Week 15: Monday, Nov 27–Wednesday, Nov 29

Tridiagonal reduction redux

Consider the problem of computing eigenvalues of a symmetric matrix A that is large and sparse. Using the "grail" code in LAPACK, we can compute all the eigenvalues of an n-by-n tridiagonal matrix in $O(n)$ time; but the usual Householder-based algorithm to reduce A to tridiagonal form costs $O(n^3)$. Though it's possible to maintain sparsity in some cases, it is not altogether straightforward. So let's see if we can try something different.

What we want is an orthogonal matrix Q such that $AQ = QT$, where

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Writing column j of this matrix equation gives us

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

where $\alpha_j = q_j^T A q_j$ and $\beta_{j-1} = q_{j-1}^T A q_j$. If we rearrange this slightly, we have

$$\begin{aligned} r_{j+1} &= Aq_j - q_j \alpha_j - q_{j-1}\beta_{j-1} \\ &= Aq_j - (q_j q_j^T) Aq_j - (q_{j-1} q_{j-1}^T) Aq_j \\ q_{j+1} &= r_{j+1}/\beta_j. \end{aligned}$$

That is, q_{j+1} is *exactly* what we compute if we orthogonalize Aq_j against the vectors q_j and q_{j-1}. Note that Aq_j is automatically orthogonal to $q_1, \ldots, q_{j-2}$, since $q_k^T A q_j = T_{kj}$ is zero for $k < j-1$.

This process of incrementally producing an orthonormal basis by multiplying the last vector by A and orthogonalizing the result against what came before gives us the basic Lanczos procedure. If we start with $q_1 = e_1$, the Lanczos procedure will give us the same T matrix[1] as the Householder

[1] Well, almost the same T matrix. Note that the off-diagonal elements β_j might be positive or negative in the Householder tridiagonalization, but they are always positive in the Lanczos algorithm. However, the two T matrices satisfy a similarity relation with a diagonal matrix whose diagonal entries are ± 1.

algorithm, at least in exact arithmetic. But while a step in the Householder algorithm costs $O(n^2)$, the cost of a Lanczos step is a matrix-vector multiply (which may cost less than $O(n^2)$ if A is sparse) followed by $O(n)$ work in dot products and gaxpy operations. Also, note that the Lanczos step requires only a few vectors of intermediate storage.

In exact arithmetic, we could iterate until we found $\beta_k = 0$, which would correspond to having an invariant subspace spanned by $\{q_1, \ldots, q_k\}$. Except for very special choices of starting vectors, though, this would happen only for $k = n$.

Partial tridiagonalization and Krylov subspaces

What happens when we run only a few steps of the Lanczos iteration? After m steps, we have computed m columns of Q and m rows of T; and there is some hope that we might be able to use the leading m-by-m subblock of T (i.e. the block Rayleigh quotient with the first m columns of Q) in order to extract some useful information about the spectrum of A. The reason why we might have such a hope is that we notice that the Krylov subspace spanned by $q_1, \ldots, q_m$ is also spanned by the first m iterates of a power method:

$$\mathcal{K}_m(A, q_1) = \text{span}\{q_1, q_2, \ldots, q_m\} = \text{span}\{q_1, Aq_1, \ldots, A^{m-1}q_1\}.$$

As we have previously discussed, the power basis is not a particularly useful basis for numerical computations, but it is useful for thinking about Krylov subspaces. In particular, because it contains the mth vector produced by power iteration starting from q_1, we expect that the largest *Ritz value* associated with $\mathcal{K}_m(A, q_1)$ (i.e. the largest magnitude eigenvalue of the block Rayleigh quotient $(Q_{:,1:m})^T A Q_{:,1:m}$) will be at least as large as the Rayleigh quotient estimate produced by m steps of power iteration.

Of course, the Krylov subspace $\mathcal{K}_m(A, q_1)$ has a lot more information than just the vector from step m of a power iteration. For example, notice that if σ is any shift, then $\mathcal{K}_m(A - \sigma I, q_1) = \mathcal{K}_m(A, q_1)$; that is, Krylov subspaces are *shift invariant*. This is relevant because adding a shift to A can change which of the exterior eigenvalues in the spectrum is dominant. So if $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, then we expect to be able to extract good estimates to λ_1 and λ_n from a few steps of Lanczos iteration under the assumption that power iteration with certain shifted matrices converges sufficiently fast.

Lanczos in inexact arithmetic

The observant reader will have noticed something unsettling about the basic Lanczos iteration: at the heart of the Lanczos iteration is a Gram-Schmidt orthogonalization step, which we said in our discussion of QR factorizations could become numerically unstable. In particular, when $r_{k+1} = (A - \alpha_k)q_k + q_{k-1}\beta_{k-1}$ is "small" (i.e. when β_k is small relative to $\|A\|$), cancellation may cause the computed vector to consisty mostly of roundoff, so that it is no longer orthogonal to the preceding vectors. But a small value of β_k corresponds exactly to convergence to a good approximation to an invariant subspace! Thus, in floating point arithmetic, the basic Lanczos iteration tends to run reasonably well until convergence, at which point it effectively restarts with a "random" starting vector which is made up primarily of roundoff error. So we get convergence of the Ritz values of the computed T to the first few exterior eigenvalues; and then more "ghost" Ritz values converge to the first few exterior eigenvalues; and we continue on in this fashion for as long as we are willing to let things run. This not state of affairs is not entirely satisfactory, but we can fix it by *re-orthogonalization*: that is, we improve the orthogonality of the computed basis by explicitly orthogonalizing Aq_k against vectors to which it should already be orthonormal in exact arithmetic. One approach is to use *complete* orthogonalization: compute Aq_j and then orthogonalize against all previous vectors (e.g. by a sequence of Householder transformations). This is expensive in terms of storage (and data movement) and arithmetic, but it does work. A less expensive solution, *selective* orthogonalization, involves orthogonalizing against converged eigenvector estimates (Ritz vectors). In the interest of time, we will not discuss reorthogonalization further. If you are interested, though — or if you are interested in any number of other aspects of Lanczos iteration that we skip over — I recommend looking at Parlett's book *The Symmetric Eigenvalue Problem*. Stewart's book *Matrix Algorithms, Vol 2: Eigensystems* also has a nice treatment.

Why did numerical instability not arise when we discussed CG? It's there behind the scenes, and it explains why running CG for n steps generally does not provide the exact answer to a linear system. But nobody sensible uses CG as a direct method anyhow; it is used as an iterative solver, and it turns out that the loss of orthogonality does not matter that much to the CG iteration continuing to make progress.

Partial tridiagonalization and residual bounds

Suppose $AQ = QT$, where $T_{ii} = \alpha_i$, $T_{i,i+1} = \beta_i$. If we take m steps of Lanczos iteration, we generate $Q_{:,1:m} = \begin{bmatrix} q_1 & q_2 & \dots & q_m \end{bmatrix}$ as well as the first m coefficients $(\alpha_i)_{i=1}^m$ and $(\beta_i)_{i=1}^m$. Let us denote Q_m and T_m as the leading m columns of Q and the leading $m \times m$ submatrix of T respectively; then writing the first m columns of AQ and QT gives us

$$AQ_m = Q_m T_m + \beta_m q_{m+1} e_m^T. \tag{1}$$

Here T_m is a block Rayleigh quotient $T_m = Q_m^T A Q_m$, and the eigenvalues of T_m (the *Ritz values*) are used to approximate the eigenvalues of A. Now consider the eigendecomposition $T_m = Y \Theta Y^T$ where $Y^T Y = I$ and $\Theta = \mathrm{diag}(\theta_1, \dots, \theta_m)$. Then postmultiplying (1) by Y gives

$$AQ_m Y = Q_m Y \Theta + \beta_m q_{m+1} e_m^T Y.$$

The columns of $Z = Q_m Y = \begin{bmatrix} z_1 & \dots & z_m \end{bmatrix}$ are approximate eigenvalues corresponding to the approximate eigenvalues $\theta_1, \theta_2, \dots, \theta_m$. For each column, we have

$$Az_k - z_k \theta_k = \beta_m q_{m+1} e_m^T y_k,$$

which means

$$\|Az_k - z_k \theta_k\|_2 = |\beta_m| |e_m^T y_k|.$$

This is useful because, as we discussed before, in the symmetric case a small residual error implies a small distance to the closest eigenvalue. This is also useful because the residual error can be computed with no further matrix operations — we need only to look at quantities that we would already compute in the process of obtaining the tridiagonal coefficients and the corresponding Ritz values. In particular, note that we can compute the residual for the Ritz pair (approximate eigenpair) (z_k, θ_k) *without* explicitly computing z_k!

The polynomial connection

Suppose $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ are the eigenvalues of A, with corresponding eigenvectors w_1 through w_n. In matrix form, then,

$$A = W \Lambda W^T.$$

Now, suppose we run m steps of Lanczos iteration (in exact arithmetic), and let θ_1 be the largest eigenvector of T_m (the largest Ritz value). This is the same as saying that θ_1 maximizes the Rayleigh quotient over the Krylov subspace $\mathcal{K}_m(A, q_1)$:

$$\theta_1 = \max_{y \in \mathbb{R}^m} \frac{y^T T_m y}{y^T y} = \max_{y \in \mathbb{R}^m} \frac{(Q_m y)^T A (Q_m y)}{(Q_m y)^T (Q_m y)} = \max_{z \in \mathcal{K}_m(A,q_1)} \frac{z^T A z}{z^T z}.$$

Now, any vector $z \in \mathcal{K}_m(A, q_1)$ can be written as

$$z = c_0 q_1 + c_1 A q_1 + \ldots + c_{m-1} A^{m-1} q_1 = p(A) q_1,$$

where p is a polynomial of degree at most $m-1$. Thus,

$$\theta_1 = \max_{\deg(p)<m} \frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1}.$$

This is still somewhat awkward, so let us rewrite things in terms of the eigenvector basis. Define $d = Z^T q_1$; then

$$\theta_1 = \max_{\deg(p)<m} \frac{d^T p(\Lambda) \Lambda p(\Lambda) d}{d^T p(\Lambda)^2 d} = \max_{\deg(p)<m} \frac{\sum_{i=1}^n d_i^2 p(\lambda_i)^2 \lambda_i}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2}.$$

Let's give a name to the function in this maximization:

$$\phi(p) = \frac{\sum_{i=1}^n d_i^2 p(\lambda_i)^2 \lambda_i}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} = \lambda_1 - \frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2 (\lambda_1 - \lambda_i)}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} \geq \lambda_1 - (\lambda_1 - \lambda_n) \frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2}{d_1^2 p(\lambda_1)^2}.$$

We know that $\theta_1 = \max_{\deg(p)<m} \phi(p) \leq \lambda_1$; we would also like a *lower* bound on θ_1. If we can show that this lower bound approaches λ_1 at some rate with increasing m, then we know θ_1 will converge to λ_1 at least as fast.

Note that if we found a polynomial that was *zero* at $\lambda_2, \ldots, \lambda_n$ and nonzero at λ_1, then we would recover λ_1 exactly. But such a polynomial usually has too high a degree. What we would like in order to get a bound, then, is a degree m polynomial which is not too big on $[\lambda_2, \lambda_n]$, but is relatively large at λ_2. Having seen it's appearance in our analysis of Chebyshev iteration and CG, you will not be surprised that a good candidate is our old friend, the rescaled *Chebyshev polynomial*.

$$\hat{T}_k(x) = T_k\left(-1 + 2\frac{x - \lambda_n}{\lambda_2 - \lambda_n}\right).$$

Note that $|\hat{T}_k(\lambda_j)| \leq 1$ for $j > 1$ and $|\hat{T}_k(\lambda_1)|$ grows rapidly with k.

Note that for any p, we have

$$\begin{aligned}\phi(p) &= \lambda_1 - \frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2(\lambda_1 - \lambda_i)}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} \\ &\geq \lambda_1 - (\lambda_1 - \lambda_n)\frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2}{d_1^2 p(\lambda_1)^2}.\end{aligned}$$

Now, note that $\sum_{i=1}^n d_i^2 = \|d\|_2^2 = \|q_1\|_2^2 = 1$, that $\hat{T}_{m-1}(\lambda_i)^2 \leq 1$ for $i = 2, \ldots, n$ and that $\hat{T}_{m-1}(\lambda_1) = Tc_{m-1}(1 + 2\rho_1)$. Thus,

$$\phi(\hat{c}_{m-1}) \geq \lambda_1 - (\lambda_1 - \lambda_n)\frac{\sum_{i=2}^n d_i^2 \hat{T}_{m-1}(\lambda_i)^2}{d_1^2 \hat{T}_{m-1}(\lambda_1)^2} \geq \lambda_1 - (\lambda_1 - \lambda_n)\frac{1 - d_1^2}{d_1^2 T_{m-1}(1 + 2\rho_1)^2}.$$

The entry $d_1 = w_1^T q_1$ is the cosine of the angle ϕ_1 between the first eigenvector w_1 and the starting vector q_1; so we can write $(1 - d_1^2)/d_1^2 = \tan(\phi_1)^2$, which gives us the final bound

$$\lambda_1 \geq \theta_1 = \max_{\deg p < m} \phi(p) \geq \phi(\hat{c}_{m-1})\lambda_1 - \frac{(\lambda_1 - \lambda_n)\tan(\phi_1)^2}{T_{m-1}(1 + 2\rho_1)^2}.$$

where $\rho_1 = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$.

We notice a few things from this bound. The rate of convergence (determined by $T_{m-1}(1 + 2\rho_1)^2$) is strictly better than the rate of convergence for power iteration, which makes sense since we are using strictly more information in the Lanczos iteration than we use in the power iteration (we maximize over an m-dimensional subspace rather than a 1-dimensional space). Also, the trick involved is a good one: instead of reasoning about matrices, reason about polynomials. But, as with our analysis of CG, this bound may be very pessimistic, since it does not take into account the distribution of eigenvalues in $[\lambda_2, \lambda_n]$. For example, if many eigenvalues of A cluster near zero (as happens with certain discretizations of compact operators, for instance), then we might get much better convergence than indicated by a basic bound that works for any distribution of eigenvalues between $[\lambda_2, \lambda_n]$.