$ we can write x_k in terms of this basis,

$$x_k = a_0 p_0 + a_1 p_1 + \cdots + a_{k-1} p_{k-1}$$

For convenience define $x_0 = 0$ and define $e_k = x^* - x_k$. Then,

$$e_k = e_0 - a_0 p_1 - a_1 p_1 - \cdots - a_{k-1} p_{k-1}$$

Since the coefficients for x_k were chosen to minimize the A-norm of the error, $\|e_k\|_A$, over $\mathcal{K}_k(A, b)$, we know that e_k has zero component in each of the directions $\{p_0, p_1, \ldots, p_{k-1}\}$. In particular, that means that $a_j p_j$ cancels exactly with e_0 in the direction of p_j.

We now observe that since the coefficients $a'_0, a'_1, \ldots, a'_{k-2}$ of x_{k-1} were chosen in exactly the same way, so that $a_0 = a'_0, a_1 = a'_1, \ldots, a_{k-2} = a'_{k-2}$. Therefore,

$$x_k = x_{k-1} + a_{k-1} p_{k-1}$$

and

$$e_k = e_{k-1} - a_{k-1} p_{k-1}$$

Now that we have explicitly written x_k in terms of an update to x_{k-1} this is starting to look like an iterative method!

Let's compute an explicit representation of the coefficient a_{k-1}. As previously noted, we have chosen x_k to minimize $\|e_k\|_A$ over $\mathcal{K}_k(A, b)$. Therefore, the component of e_k in each of the directions $p_0, p_1, \ldots, p_{k-1}$ must be zero. That is, $\langle e_k, p_j \rangle = 0$ for all $i = 0, 1, \ldots, k-1$.

$$0 = \langle e_k, p_{k-1} \rangle_A = \langle e_{k-1}, p_{k-1} \rangle - a_{k-1} \langle p_{k-1}, p_{k-1} \rangle_A$$

Thus

$$a_{k-1} = \frac{\langle e_{k-1}, p_{k-1} \rangle_A}{\langle p_{k-1}, p_{k-1} \rangle_A}$$

This expression might look like a bit of a roadbock, since if we knew the initial error $e_0 = x^* - 0$ then we would know the solution to the original system! However, we have been working with the A-inner product so we can write,

$$A e_{k-1} = A(x^* - x_{k-1}) = b - A x_{k-1} = r_{k-1}$$

Therefore, we can compute a_{k-1} as,

$$a_{k-1} = \frac{\langle r_{k-1}, p_{k-1} \rangle}{\langle p_{k-1}, A p_{k-1} \rangle}$$

At this point we are almost done. The last thing to do is understand how to update p_k. The first thing we might try would be to do something like Gram-Schmidt on $\{b, Ab, A^2 b, \ldots\}$ to get the p_k. This will work fine if you take some care with the exact implementation (discussed below). However, it turns out that there is a much more elegant way to construct p_k which takes advantage of the symmetry of A (also discussed below).

Since $r_k = b - A x_k$ and $x_k \in \mathcal{K}_k(A, b)$, then $r_k \in \mathcal{K}_{k+1}(A, b)$. Thus, we can obtain p_k by A-orthogonalizing r_k against $\{p_0, p_1, \ldots, p_{k-1}\}$.

Recall that e_k is A-orthogonal to $\mathcal{K}_k(A,b)$. That is,

$$\langle e_k, A^j b\rangle_A = 0, \quad j = 0,1,\ldots,k-1$$

Therefore, noting that $Ae_k = r_k$,

$$\langle r_k, A^j b\rangle_A = 0, \quad j = 0,1,\ldots,k-2$$

That is, r_k is A-orthogonal to $\mathcal{K}_{k-1}(A,b)$. In particular, this means that,

$$\langle r_k, p_j\rangle_A = 0, \quad j = 0,1,\ldots,k-2$$

That means that to obtain p_k we really only need to A-orthogonalize r_k against p_{k-1}! That is,

$$p_k = r_k + b_k p_{k-1}, \quad b_k = -\frac{\langle r_k, p_{k-1}\rangle_A}{\langle p_{k-1}, p_{k-1}\rangle_A}$$

The immediate consequence is that we do not need to save the entire basis $\{p_0, p_1, \ldots, p_{k-1}\}$, but instead can just keep x_k,r_k, and p_{k-1}. **expand on this**!!

We are now essentially done. In practice, slightly different but equivalent formulas are used to compute a_{k-1} and b_k.

Arnoldi and Lanczos Algorithms

The Arnoldi algorithm for computing an orthonormal basis for Krylov subspaces is at the core of most Krylov subspace methods. Essentially, the Arnoldi algorithm is the Gram-Schmidt procedure applied to the vectors $v, Av, A^2v, A^3v, \ldots$ in a clever way.

Recall that given a set of vectors $v_0, v_1, \ldots, v_k$ the Gram-Schmidt procedure computes an orthonormal basis $q_0, q_1, \ldots, q_k$ so that for all $j \leq k$,

$$\text{span}\{v_0, \ldots, v_j\} = \text{span}\{q_0, \ldots, q_j\}$$

The trick behind the Arnoldi algorithm is the fact that you do not need to construct the whole set $v, Av, A^2v, \ldots$ ahead of time. Instead, you can compute Aq_j in place of $A^{j+1}v$ once you have found an orthonormal basis $q_0, q_1, \ldots, q_j$ spanning $v, Av, \ldots, A^j v$.

If we assume that $\text{span}\{v, Av, \ldots A^j v\} = \text{span}\{q_0, \ldots, q_j\}$ then q_j can be written as a linear combination of $v, Av, \ldots, A^j v$. Therefore, Aq_j will be a linear combination of $Av, A^2v, \ldots, A^{j+1}v$. In particular, this means that $\text{span}\{q_0, \ldots, q_j, Aq_j\} = \text{span}\{v, Av, \ldots, A^{j+1}v\}$. Therefore, we will get exactly

the same set of vectors by applying Gram-Schmidt to $\{v, Av, \ldots, A^k v\}$ as if we compute Aq_j once we have computing q_j.

Since we obtain q_{j+1} by orthogonalizing Aq_j against $\{q_0, q_1, \ldots, q_j\}$ then q_{j+1} is in the span of these vectors. That means we can can some c_i so that,

$$q_{j+1} = c_0 q_0 + c_1 q_1 + \cdots + c_j q_j + c_{j+1} A q_j$$

If we rewrite using new scalars d_i we have,

$$Aq_j = d_0 q_0 + d_1 q_1 + \cdots + d_{j+1} q_{j+1}$$

This can be written in matrix form as,

$$AQ = QH$$

where H is "upper Hessenburg" (like upper triangular but the first subdiagonal also has nonzero entries). In fact, while we have not showed it, the entries of H come directly from the Arnoldi algorithm (just like how the entries of R in a QR factorization can be obtained from Gram Schmidt).

When A is Hermetian, then $Q^* AQ = H$ is also Hermetian. Since H is upper Hessenburg and Hermitian, it must be tridiagonal! This means that the q_j satisfy a three term recurrence,

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}$$

where $\alpha_1, \ldots, \alpha_n$ are the diagonal entries of T and $\beta_1, \ldots, \beta_{n-1}$ are the off diagonal entries of T. The Lanczos algorithm is an efficient way of computing this decomposition.

Finite Precision Conjugate Gradient

- Greenbaum analysis 89