Equivalent of Section 5.6: the QR iteration. The Francis algorithm is too complex to be presented as the first significant eigenvalue-calculating algorithm, so we will instead offer a simpler version of it in the QR iteration.

The QR iteration is a generalization of the power method. In the power method, we apply an iteration that roughly works like this:

\[ q_{k+1} r_k = A q_k , \]

by scaling by \( r_k \) (which we sometimes called \( s_k \)) and multiplying with \( A \) until \( q_k \) approaches the eigenvector corresponding to the unique largest absolute value eigenvalue.

We will now extend this to the iteration

\[ U_{k+1} R_k = A U_k , \]

with \( U_k \) a matrix approaching a matrix of eigenvectors (if the largest \( p \) eigenvalues are all distinct in absolute value, then the first \( p \) columns of \( U_k \) approach a basis of the space determined by the \( p \) eigenvectors corresponding to those eigenvalues).

Therefore we seek to find a sequence of \( U_k \)’s such that \( U_k^* A U_k \) approaches \( T \), with \( A = U T U^* \) being the Schur decomposition of \( A \) (and \( T \) being upper triangular, and, for consistency, ordered so that \( T_{11} = \lambda_1, \ldots, T_{pp} = \lambda_p \)).

So, we are looking to transform our recurrence into a sequence \( A_k = U_k^* A U_k \) matrices which converge to upper triangular form.

It turns out that, with this definition of \( A_k \), we have a pretty simple and quite fast way of constructing \( A_{k+1} \) from \( A_k \) without even computing \( U_k \) explicitly.

Let us examine \( A_k = U_k^* A U_k \). As per the recurrence, this means \( A_k = U_k^* (U_{k+1} R_k) = (U_k^* U_{k+1}) R_k \).

Denote by \( Q_k = U_k^* U_{k+1} \), and note now that \( A_k = Q_k R_k \) is the/a full QR decomposition of \( A_k \).

The above is a key insight, and the reason why the method is called “the QR iteration.”

On the other hand, note that \( U_{k+1} R_k = A U_k \) means that, by multiplication with \( U_{k+1}^* \) to the left and by \( U_k^* \) to the right, \( R_k U_{k+1}^* = U_k^* A \). Replacing this in \( A_{k+1} = U_{k+1}^* A U_{k+1} \), we obtain that

\[ A_{k+1} = R_k U_{k+1}^* U_{k+1} = R_k (U_k^* U_{k+1}) = R_k Q_k , \]

by the definition of \( Q_k \).

So, \( A_{k+1} \) is defined by taking the full QR decomposition of \( A_k \) and multiplying \( Q_k \) and \( R_k \) in the reverse order.

The entirety of the QR method idea rests in the following two-line recurrence:

\[ [Q_k, R_k] = \text{qr}(A_k) ; \]
\[ A_{k+1} = R_k Q_k . \]
We will of course have to define some parameters for where to start the recurrence (initial conditions) and for how long to run it (for example, until
\[ \sqrt{\sum_{i>j} |A_{k+1}(i,j)|^2} < \epsilon \]
that is, the lower triangular part of \( A_{k+1} \) is smaller in Frobenius norm than some pre-set tolerance \( \epsilon \). But the gist of it is encompassed in the two lines (1), (2).

**A smart start.** We have mentioned the Hessenberg form before; since we have access (without too much additional computational costs) to a matrix \( Q \) which defines a similarity transformation \( A = QHQ^* \) where \( H \) is upper Hessenberg, and so a big step already in the direction of making the lower triangular part of \( A_k \) really small, we could start with \( A_1 = H \).

The sharp student may be skeptical at this point. All right, she may say, we start with a Hessenberg matrix \( A_1 \), but what’s the point if at the next step, \( A_2 \) is once again full? Have we truly gained something?

The answer is a surprising and resounding “yes!” And that is given by the lemma below.

**Lemma 1.** If \( A_k \) is Hessenberg, \( A_{k+1} \) is Hessenberg.

In other words, the QR iteration does not introduce any new zeros. This, as we shall see, is extremely important.

**Proof.** This is just a sketch, meant to give you a good idea of what happens, rather than a complete proof.

Recall that \( A_k \) is Hessenberg. How do we compute a full QR factorization for a Hessenberg matrix?

Recall that one way the full QR factorization can be computed is via Householder reflectors. Column-by-column, these create zeros under the diagonal of the matrix, and \( Q \) is found by multiplying them.

Let us examine what happens in the case of a Hessenberg matrix \( A_k \):

\[
A_k = \begin{bmatrix}
  a_1 & * & * & *
  
  b_1 & * & * & *
  
  0 & * & * & *
  
  0 & 0 & * & *
\end{bmatrix},
\]

so the first reflector’s job is simply to map \( x = [a_1, b_1]^T \) into \( [\|x\|_2, 0]^T \). We have seen several times now that such a thing is done by using a rotation \( \begin{bmatrix}
\cos(\theta_1) & \sin(\theta_1) \\
\sin(\theta_1) & -\cos(\theta_1)
\end{bmatrix} \), for a suitable \( \theta_1 \). So, the first reflector will be built from this, and leaving the rest of the rows alone:

\[
H_k^{(1)} = \begin{bmatrix}
\cos(\theta_1) & \sin(\theta_1) & 0 & 0 \\
\sin(\theta_1) & -\cos(\theta_1) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.
\]

After applying \( H_k^{(1)} \), we obtain that

\[
H_k^{(1)} A_k = \begin{bmatrix}
\ast & \ast & \ast & \ast \\
0 & a_2 & \ast & \ast \\
0 & b_2 & \ast & \ast \\
0 & 0 & \ast & \ast
\end{bmatrix}.
\]
Next, we will map \( x = [a_2, b_2]^T \) into \( ||x||_2, 0 \|^T \) by means of another rotation
\[
\begin{bmatrix}
\cos(\theta_2) & \sin(\theta_2) \\
\sin(\theta_2) & -\cos(\theta_2)
\end{bmatrix},
\]
for a suitable \( \theta_2 \), with
\[
H_k^{(2)} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\theta_2) & \sin(\theta_2) & 0 \\
0 & \sin(\theta_2) & -\cos(\theta_2) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}.
\]
This leads to
\[
H_k^{(2)} H_k^{(1)} A = \begin{bmatrix}
\Box & \Box & \Box & \Box \\
0 & 0 & a_3 & \Box \\
0 & 0 & b_3 & \Box
\end{bmatrix}
\]
Finally, construct a \( H_k^{(3)} \) using a rotation that takes \( x = [a_3, b_3]^T \) into \( ||x||_2, 0 \|^T \), and obtain that
\[
H_k^{(3)} H_k^{(2)} H_k^{(1)} A = \begin{bmatrix}
\Delta & \Delta & \Delta & \Delta \\
0 & \Delta & \Delta & \Delta \\
0 & 0 & \Delta & \Delta \\
0 & 0 & 0 & \Delta
\end{bmatrix};
\]
at this point, for this 4 \( \times \) 4 matrix, this is enough.

For an \( n \times n \) matrix, we can construct the \( Q_k = (H_k^{(1)})^d (H_k^{(2)})^d \ldots (H_k^{(n-1)})^d \) similarly, and note that \( R_k \) is what we wanted: upper triangular.

This, however, is just half the work. As we must now assemble \( A_{k+1} \), we need to multiply \( Q_k \) into \( R_k \), to the right.

Examine \( R_k (H_k^{(1)})^d \):
\[
R_k (H_k^{(1)})^d = \begin{bmatrix}
\Delta & \Delta & \Delta & \Delta \\
0 & R_k(2, 2) & \Delta & \Delta \\
0 & 0 & \Delta & \Delta \\
0 & 0 & 0 & \Delta
\end{bmatrix} \begin{bmatrix}
\cos(\theta_1) & \sin(\theta_1) & 0 & 0 \\
\sin(\theta_1) & -\cos(\theta_1) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}^d = \begin{bmatrix}
\Box & \Box & \Box & \Box \\
0 & R_k(2, 2)\sin(\theta_1) & \Box & \Box \\
0 & 0 & 0 & \Box \\
0 & 0 & 0 & \Box
\end{bmatrix};
\]
indeed, the only previous zero in the lower triangular part of the matrix that is going to be “filled in” will be in position (2, 1).

Similarly, at the next multiplication step by \( (H_k^{(2)})^d \), we fill in one more 0 in position (3, 2), and so on... by the time we have finished the multiplication, \( A_{k+1} \) is Hessenberg.

This can of course be shown in full generality, for an \( n \times n \) matrix.

Finally, why should we care that the matrix is Hessenberg at every step? The reason is simple: much lower computational costs.

Consider starting the QR iteration at Hessenberg form \( A_1 = H \). Computing QR via the kind of rotation shown above takes \( n - 1 \) steps, each involving \( O(n) \) operations (it modifies only 2 rows of the matrix!) for a total \( O(n^2) \) flop count. Similarly, multiplying the rotations back into \( R_k \) will also take \( O(n^2) \) flops. \textit{You will have to argue this completely for your homework.}

The flop count for the Hessenberg reduction is \( 10/3 n^3 \). If we apply the iteration \( k \) times, this brings the total count to \( 10/3n^3 + O(kn^2) \).
Let us now see what the count would be if we start at a full $A_1$. Each iteration would be $O(n^3)$, as both full $QR$ and matrix-matrix multiplication normally take that many operations. So after $k$ iterations, the cost would be $O(\frac{k}{3}n^3)$.

A flop count of $10/3 \ n^3 + O(kn^2)$, versus one of $O(kn^3)$: the two-step idea of first constructing the Hessenberg form of the matrix, and then applying the $QR$ iteration starting there, is MUCH more efficient that starting the $QR$ iteration at a full matrix. Note: $k$ is not likely to be small, like 3; it is much more likely to be on the order of tens or hundreds... so the saving is indeed huge.