Department of Mathematics, University of Utah

Analysis of Numerical Methods I
MTH6610 – Section 001 – Fall 2019

Lecture notes – The QR algorithm
Wednesday, November 6, 2019

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. Thus, we have

$$A = V\Lambda V^T,$$

where V is a unitary matrix whose columns are the eigenvalues of A, and Λ is a diagonal matrix containing the eigenvalues.
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

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

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

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

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

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

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 $W \in \mathbb{R}^{n\times n}$ is any full-rank matrix, then performing a QR decomposition of $A^k W$ yields

$$A^k W = QR \approx [v_1 \;\; v_2 \;\; \cdots \;\; v_n]\, R =: VR$$

I.e., the matrix Q contains (approximations to) the eigenvectors V 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 $A^k W$ 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:

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

Above, we have chosen the initial matrix W 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:

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

Thus, this iterated QR method computes the same Q factor, and if we desired the R factor from pure power iteration is also computable. Therefore, for large k we have

$$Q_k^{PI} \approx V \tag{2}$$

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:

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

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:

$$A_k^{QR} = \left(Q_1^{QR} \cdots Q_k^{QR}\right)^T A \left(Q_1^{QR} \cdots Q_k^{QR}\right) \tag{3}$$

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. To see this, we investigate how A^k depends on the QR matrices. For $k = 1$, we have

$$A^1 = Q_1^{QR} R_1^{QR}. \tag{4}$$

We also note by virtue of the QR algorithm, we have the following relationship allowing us to permute Q^{QR} and R^{QR} matrices:

$$R_j^{QR} Q_j^{QR} = Q_{j+1}^{QR} R_{j+1}^{QR}. \tag{5}$$

Now we make the following inductive hypothesis:

$$A^k = \left(Q_1^{QR} Q_2^{QR} \cdots Q_k^{QR}\right) \left(R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR}\right). \tag{6}$$

Then for arbitrary $k \geq 1$:

$$\begin{aligned}
A^{k+1} &= AA^k \overset{(4)}{=} Q_1^{QR} R_1^{QR} A^k \\
&\overset{(6)}{=} Q_1^{QR} R_1^{QR} \left(Q_1^{QR} Q_2^{QR} \cdots Q_k^{QR}\right) \left(R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR}\right) \\
&= Q_1^{QR} \left(R_1^{QR} Q_1^{QR}\right) Q_2^{QR} \cdots Q_k^{QR} R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR} \\
&\overset{(5),j=1}{=} Q_1^{QR} \left(Q_2^{QR} R_2^{QR}\right) Q_2^{QR} \cdots Q_k^{QR} R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR} \\
&= Q_1^{QR} Q_2^{QR} \left(R_2^{QR} Q_2^{QR}\right) Q_3^{QR} \cdots Q_k^{QR} R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR} \\
&\overset{(5),j=2}{=} Q_1^{QR} Q_2^{QR} \left(Q_3^{QR} R_3^{QR}\right) Q_3^{QR} \cdots Q_k^{QR} R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR} \\
&\vdots \\
&\overset{(5),j=k}{=} \left(Q_1^{QR} Q_2^{QR} \cdots Q_{k+1}^{QR}\right) \left(R_{k+1}^{QR} R_k^{QR} \cdots R_1^{QR}\right)
\end{aligned}$$

This completes the inductive step, proving (6). We have thus computed a QR decomposition of A^k. Comparing this QR decomposition of A^k with that given by (1), we conclude:

$$\begin{aligned}
Q_k^{PI} &= Q_1^{QR} \cdots Q_k^{QR} \\
R_k^{PI} R_{k-1}^{PI} \cdots R_1^{PI} &= R_k^{QR} R_{k-1}^{QR} \cdots R_1^{QR}.
\end{aligned} \tag{7}$$

Thus, (7) implies that the product of Q^{QR} matrices can be used to compute Q_k^{PI}, which by (2) approximates the eigenvectors V.
This is an interesting result, but a more surprising result is true: Using (7) and (2) in (3), we conclude:

$$A_k^{QR} \approx V^T A V = \Lambda.$$

Thus, the QR algorithm produces iterates A_k^{QR} that converge to Λ. Furthermore, these iterates are computed from A by a sequence *unitary* (well-conditioned!) similarity transforms. This is the power of the QR algorithm: a methodical procedure with *unitary* transformations that reveals the spectrum of A.

In fact, a stronger statement holds: if A lies in a more general class of matrices (which includes those that are complex-valued, non-Hermitian, etc), then A_k^{QR} converges to the Schur factor of A. Hence, the eigenvalues can be read off the diagonal, and the QR algorithm essentially computes the Schur decomposition of A.

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.