Dr H. Fawzi

Mathematical Tripos Part II: Michaelmas Term 2023

Numerical Analysis – Lecture 23

QR iteration with shifts In the last lecture we introduced simultaneous iteration as a generalization of the power method to multiple orthogonal vectors. When the number of such vectors is $p = n$ (the dimension of the space), we saw that simultaneous iteration can also be seen as a generalization of inverse iteration. More precisely, we saw that if $X^{(k)}$ is the sequence of orthogonal matrices produced by simultaneous iteration, then

$$X_1^{(k)} = \frac{A^k X_1^{(0)}}{\|A^k X_1^{(0)}\|_2} \qquad \text{and} \qquad X_n^{(k)} = \frac{A^{-k} X_n^{(0)}}{\|A^{-k} X_n^{(0)}\|_2}.$$

We know from Lecture 21 that the convergence of inverse iteration can be significantly improved if we update the shift s at each iteration, such as in the Rayleigh Quotient Iteration. This motivates us to consider the following shifted version of simultaneous iteration.

> SHIFTED SIMULTANEOUS ITERATION
> Let $X^{(0)} = I$
> For $k = 0, 1, 2, \ldots$
> - Compute shift s_k (eg $s_k = (X_n^{(k)})^T A X_n^{(k)}$)
> - $Y = (A - s_k I) X^{(k)}$
> - $[X^{(k+1)}, R] = \text{qr}(Y)$

As mentioned in the previous lecture, this algorithm can be rewritten in terms of the matrices $A^{(k)} = (X^{(k)})^T A X^{(k)}$ instead of $X^{(k)}$.

> SHIFTED QR ITERATION
> Let $A^{(0)} = A$
> For $k = 0, 1, 2, \ldots$
> - Compute shift s_k (e.g., $s_k = A_{nn}^{(k)}$)
> - $[Q, R] = \text{qr}(A^{(k)} - s_k I)$
> - $A^{(k+1)} = Q^T A^{(k)} Q = RQ + s_k I$

One can prove the formal equivalence between these two algorithms in exactly the same way it was done in Lecture 22, via induction. Note that the matrix $X^{(k)}$ in simultaneous iteration can be obtained as the product of the orthogonal matrices Q in the QR iteration.

Using the shifting strategy above, we expect the last row of $X^{(k)}$ to converge very quickly to an eigenvector of A; equivalently, this means that the last row of $A^{(k)} = (X^{(k)})^T A X^{(k)}$ converges very quickly to the vector $(0, \ldots, 0, \lambda)$ where λ is an eigenvalue of A. Once we have convergence, the matrix $A^{(k)}$ becomes block diagonal, i.e., it can be written as

$$A^{(k)} = \begin{bmatrix} & & & 0 \\ & \hat{A} & & \vdots \\ & & & 0 \\ 0 & \ldots & 0 & \lambda \end{bmatrix}.$$

In this case, we need only focus on the matrix $\hat{A}$ which is of size $(n-1) \times (n-1)$. This is the idea of *deflation*, and leads us to the following algorithm. We use the convenient Matlab-style notations $1 : k$ for the set $\{1, \ldots, k\}$, and $M[I, J]$ to be the submatrix with row indices I and column indices J.

QR ITERATION WITH SHIFTS AND DEFLATION
Input: symmetric matrix A_0
Initialize $A = A_0$ (upon termination, A will hold the eigenvalues of A_0)
Initialize $X = I_n$ (upon termination, X will hold the matrix of eigenvectors)
For $j = n, n-1, \ldots, 2$

- While $\|A[j, 1:j-1]\| \geq \epsilon$ (i.e., while $A[j, 1:j-1]$ is "numerically" nonzero)
  - Let $s = A_{jj}$ (shift)
  - $[Q, R] = \mathrm{qr}\,(A[1:j, 1:j] - sI_j)$
  - $A[1:j, 1:j] = RQ + sI_j$
  - $X = X \cdot \begin{bmatrix} Q & 0 \\ 0 & I_{n-j} \end{bmatrix}$ (update X)

Upon termination of the algorithm, the matrix A has been reduced to a diagonal matrix containing the eigenvalues, and the matrix X contains the eigenvectors of A_0, so that $A_0 = XAX^T$.

Remark 5.15 *In the above algorithm we always deflate the last row/column of the matrix for simplicity, and because it is the one that generally has the fastest convergence. However in practice it is useful to check for other rows/columns that can also be deflated, i.e., other rows i such that $|A_{ij}| \leq \epsilon$ for $j \neq i$.*

Reduction to tridiagonal matrices Computing a QR factorization of a $n \times n$ matrix requires $\approx n^3$ floating point operations. If the algorithm above performs a QR factorization for each $j = n, \ldots, 2$ then the cost of the algorithm scales like n^4.

To remedy this high computational cost, one first starts by putting A into *tridiagonal form* by an orthogonal transformation, before calling the QR iteration algorithm. Recall that a symmetric matrix A is tridiagonal if $A_{ij} = 0$ whenever $|i - j| > 1$. There are two reasons why tridiagonal structure is advantageous:

- Computing the QR factorization of a symmetric tridiagonal matrix can be done in $O(n)$ operations, using Givens rotations.
- The QR iterations preserve the tridiagonal structure.

We start by proving the second point:

Proposition 5.16 *Assume that A is a $n \times n$ symmetric tridiagonal matrix, and consider one step of shifted QR iteration: $A^+ = RQ + sI$ where $[Q, R] = \mathrm{qr}(A - sI)$. Then A^+ is symmetric tridiagonal.*

Proof. Since $A - sI$ is tridiagonal, it is easy to verify that $Q_{ij} = 0$ if $i > j + 1$.[1] It thus follows that $(A^+)_{ij} = (RQ + sI)_{ij} = 0$ if $i > j + 1$. Since A^+ is symmetric we must also have $(A^+)_{ij} = 0$ if $j > i + 1$. This means that A^+ is tridiagonal. $\square$

Proposition 5.17 *The QR factorization of a $n \times n$ symmetric tridiagonal matrix A can be computed in $O(n)$ operations using Givens rotations.*

Sketch of proof. We apply sequentially Givens rotation matrices $\Omega^{[i,i+1]}$ that annihilate the $(i, i+1)$ entry below the diagonal. After applying $n - 1$ such rotation matrices we arrive at the upper triangular matrix R. Note that applying a single Givens rotation matrix requires a constant number of floating point operations since A is tridiagonal and has only at most 3 nonzero elements per row. Thus the total cost of the algorithm is $O(n)$. Schematically:

$$A = \begin{bmatrix} * & * & 0 & 0 \\ * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} \overset{\Omega^{[1,2]}\times}{\rightarrow} \begin{bmatrix} \bullet & \bullet & \bullet & 0 \\ \mathbf{0} & \bullet & \bullet & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix} \overset{\Omega^{[2,3]}\times}{\rightarrow} \begin{bmatrix} * & * & * & 0 \\ 0 & \bullet & \bullet & \bullet \\ 0 & \mathbf{0} & \bullet & \bullet \\ 0 & 0 & * & * \end{bmatrix} \overset{\Omega^{[3,4]}\times}{\rightarrow} \begin{bmatrix} * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & \bullet & \bullet \\ 0 & 0 & \mathbf{0} & \bullet \end{bmatrix} = R.$$

[1] Indeed, since the jth column of Q is a linear combination of the columns $1, \ldots, j$ of $A - sI$, and since $A - sI$ is tridiagonal, we get that $Q_{ij} = 0$ for $i > j + 1$.

The '$\bullet$' indicate the entries that get modified at each iteration. Note that the resulting upper triangular R satisfies $R_{ij} = 0$ when $i < j - 2$. $\square$

In the above algorithm we do not explicitly form Q but we only keep track of the Givens rotation matrices $\Omega^{[1,2]}, \ldots, \Omega^{[n-1,n]}$. Computing the product $RQ = R(\Omega^{[1,2]})^T \cdots (\Omega^{[n-1,n]})^T$ can be done in $O(n)$ time since we know already from Proposition 5.16 that the resulting matrix RQ is tridiagonal:

$$R = \begin{bmatrix} * & * & * & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix} \overset{\times(\Omega^{[1,2]})^T}{\rightarrow} \begin{bmatrix} \bullet & \bullet & * & 0 \\ \bullet & \bullet & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix} \overset{\times(\Omega^{[2,3]})^T}{\rightarrow} \begin{bmatrix} * & \bullet & \mathbf{0} & 0 \\ * & \bullet & \bullet & * \\ 0 & \bullet & \bullet & * \\ 0 & 0 & 0 & * \end{bmatrix} \overset{\times(\Omega^{[3,4]})^T}{\rightarrow} \begin{bmatrix} * & * & 0 & 0 \\ * & * & \bullet & \mathbf{0} \\ 0 & * & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet \end{bmatrix} = RQ.$$