# Introduction to Conjugate Gradient

Tyler Chen

## Linear Systems

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. 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. In 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". Similarly, many iterative methods such as Conjugate Gradient do not need much additional storage to compute the solution.

The rest of this series gives an introduction to the analysis of 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 this method comes from and how it works 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. A popular introduction to Conjugate Gradient in exact arithmetic written by Jonathan

Shewchuk can be found here. Finally, for a recent 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.

## 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 the iterate $x_k$ is to the true solution $x^* = A^{-1}b$. 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. 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 $x^*$, it doesn't make sense for an algorithm to explicitly depend on $x^*$. The *residual* of $x_k$ is defined as $b - Ax_k$. Again $b - Ax^* = 0$, and minimizing $\|b - Ax\|$ gives the true solution. The advantage here is of course that we can easily compute the residual $b - Ax_k$ 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 at each step with how accurate the solutions they give are. One way to do this is to start with an easy problem and get a coarse solution, and then gradually increase the difficulty of the problem while refining the solution. If we do it in the right way, "increasing the difficulty" of the problem we are solving won't lead to extra work, because we can take advantage of having solve the easier problems at previous steps.

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 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,

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

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 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 later from our description of some algorithms, 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 $-c_0$ 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 $x^*$ to the system $Ax = b$ is a linear combination of $b, Ab, A^2b, \dots, A^{n-1}b$. 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 $x = A^{-1}b \in \mathcal{K}_n(A, b)$.

# The Arnoldi and Lanczos algorithms

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

## The Arnoldi algorithm

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 $j \leq k$,

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

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 $Aq_k$ in place of $A^{k+1}v$ once you have found an orthonormal basis $q_0, q_1, \ldots, q_k$ spanning $v, Av, \ldots, A^kv$.

If we assume that $\operatorname{span}\{v, Av, \ldots A^kv\} = \operatorname{span}\{q_0, \ldots, q_k\}$ then $q_k$ can be written as a linear combination of $v, Av, \ldots, A^kv$. Therefore, $Aq_k$ will be a linear combination of $Av, A^2v, \ldots, A^{k+1}v$. In particular, this means that $\operatorname{span}\{q_0, \ldots, q_j, Aq_k\} = \operatorname{span}\{v, Av, \ldots, A^{k+1}v\}$. Therefore, we will get exactly the same set of vectors by applying Gram-Schmidt to $\{v, Av, \ldots, A^kv\}$ as if we compute $Aq_k$ once we have computing $q_k$.

Since we obtain $q_{k+1}$ by orthogonalizing $Aq_k$ against $\{q_0, q_1, \ldots, q_k\}$ then $q_{k+1}$ is in the span of these vectors, there exist some $c_i$ so that,

$$q_{k+1} = c_0q_0 + c_1q_1 + \cdots + c_kq_k + c_{k+1}Aq_k$$

We can rearrange this (using new scalars $d_i$) to,

$$Aq_k = d_0q_0 + d_1q_1 + \cdots + d_{k+1}q_{k+1}$$

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). While I'm not going to derive them here, since 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) their explicit expressions can be easily written down.

Since $Q$ is orthogonal then, $Q^*AQ = H$, so $H$ and $A$ are similar. This means that finding the eigenvalues and vectors of $H$ will give us the eigenvalues and vectors of $A$. However, since $H$ is upper Hessenburg, then solving the eigenproblem is easier than for a general matrix.

## The Lancozs algorithm

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,

$$Aq_k = \beta_{k-1}q_{k-1} + \alpha_k q_k + \beta_k q_{k+1}$$

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.

I will present a brief derivation for the method motivated by the three term recurrence above. Since we know that the $q_k$ satisfy the three term recurrence, we would like the method to store as few of the $q_k$ as possible (i.e. take advantage of the three term recurrence as opposed to the Arnoldi algorithm).

Suppose that we have $q_k$, $q_{k-1}$, and the coefficient $\beta_{k-1}$. We need to expand the Krylov subspace to find $q_{k+1}$ in a way that takes advantage of the three term recurrence. To do this we can expand the subspace by computing $Aq_k$ and then orthogonalizing $Aq_k$ against $q_k$ and $q_{k-1}$. By the three term recurrence, $Aq_k$ will be orthogonal to $q_j$ for all $j \leq k-2$ so we do not need to explicitly orthogonalize against those vectors.

We orthogonalize,

$$\tilde{q}_{k+1} = Aq_k - \alpha_k q_k - \langle Aq_k, q_{k-1}\rangle q_{k-1}, \qquad \alpha_k = \langle Aq_k, q_k\rangle$$

and finally normalize,

$$q_{k+1} = \tilde{q}_{k+1}/\beta_j, \qquad \beta_j = \|\tilde{q}_{k+1}\|$$

Note that this is not the most "numerically stable" form of the algorithm, and care must be taken when implementing the Lanczos method in practice. We can improve stability slightly by using $Aq_k - \beta_{k-1}q_{k-1}$ instead of $Aq_k$ when finding a vector in the next Krylov subspace. This allows us to ensure that we have orthogonalized $q_{k+1}$ against $q_k$ and $q_{k-1}$ rather than just $q_k$. It also ensures that the tridiagonal matrix produces is symmetric in finite precision (since $\langle Aq_k, q_{k-1}\rangle$ may not be equal to $\beta_j$ in finite precision).

We can implement Lanczos iteration in numpy. Here we assume that we only want to output the diagonals of the tridiagonal matrix $T$, and don't need any of the vectors.

```
def lanczos(A,q0,max_iter):
    alpha = np.zeros(max_iter)
    beta = np.zeros(max_iter)
    q_ = np.zeros(len(q0))
    q = q0/np.sqrt(q0@q0)

    for k in range(max_iter):
        qq = A@q-(beta[k-1]*q_ if k>0 else 0)
        alpha[k] = qq@q
        qq -= alpha[k]*q
        beta[k] = np.sqrt(qq@qq)
        q_ = np.copy(q)
        q = qq/beta[k]

return alpha,beta
```

# A derivation of the Conjugate Gradient Algorithm

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,

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

## 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, $\{p_0, p_1, \dots, p_{k-1}\}$, 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 $\{p_0, p_1, \dots, p_{k-1}\}$. 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 $a_0', a_1', \dots, a_{k-2}'$ of $x_{k-1}$ were chosen in exactly the same way, so that $a_0 = a_0', a_1 = a_1', \dots, a_{k-2} = a_{k-2}'$. 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
```

# Error Bounds for the Conjugate Gradient Algorithm

This page is a work in progress.

In our derivation of the Conjugate Gradient method, we minimized the $A$-norm of the error over sucessive Krylov subspaces. Ideally we would like to know how quickly this method converge. That is, how many iterations are needed to reach a specified level of accuracy.

## Linear algebra review

- The 2-norm of a symmetric positive definite matrix is the largest eigenvalue of the matrix
- The 2-norm is submultiplicative. That is, $\|A\|\|B\| \leq \|AB\|$
- A matrix $U$ is called unitary if $U^*U = UU^* = I$.

## Polynomial error bounds

Previously we have show that,

$$e_k \in e_0 + \operatorname{span}\{p_0, p_1, \ldots, p_{k-1}\} = e_0 + \mathcal{K}_k(A, b)$$

Observing that $r_0 = Ae_0$ we find that,

$$e_k \in e_0 + \operatorname{span}\{Ae_0, A^2e_0, \ldots, A^ke_0\}$$

Thus, we can write,

$$\|e_k\|_A = \min_{p \in \mathcal{P}_k} \|p(A)e_0\|_A, \qquad \mathcal{P}_k = \{p : p(0) = 1, \deg p \leq k\}$$

Since $A^{1/2}p(A) = p(A)A^{1/2}$ we can write,

$$\|p(A)e_0\|_A = \|A^{1/2}p(A)e_0\| = \|p(A)A^{1/2}e_0\|$$

Now, using the submultiplicative property of the 2-norm,

$$\|p(A)A^{1/2}e_0\| \leq \|p(A)\|\|A^{1/2}e_0\| = \|p(A)\|\|e_0\|_A$$

Since $A$ is positive definite, it is diagonalizable as $U\Lambda U^*$ where $U$ is unitary and $\Lambda$ is the diagonal matrix of eigenvalues of $A$. Thus,

$$A^k = (U\Lambda U^*)^k = U\Lambda^kU^*$$

We can then write $p(A) = Up(\Lambda)U^*$ where $p(\Lambda)$ has diagonal entries $p(\lambda_i)$. Therefore, using the *unitary invariance* property of the 2-norm,

$$\|p(A)\| = \|Up(\Lambda)U^*\| = \|p(\Lambda)\|$$

Now, since the 2-norm of a symmetric matrix is the magnitude of the largest eigenvalue,

$$\|p(\Lambda)\| = \max_i |p(\lambda_i)|$$

Finally, putting everything together we have,

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \min_{p\in\mathcal{P}_k} \max_i |p(\lambda_i)|$$

Since the inequality we obtained from the submultiplicativity of the 2-norm is tight, this bound is also tight in the sense that for a fixed $k$ there exists an initial error $e_0$ so that equality holds.

Computing the optimal $p$ is not trivial, but an algorithm called the Remez algorithm can be used to compute it.

Let $L \subset \mathbb{R}$ be some closed set. The *minimax polynomial of degree $k$ on $L$* is the polynomial satisfying,

$$\min_{p\in\mathcal{P}_k} \max_{x\in L} |p(x)|, \qquad \mathcal{P}_k = \{p : p(0) = 1, \deg p \leq k\}$$

**Chebyshev bounds**

The minimax polynomial on the eigenvalues of $A$ is a bit tricky to work with. Although we can find it using the Remez algorithm, this is somewhat tedious, and requires knowledge of the whole spectrum of $A$. We would like to come up with a bound which depends on less informationabout $A$. One way to obtain such a bound is to expand the set on which we are looking for the minimax polynomial.

To this end, let $\mathcal{I} = [\lambda_{\min}, \lambda_{\max}]$. Then, since $\lambda_i \in \mathcal{I}$,

$$\min_{p\in\mathcal{P}_k} \max_i |p(\lambda_i)| \leq \min_{p\in\mathcal{P}_k} \max_{x\in\mathcal{I}} |p(x)|$$

The right hand side requires that we know the largest and smallest eigenvalues of $A$, but doesn't require any of the ones between. This means it can be useful in practice, since we can easily compute the top and bottom eignevalues with the power method.

The polynomials satisfying the right hand side are called the *Chebyshev Polynomials* and can be easily written down with a simple recurrence relation. If $\mathcal{I} = [-1, 1]$ then the relation is,

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad T_0 = 1, \quad T_1 = x$$

For $\mathcal{I} \neq [-1, 1]$, the above polynomials are simply stretched and shifted to the interval in question.

Let $\kappa = \lambda_{\max}/\lambda_{\min}$ (this is called the condition number). Then, from properties of these polynomials,

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

## The Conjugate Gradient Algorithm in Finite Precision

This page is a work in progress.

A key component of our derivations of the Lanczos and Conjugate Gradient methods was the orthogonality of certain basis vectors. In finite precision, our induction based arguments no longer hold, and so it's reasonable to expect the algorithms will fail. That said, since you're reading about these methods, they must somehow still be usable in practice. This turns out to be the case, and both methods are widely used for eignevalue problems and solving linear systems.

The first major progress in the analysis of the Lanczos algorithm was done by Chris Paige, who characterized the behavior of the method in finite precision. A similarly important analysis of Conjugate Gradient was done by Anne Greenbaum in her 1989 paper, "Behavior of slightly perturbed Lanczos and conjugate-gradient recurrences". A big takeaway from Greenbaum's analysis is that the error bound from the Chevyshev polynomials still holds in finite precision (to a close approximation). My goal here is to present the highlights of that paper.

### The results

Remez Algorithm

### Some conditions for the analysis

CG is doing the Lanczos algorithm in disguise. In particular, normalizing the residuals from CG gives the vectors $q_j$ produced by the Lanczos algorithm, and combing the CG constants in the right way gives the coefficients for the three term Lanczos recurrence.

The analysis by Greenbaum requires that the finite precision Conjugate Gradient algorithm (viewed as the Lanczos algorithm) satisfy a few properties. Namely,

- the three term Lanczos recurrence is well satisfied
- the Lanczos vectors have norm close to one
- successive Lanczos vectors are nearly orthogonal

As it turns out, nobody has actually ever proved that any of the Conjugate Variant methods used in practice actually satisfy these conditions (although some do numerically satisfy the conditions).