

Chebyshev Polynomials, applied to analyzing CG and accelerating SOR

This lecture will cover a topic used twice in Chap 6, Chebyshev Polynomials, which are used to analyze the convergence of Conjugate Gradients (CG), accelerate SOR, and other similar purposes. To further motivate the study of these and other polynomials in studying linear algebra algorithms, recall that Krylov subspace methods (KSMs) seek to find the “best” approximate solution x_k of to $Ax=b$ in a k -dimensional Krylov subspace

$$\begin{aligned} \text{script}\{K\}_k(A,B) &= \text{span}\{ b, A*b, A^2*b, \dots A^{(k-1)}*b \} \\ &= \text{span}\{ p_{(k-1)}(A)*b, \text{ where } p \text{ is a polynomial of degree } \leq k-1 \} \end{aligned}$$

In the case of CG, recall that the “best” polynomial is defined to minimize

$$\begin{aligned} || A*x_k - b ||_{\{A^{-1}\}}^2 &= (A*x_k - b)^T * A^{-1} * (A*x_k - b) \\ &= (A*p_{(k-1)}(A)*b - b)^T * A^{-1} * (A*p_{(k-1)}(A)*b - b) \\ &= (q_k(A)*b)^T * A^{-1} * (q_k(A)*b) \end{aligned}$$

$$\begin{aligned} \text{where } q_k(A) &= I - A*p_{(k-1)}(A) \text{ is a polynomial of degree } k \text{ with } q_k(0) = 1 \\ &= b^T * q_k(A)^2 * A^{-1} * b \end{aligned}$$

Since A is spd, we can write its eigendecomposition $A = Q*\Lambda*Q^T$, and let $y = Q^T*b$ to get

$$\begin{aligned} &= y^T * q_k(\Lambda)^2 * \Lambda^{-1} * y \\ &= \sum_i q_k(\lambda_i)^2 * \lambda_i^{-1} * y_i^2 \end{aligned}$$

So the “best” x_k corresponds to the “best” polynomial q_k , i.e. that minimizes this weighted sum of its squared values $q_k^*(\lambda_i)^2$ at the eigenvalues of A .

To simplify, we upper bound this by

$$\begin{aligned} &\leq \max_i | q_k(\lambda_i) |^2 * y^T * \Lambda^{-1} * y \\ &= \max_i | q_k(\lambda_i) |^2 * || b ||_{\{A^{-1}\}}^2 \end{aligned}$$

or

$$(*) \quad || A*x_k - b ||_{\{A^{-1}\}} / || b ||_{\{A^{-1}\}} \leq \max_i | q_k(\lambda_i) |$$

So if we can bound the magnitude of some “good” polynomial q_k , where “good” means $q_k(0)=1$ while $| q_k(\lambda_i) |$ is “small” for all eigenvalues λ_i of A (note that 0 can’t be an eigenvalue of A , since A is spd),

then we can get a useful bound on the convergence rate. Chebyshev polynomials, suitably scaled depending on A , will serve this purpose, and give us a bound on how fast CG converges.

Another example where polynomials can help is accelerating a splitting method, like SOR.

Recall that a splitting method produces a sequence of approximate solutions

$x_k = R*x_{(k-1)} + c$, where the exact solution of $A*x=b$ satisfies $x = R*x + c$, and so $x_k - x = R*(x_{(k-1)} - x) = R^k * (x_0 - x)$. Suppose we tried to find a linear combination

$$y_k = \sum_{i=0}^k c_{(k,i)} * x_i$$

of these approximations that converged faster. Note that if $x_0 = x$, then all subsequent $x_i = x$, so we need to have $x = \sum_{i=0}^k c_{(k,i)} * x$ or $1 = \sum_{i=0}^k c_{(k,i)}$ for all k . Then

$$\begin{aligned}
(**) \quad y_k - x &= \sum_{i=0}^k c_{(k,i)} x_i - x \\
&= \sum_{i=0}^k c_{(k,i)} (x_i - x) \\
&= \sum_{i=0}^k c_{(k,i)} R^i (x_0 - x) \\
&= p_k(R) (x_0 - x)
\end{aligned}$$

So now our goal is again to find a degree- k polynomial $p_k(R)$ that is small at the eigenvalues of R , but with the slightly different constraint that the sum of its coefficients is 1, i.e. $p_k(1) = 1$. Again, suitably scaled Chebyshev polynomials will serve this purpose. Furthermore, Chebyshev polynomials have one more important property that will make computing y_k cheap: we will only need to keep $y_{(k-1)}$ and $y_{(k-2)}$ in memory and operate on them to get y_k , not x_0 through x_k .

Now we define Chebyshev polynomials, and just the few of their many properties that we will need for our purposes.

Def: The m -th Chebyshev polynomial is defined by the 3-term recurrence

$$T_0(z) = 1, T_1(z) = z, \text{ and } T_m(z) = 2zT_{m-1}(z) - T_{m-2}(z)$$

The following lemma follows straightforwardly from the definition:

Lemma: Chebyshev polynomials have the following properties (see Lemma 6.7 in text for more):

- (a) $T_m(1) = 1$
- (b) $T_m(z) = 2^m z^m + O(z^{m-1})$
- (c) $T_m(\cos y) = \cos(m*y)$; this applies to $T_m(z)$ when $|z| \leq 1$, hence $|T_m(z)| \leq 1$ if $|z| \leq 1$.
- (d) $T_m(\cosh y) = \cosh(m*y)$; this applies to $T_m(z)$ when $z \geq 1$, hence $T_m(z) \geq 1$ if $z \geq 1$.
- (e) If m is even (resp. odd) then $T_m(z)$ is an even (resp. odd) polynomial
- (f) If $|z| \geq 1$ then $T_m(z) = .5 * [(z + \sqrt{z^2 - 1})^m + (z + \sqrt{z^2 - 1})^{-m}]$
- (g) $T_m(1 + \epsilon) \geq .5 * (1 + m * \sqrt{2 * \epsilon})$ if $\epsilon > 0$

Property (e) extends property (d) to $z \leq -1$, since $T_m(z) = (-1)^m * T_m(-z)$.

Property (g) follows from Property (f).

Properties (f) and (g) provide different ways to estimate how fast $T_m(z)$ grows for $z > 1$, and help explain the plots of $T_m(z)$ in Fig 6.6 in the text, which show that $|T_m(z)|$ grows monotonically and very rapidly outside the interval z in $[-1, 1]$, within which it is bounded by 1.

We apply this to understanding the convergence of CG using (*) as follows.

We need to pick a polynomial $q_k(z)$ with the properties that $q_k(0) = 1$, and

$|q_k(\lambda_i)|$ is small for eigenvalues λ_i of A . Let $0 < \lambda_{\min} \leq \lambda_{\max}$ be the eigenvalues of the spd matrix A , and note that if $\lambda_{\min} \leq z \leq \lambda_{\max}$, then

$$-1 \leq (\lambda_{\max} + \lambda_{\min} - 2z) / (\lambda_{\max} - \lambda_{\min}) \leq 1$$

and so

$$q_k(z) = \frac{T_k((\lambda_{\max} + \lambda_{\min} - 2z) / (\lambda_{\max} - \lambda_{\min}))}{T_k((\lambda_{\max} + \lambda_{\min}) / (\lambda_{\max} - \lambda_{\min}))}$$

satisfies $q_k(0) = 1$ as required, and for $\lambda_{\min} \leq z \leq \lambda_{\max}$ it satisfies

$$|q_k(z)| \leq 1 / T_k((\lambda_{\min} + \lambda_{\max}) / (\lambda_{\max} - \lambda_{\min}))$$

... by property (c) of the lemma above

$$\begin{aligned}
&= 1 / T_k((\kappa + 1) / (\kappa - 1)) \\
&\quad \dots \text{ where } \kappa = \lambda_{\max} / \lambda_{\min} \text{ is the condition number of } A \\
&= 1 / T_k(1 + 2/(\kappa - 1)) \\
&\leq 2 / (1 + 2\kappa/\sqrt{\kappa - 1}) \\
&\quad \dots \text{ by property (g) of the lemma}
\end{aligned}$$

Thus using (*) we can conclude that $k = O(\sqrt{\kappa})$ steps of CG are enough to decrease the error by a constant factor < 1 , and so that $O(\sqrt{\kappa})$ steps of CG are needed to converge to any fixed residual norm $\|r\|_{A^{-1}} / \|b\|_{A^{-1}}$. This proves the convergence result for CG claimed earlier.

(We pause the recorded lecture here.)

Now we consider how to accelerate the convergence of splitting methods like SOR. Now we need to pick a polynomial $p_k(z)$ with the properties that $p_k(z)$ is small for z an eigenvalue of R , and $p_k(1)=1$. Since we are assuming the splitting method converges, this means the spectral radius $\rho(R) < 1$, i.e. all the eigenvalues of R are less than 1 in absolute value. Since the useful properties of Chebyshev polynomials proven above only apply to real arguments, we need to assume R only has real eigenvalues. Assuming this is true, let ρ satisfy

$$-1 < -\rho \leq \lambda_{\min} \leq \lambda_{\max} \leq \rho < 1$$

where λ_{\min} and λ_{\max} are the smallest and largest eigenvalues of R .

In contrast to the previous analysis of CG, where these eigenvalues were only used for analysis, we actually need their values (or a bound ρ) in order to implement an accelerator. This limits the usability of this approach, but in some cases (eg Poisson equation and variations), these are known (along with faster algorithms, like multigrid).

Given ρ , we let

$$p_k(z) = T_k(z/\rho) / T_k(1/\rho)$$

Clearly $p_k(1)=1$ as desired, and if $|z| \leq \rho$ then $|p_k(z)| \leq 1/T_k(1/\rho)$.

How much faster can this converge than the original splitting method?

Assuming we pick ρ as small as possible, so $\rho = \rho(R) = \max_i |\lambda_i(R)|$,

and $\rho = 1 - \epsilon$, then again using property (g) of the Lemma we get

$$\begin{aligned}
1 / T_k(1/\rho) &= 1 / T_k(1/(1-\epsilon)) \\
&= 1 / T_k(1 + \epsilon/(1-\epsilon)) \\
&\leq 2 / (1 + k\sqrt{2\epsilon/(1-\epsilon)}) \\
&\sim 2 * (1 - k\sqrt{2\epsilon})
\end{aligned}$$

This is to be contrasted with the original splitting method, which after k steps decreases the error by $\rho^k = (1-\epsilon)^k \sim 1 - k\epsilon$

In other words, when ϵ is small, Chebyshev acceleration can reduce the number of iterations by a square root, an asymptotic improvement.

To see how to do this cheaply, we use the 3-term term recurrence

$$T_{k+1}(z) = 2zT_k(z) - T_{k-1}(z)$$

defining Chebyshev polynomials, and scale it to get a 3-term recurrence for $p_{k+1}(z)$.

Letting $\mu_k = 1/T_k(1/\rho)$ for brevity, we get

$$\begin{aligned}
p_{(k+1)}(z) &= \mu_{(k+1)} * T_{(k+1)}(z/\rho) \\
&= \mu_{(k+1)} * (2 * z/\rho * T_k(z/\rho) - T_{(k-1)}(z/\rho)) \\
&= \mu_{(k+1)} * (2 * z/\rho * p_k(z)/\mu_k - p_{(k-1)}(z)/\mu_{(k-1)}) \\
&= (2 * \mu_{(k+1)}/(\rho * \mu_k) * z * p_k(z) - (\mu_{(k+1)}/\mu_{(k-1)}) * p_{(k-1)}(z)) \\
&= \alpha_k * z * p_k(z) + \beta_k * p_{(k-1)}(z)
\end{aligned}$$

Note that $1/\mu_k = T_k(1/\rho)$ also can be computed using a 3-term recurrence, so allowing us to compute α_k and β_k cheaply too.

Applying this 3 term recurrence for $p_{(k+1)}(z)$ to $(**)$ yields

$$\begin{aligned}
y_{(k+1)} - x &= p_{(k+1)}(R) * (x_0 - x) \\
&= \alpha_k * R * p_k(R) * (x_0 - x) + \beta_k * p_{(k-1)}(R) * (x_0 - x) \\
&= \alpha_k * R * (y_k - x) + \beta_k * (y_{(k-1)} - x) \\
&= \alpha_k * R * y_k + \beta_k * y_{(k-1)} - \alpha_k * R * x - \beta_k * x
\end{aligned}$$

or

$$y_{(k+1)} = \alpha_k * R * y_k + \beta_k * y_{(k-1)} - \alpha_k * R * x - \beta_k * x + x$$

To simplify the last three terms which depend on x , use $x = R * x + c$ to get

$$\begin{aligned}
-\alpha_k * R * x - \beta_k * x + x &= -\alpha_k * (x - c) - \beta_k * x + x \\
&= (-\alpha_k - \beta_k + 1) * x + \alpha_k * c \\
&= \alpha_k * c
\end{aligned}$$

which follows from the definitions of α_k and β_k . This yields the final algorithm:

$$y_{(k+1)} = \alpha_k * R * y_k + \beta_k * y_{(k-1)} + \alpha_k * c$$

whose cost is rough the same as the original splitting method.

Recall that this depended on R having all real eigenvalues, as well as knowing a reasonably tight bound on $\rho(R)$. Let us first try applying this to Jacobi's method for our model problem, Poisson's equation on an $N \times N$ mesh. Recall that R_J was symmetric, so all real eigenvalues, and that $\rho(R_J) = \cos(\pi/(N+1)) \sim 1 - \pi^2/(2 * (N+1)^2) = 1 - O(1/N^2)$, where N is the number of mesh points in each dimension. This means Jacobi would take $O(N^2)$ iterations to converge versus $O(N)$ for the Chebyshev accelerated version, a big improvement. But we already know that we can use $SOR(w_{opt})$ to converge in $O(N)$ iterations, so it is natural to try to applying Chebyshev acceleration to $SOR(w_{opt})$ to converge in just $O(\sqrt{N})$ iterations. Unfortunately $R_{SOR}(w)$ has complex eigenvalues, so does not satisfy our assumption. Fortunately there is a clever fix: we apply $R_{SOR}(w)$ once, updating the entries of x in the usual order, and then apply it again, updating them in the reverse order. This effectively "symmetrizes" SOR , yielding $SSOR(w)$, and it can be shown that $SSOR(w)$ has all real eigenvalues. The choice of optimal w is no longer the same for $SOR(w)$, But a good choice is known with $\rho(R_{SSOR}(w)) = 1 - O(1/N)$, allowing us to use Chebyshev acceleration to converge in $O(\sqrt{N})$ iterations instead of $O(N)$ as desired. For the $N \times N$ Poisson equation, this means $O(N^2 * \sqrt{N}) = O(n^{5/4})$ flops are needed to converge, where $n = N^2$ is the size of the matrix, versus $O(n^{3/2})$ for $SOR(w_{opt})$. See section 6.5.6 of the text for more details.

This completes the analysis of all the algorithms in Table 6.1 in the textbook, showing how fast they all converge on the $N \times N$ Poisson equation.