A derivation of the Conjugate Gradient Algorithm

Tyler Chen

There are many ways to view and derive the Conjugate Gradient algorithm. To me, this is of those topics where you have to go through the explanations a few times before you start to really understanding what is going on. I'll derive the algorithm by directly minimizing by minimizing the A-norm of the error over successive Krylov subspaces, $\mathcal{K}_k(A,b)$, which to me is the most natural way to think about the algorithm. My hope is that the derivation here provides an intuitive introduction to CG. Of course, what I think is a good way to present the topic won't match up with ever reader's own preference, so I highly recommend reading through some other resources as well.

Linear algebra review

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 (Euclidian) norm by $\| \cdot \|$ or $\| \cdot \|_2$.
- A martix A is positive definite if $\langle x, Ax \rangle > 0$ for all x.
- A symmetric 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-norm 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.
- The k-th Krylov subspace generated by A and b is,

$$\mathcal{K}_k(A,b) = \operatorname{span}\{b, Ab, \ldots, A^{k-1}b\}$$

Minimizing the error

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 successive Krylov subspaces generated by A and b. That is to say x_k will be the point so that,

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

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, $\{p_0, p_1, \ldots, p_{k-1}\}$, ahead of time. Since $x_k \in \mathcal{K}_k(A,b)$ we can write x_k in terms of this basis,

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

Note that we have $x_0 = 0$ and $e_k = x^* - x_k$. Then,

$$e_k = e_0 - a_0 p_0 - a_1 p_1 - \cdots - a_{k-1} p_{k-1}$$

By definition, the coefficients for x_k were chosen to minimize the A-norm of the error, $\|e_k\|_A$, over $\mathcal{K}_k(A,b)$. Therefore, 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,

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

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

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

Finding the Search Directions

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 $\{b, Ab, A^2b, \ldots\}$ to get the p_k, i.e. Arnoldi iteration in the inner product induced by A. This will work fine if you take some care with the exact implementation. However, since A is symmetric we might hope to be able to use some short recurrence, which turns out to be the case.

Since $r_k = b - Ax_k$ 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, for $j \leq k-1$,

$$\langle e_k, A^j b \rangle_A = 0$$

Therefore, noting that $Ae_k = r_k$, for $j \leq k-2$,

$$\langle r_k, A^j b \rangle_A = 0$$

That is, r_k is A-orthogonal to $\mathcal{K}_{k-1}(A, b)$. In particular, this means that, for $j \leq k-2$,

$$\langle r_k, p_j \rangle_A = 0$$

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

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

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**!!

Putting it all together

We are now essentially done! In practice, people generally use the following equivalent formulas for a_{k-1} and b_k,

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

We can now put everything together and implement it in numpy. Note that we use f for the right hand side vector to avoid conflict with the coefficient b.

```
def cg(A,f,max_iter):
    x = np.zeros(len(f)); r = np.copy(f); p = np.copy(r); s=A@p
    nu = r @ r; a = nu/(p@s); b = 0
    for k in range(1,max_iter):
        x += a*p
        r -= a*s

        nu_ = nu
        nu = r@r
        b = nu/nu_

        p = r + b*p
        s = A@p

        a = nu/(p@s)

    return x
```