Department of Mathematics, University of Utah

Analysis of Numerical Methods I

MTH6610 – Section 001 – Fall 2017

Lecture notes – The QR algorithm

Friday, November 3, 2017

These notes are not a substitute for class attendance. Their main purpose is to provide a lecture overview summarizing the topics covered.

Reading: Trefethen & Bau III, Lectures 28

We have seen that Rayleigh iteration can be an effective way to compute eigenvalues. We describe here an alternative, far more utilized, algorithm for computing eigenvalues: the QR algorithm. Before getting to this algorithm, we take a detour to explain *simultaneous* power iteration. As before, we assume here that A is a real-valued, square, symmetric matrix of size n.

Let $\lambda_1, \ldots, \lambda_n$, and $v_1, \ldots, v_n$ denote the eigenvalues and corresponding eigenvectors of A. We assume that the eigenvalues are ordered in decreasing magnitude. Recall that the method of power iteration uses the product $A^k v$ (for sufficient large k and a random vector v) to compute an approximation to the domainant eigenvector of A. Now consider using power iteration on two vectors v and w for large k, we have

$$v = \sum_{j=1}^{n} c_j v_j, \qquad w = \sum_{j=1}^{m} d_j v_j,$$

$$A^k v = \lambda_1^k \left[c_1 v_1 + c_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \sum_{j=3}^{n} c_j \left(\frac{\lambda_j}{\lambda_1}\right)^k v_j \right] \approx \lambda_1^k c_1 v_1,$$

$$A^k w = \lambda_1^k \left[d_1 v_1 + d_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \sum_{j=3}^{n} d_j \left(\frac{\lambda_j}{\lambda_1}\right)^k v_j \right] \approx \lambda_1^k \left[d_1 v_1 + d_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 \right]$$

Since v_1 is orthogonal to v_2, then the QR factorization of the $n \times 2$ concatentation of these vectors is

$$\left[A^k v \ \ A^k w \right] = [v_1 \ \ v_2] \begin{pmatrix} c_1 \lambda_1^k & d_1 \lambda_1^k \\ 0 & d_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k \end{pmatrix} =: QR$$

We have discovered that by performing power iteration on two linearly independent vectors, we can obtain approximations to the two dominant eigenvectors of A. Repeating this argument, one can conclude that if $V \in \mathbb{R}^{n \times n}$ is any full-rank matrix, then performing a QR decopmosition of $A^k V$ yields

$$A^k V = QR = [v_1 \ \ v_2 \ \ \cdots \ \ v_n] \, R.$$

I.e., the matrix Q contains (approximations to) the eigenvectors of A. This is the method of simultaneous power iteration to find collections of eigenvectors (as opposed to sequential power iteration, using matrix deflation after each eigenvalue is found.)
With some experience in ill-conditioned algorithms, one can reason that $A^k V$ will be a relatively ill-conditioned matrix, so that directly performing a QR decomposition of this matrix is probably not a good idea. Instead, a slightly more intelligent version of this performs a QR decomposition each time A is applied:

$$\begin{aligned}
&Q_0^{PI} = I, \\
&k = 0, 1, \ldots \\
&\qquad A_{k+1}^{PI} \coloneqq A Q_k^{PI} \\
&\qquad A_{k+1}^{PI} \eqqcolon Q_{k+1}^{PI} R_{k+1}^{PI}
\end{aligned}$$

Above, we have chosen the initial matrix V to be the identity I. How do the matrices Q_{k+1}^{PI} and R_{k+1}^{PI} compare to the QR decomposition of A^k? The following result holds:

$$\begin{aligned}
A^k &\eqqcolon Q_k R_k \\
Q_k &= Q_k^{PI} \\
R_k &= R_k^{PI} R_{k-1}^{PI} \cdots R_1^{PI}.
\end{aligned}$$

Thus, this iterated QR method computes the same Q factor, and if we desired the R factor from pure power iteration is also computable.
We are now in a position to introduce the QR algorithm. Given a square matrix A, the (basic) QR algorithm is the following sequence of operations:

$$\begin{aligned}
&A_0^{QR} = A, \\
&k = 0, 1, \ldots \\
&\qquad A_k^{QR} \eqqcolon Q_{k+1}^{QR} R_{k+1}^{QR} \\
&\qquad A_{k+1}^{QR} \coloneqq R_{k+1}^{QR} Q_{k+1}^{QR}
\end{aligned}$$

Noting that

$$A_{k+1}^{QR} = \left(Q_{k+1}^{QR}\right)^T Q_{k+1}^{QR} R_{k+1}^{QR} Q_{k+1}^{QR} = \left(Q_{k+1}^{QR}\right)^T A_k^{QR} Q_{k+1}^{QR},$$

we see that A_k^{QR} is related to A by a unitary similarity transform:

$$A_k^{QR} = \left(Q_1^{QR} \cdots Q_k^{QR}\right)^T A \left(Q_1^{QR} \cdots Q_k^{QR}\right)$$

This fact, that iterating this way is simply unitary similarity transformations, and hence stable, is one reason why this algorithm is attractive.
The surprising fact about this algorithm is that it is essentially a stable way to perform simultaneous power iteration. Indeed:

$$\begin{aligned}
Q_k^{PI} &= \left(Q_1^{QR} \cdots Q_k^{QR}\right) \\
R_k^{PI} &= R_k^{QR}
\end{aligned}$$

This fact can be used to motivate the following conclusion: since the matrix product $\left(Q_1^{QR} \cdots Q_k^{QR}\right) = Q_k^{PI}$ for large k converges to a matrix of eigenvectors, then the matrix A_k^{QR} must converge to a diagonal matrix with the eigenvalues of A on the diagonal.
In fact, a stronger statement holds: if A is *almost* any matrix (complex, non-Hermitian, etc), then A_k^{QR} converges to the Schur factor of A. (Hence, the eigenvalues can be read off the diagonal.)
This is the power of the QR algorithm: it is a stable method for computing the Schur factor (and hence the eigenvalues) of A. The form of this algorithm we have explained above is too expensive to use directly (it requires a large number of QR factorizations). Just as power iteration can be sped up using inverse iteration, so too can the basic QR algorithm be sped up using *shifts*, which is an implicit way of performing inverse iteration and Rayleigh quotient operations.